UNIVERSITÉ DE LORRAINE | BIBLIOTHÈQUES UNIVERSITAIRES

AVERTISSEMENT
DROIT D'AUTEUR

Ce document est le fruit d'un long travail approuvé par le jury de soutenance et mis à disposition de l'ensemble de la communauté universitaire élargie.

Il est soumis à la propriété intellectuelle de l'auteur. Ceci implique une obligation de citation et de référencement lors de l'utilisation de ce document. Toute contrefaçon, plagiat ou reproduction illicite encourt une poursuite pénale.

Contact bibliothèque : ddoc-theses-contact@univ-lorraine.fr
(Cette adresse ne permet pas de contacter les auteurs)

Code de la Propriété Intellectuelle. articles L 122. 4
Code de la Propriété Intellectuelle. articles L 335.2- L 335.10

UNIVERSITÉ DE LORRAINE
34 COURS LÉOPOLD — CS 25233
54052 NANCY CEDEX
TÉL : 03 54 50 54 00
CONTACT@UNIV-LORRAINE.FR
WWW.UNIV-LORRAINE.FR

Graph Reinforcement Learning for Automated Attack Surface Discovery in Networked Systems

Ph.D. Thesis

presented and publicly defended on October 9, 2026

for obtaining the

Doctorate from the Université de Lorraine

(mention informatique)

by

Franco Terranova

Jury composition

President :	Bernardetta Addis	Professor, Université de Lorraine
Reviewers :	Anna Sperotto	Professor, University of Twente
	Marco Anisetti	Professor, Università degli Studi di Milano
Examiners :	Bernardetta Addis	Professor, Université de Lorraine
	Julien Francq	Head of Research & Innovation, Naval Group (CERT)
PhD supervisor :	Isabelle Chrisment	Senior Researcher, INRIA
Co-supervisor :	Abdelkader Lahmadi	Professor, Universite de Lorraine

Laboratoire Lorrain de Recherche en Informatique et ses Applications — UMR 7503

Mis en page avec la classe thesul.

Apprentissage par renforcement sur graphes pour la découverte automatisée de surfaces d'attaque réseau

THÈSE

présentée et soutenue publiquement le 9 octobre 2026

pour l'obtention du

Doctorat de l'Université de Lorraine

(mention informatique)

par

Franco Terranova

Composition du jury

Président :	Bernardetta Addis	Professeure, Université de Lorraine
Rapporteurs :	Anna Sperotto	Professeure, University of Twente
	Marco Anisetti	Professeur, Università degli Studi di Milano
Examinateurs :	Bernardetta Addis	Professeure, Université de Lorraine
	Julien Francq	Responsable Recherche & Innovation, Naval Group (CERT)
Directrice de thèse :	Isabelle Chrisment	Directrice de recherche, INRIA
Co-directeur :	Abdelkader Lahmadi	Professeur, Université de Lorraine

Remerciements

At the end of these three years, I find myself here, trying to thank the people who have accompanied me along the way, and not quite knowing where to begin. There is far too much to say, and far too many people to mention. A PhD is often presented as an individual journey, but looking back, I realise how deeply it has been shaped by the people I met, the places I visited, the conversations I had, and the moments, both difficult and joyful, that I shared with others.
This journey was lived differently from many other chapters of my life. I began it with few expectations and little planning, mostly taking things as they came, one opportunity and one decision at a time. Along the way, I met people from many backgrounds and cultures, discovered parts of the world I had never known, and learned to see both the world and myself differently. Sometimes in better ways, sometimes in more difficult ones, but always in ways that changed me.
These acknowledgements are therefore an attempt to recognise the people who became particularly important during these years. Many others could be mentioned, from the first person I met during this PhD to the last. To all of them, whether their names appear here or not, thank you.
First and foremost, thank you to my family. You have supported me throughout this opportunity, just as you have supported me in every other step of my life. You supported my relocation, trusted my choices, and believed in whatever I decided to pursue. In particular, thank you to my mother, Nunzia; my sister, Alisia; and my grandparents, Salvatore, Maria, Grazia, and Franco. A special place in these acknowledgments is reserved for Salvatore, whose memory will always be part of my journey. Thank you also to Nuvola, my dog, who always waits for me when I come home, even if those returns are becoming less frequent.
Thank you to LORIA, the laboratory that hosted me throughout my PhD. Thank you for the office, the corridors where so many unexpected conversations happened, and, above all, for providing an environment in which I could work, grow, and meet so many wonderful people.
A special thank you goes to Professors Lahmadi and Chrisment, who were exceptional supervisors. Thank you for trusting my choices, giving me the freedom I needed to explore my ideas, and supporting my research throughout these years. I could not have asked for better supervisors for the kind of PhD I wanted to pursue.
Thank you to the amazing RESIST team, in all its different forms and evolutions throughout these years. In many ways, this team became my family away from home.
Omar, thank you for being such a great office mate and for bringing so much energy, humour, and good vibes to the team. Wafik, thank you for showing me what genuine kindness means, and for reminding me what it means to remain strong. Philippe and Zozo, thank you for being such important companions during the first years. And thank you to those who became an important part of the later years: Sebastian, for always improving the vibes; Santiago, for showing us how even pain can be fought with humour; Reza, for giving us someone to attack; and Aurélie, for the countless funny moments.
Thank you to Sana, who gave me my first real opportunity to supervise someone and introduced me to a wonderful human being. Thank you to Matthews for all the coffee-break conversations, to Amine for the great moments at events and our discussions about the future, and to all the others who shared this journey with me: Ahmed, Ahmed, Joel, Mael, Gaelle, Rémi, and Sasha.
I am also grateful for the opportunities this PhD gave me to work and live abroad. Thank you to Professor Limam for his hospitality at the University of Waterloo and for the work we carried out together. A very special thank you goes to Professor Cabellos, who hosted my mobility at UPC Barcelona, believed in my project, and helped me discover not only an incredible academic environment but also an incredible city. Thank you to Guillermo and Professor Miolane, from

UC Santa Barbara, for believing in the project and supporting me throughout our attempt at my first AI conference paper.
Thank you to Professor Kondo for the countless conversations, ideas, and plans we made together, and for the unforgettable experiences during our stays at the University of Tokyo and in Japan. Those moments were truly special. Thank you also to Frédéric, the engineer of our laboratory, for all the breakfasts we shared and for the time spent discussing almost every possible aspect of life.
Thank you as well to the many other wonderful people I met through the laboratory. Fatou, Wissal, and Khensa, thank you for turning random kitchen and bathroom breaks, jokes, and everyday conversations into amazing moments. What started as small breaks during the day became genuine friendships. Franck, thank you for all the "ça va?" moments, your kindness, and for becoming a good friend.
I am also grateful for the doctoral events and for all the people I met through them, many of whom eventually became some of my closest friends during these years.
Niloufar, thank you for being there since the very first UL event, when you were still a little shy, and for the friendship and support that grew from there. You became probably the person I can rely on most in this city, and I am deeply grateful for that.
Sushant, thank you for being the social machine that you are, but even more for the person behind it: someone who would go to extraordinary lengths to help another person, and who has done so for me more than once. Thank you also to Sher, whose laughter and chill vibes made so many trips during this PhD better. Thank you to Krishna, Majideh, Aditi, Smitkriti, Chaimae, and Boutaina for the company, conversations, and memories across trips.
Thank you to the people involved in the Superviz project for the interesting meetings, in particular Ludovic and Pierre-François.
Thank you also to Professors Addis and Scherrer for being part of my CSI committee, for the time you dedicated to assessing my work, and for following my progress throughout these years.
Thank you to Samantha and to the Italian group I met in Nancy for helping me feel a little closer to home. Claudia, Francesco, Giulia, and all the others, thank you for all the small things that made being far from home feel a little less distant.
And thank you to the people from home and from earlier chapters of my life who, despite the distance and the passage of time, remained by my side. In particular, thank you to Professor Ivana Ferraro, for her enormous support, her constant presence, and her guidance throughout the years. Thank you also to Giorgia, Danzika, Lorenzo, Lidia, Gloria, and Emanuele, for remaining an important part of my life.
Finally, I thank myself for waking up every morning and doing the work, even on the days when motivation was nowhere to be found. Thank you for taking one step after another, for dealing with uncertainty, failures, and moments of doubt, and for continuing anyway.
This PhD was made of all those small steps. None of them was taken entirely alone.
To everyone who was part of this journey, in whatever way, thank you.

We shall not cease from exploration
And the end of all our exploring
Will be to arrive where we started
And know the place for the first time.

Non smetteremo di esplorare
e la fine di tutta la nostra esplorazione
sarà arrivare dove abbiamo iniziato
e conoscere il luogo per la prima volta.

Nous ne cesserons pas d'explorer,
et au terme de toutes nos explorations,
nous reviendrons là où nous avons commencé,
et nous connaîtrons le lieu pour la première fois.

— T. S. Eliot, "Little Gidding," Four Quartets

CONTENTS

Introduction

Existing challenges 2
List of contributions 3
Thesis organization 7
Context 7

Part I Background and related work

Chapter 1
Vulnerability data sources

1.1 Introduction 13
1.2 Vulnerability databases 14
1.2.1 Common vulnerabilities and exposures 14
1.2.2 Exploit database. 16
1.2.3 Vulnerability disclosure process 16
1.3 Common vulnerability scoring system 18
1.4 MITRE frameworks 20
1.4.1 Common weakness enumeration 20
1.4.2 Common attack pattern enumeration and classification 21
1.4.3 Adversarial tactics, techniques, and common knowledge 22
1.5 Cyber-threat intelligence 23
1.6 Filling the gaps through automation 24
1.7 Summary 25

Chapter 2
Automated attack path analysis

2.1 Introduction 27

2.2 Attack graphs . . . 29
2.2.1 Attack graph representations . . . 30
2.2.2 Logical attack graphs . . . 31
2.2.3 Required knowledge and inputs . . . 31
2.2.4 Remaining challenges . . . 32
2.3 From exhaustive to critical attack paths . . . 32
2.4 Critical attack path discovery as a combinatorial optimization . . . 34
2.5 From exact optimization to heuristic search . . . 36
2.6 Learning paradigms . . . 37
2.6.1 Supervised learning . . . 37
2.6.2 Unsupervised learning . . . 37
2.6.3 Reinforcement learning . . . 38
2.7 Heuristic learning approaches . . . 39
2.7.1 One-shot optimization . . . 39
2.7.2 Sequential optimization . . . 40
2.8 Existing heuristics solutions . . . 41
2.9 Summary . . . 43

Part II A learning framework for attack path planning

Chapter 3
A POMDP-based simulation for attack path planning

3.1 Introduction . . . 47
3.2 Existing cyber-range platforms . . . 49
3.3 Methodology . . . 51
3.3.1 POMDP design . . . 51
3.3.2 Automated POMDP environment construction . . . 57
3.3.3 Domain randomization . . . 61
3.4 Implementation . . . 62
3.5 Summary . . . 64

Chapter 4
Automated vulnerability taxonomy enrichment

4.1 Introduction . . . 65
4.2 Text-based multi-label classification . . . 67
4.2.1 Language models . . . 68
4.2.2 Language models for multi-label classification . . . 69
4.3 Existing approaches . . . 69
4.3.1 Limitations of manual mappings . . . 70
4.3.2 Automated mapping strategies . . . 71
4.4 Methodology . . . 72
4.4.1 Data collection . . . 73
4.4.2 Hierarchical classification . . . 74
4.5 Experiments . . . 75
4.5.1 Experimental setup . . . 76
4.5.2 Experimental results . . . 77
4.6 Environment integration . . . 83
4.7 Summary . . . 84

Part III Reinforcement learning for graph combinatorial optimization

Chapter 5
Scalable and generalizable RL agents for graph combinatorial optimization

5.1 Introduction . . . 89
5.2 Graph representation learning . . . 91
5.2.1 Unsupervised auto-encoders . . . 92
5.2.2 Graph neural networks . . . 92
5.3 Problem formulation . . . 94
5.4 Methodology . . . 94
5.4.1 Node representation learning . . . 94
5.4.2 Continuous spaces . . . 95
5.4.3 Projection agent . . . 96

5.4.4 RL-GCO automated framework . . . 97
5.5 Existing RL-GCO approaches . . . 98
5.6 Experiments . . . 99
5.6.1 Generalization study . . . 102
5.6.2 Inference time scalability . . . 103
5.7 Summary . . . 105

Chapter 6
Scalable and generalizable RL agents for critical attack path discovery

6.1 Introduction . . . 107
6.2 Action space dynamics . . . 109
6.3 Existing approaches . . . 110
6.3.1 Agent formulation . . . 110
6.3.2 Benchmark scenario complexity . . . 111
6.4 Methodology . . . 112
6.4.1 Graph representation . . . 112
6.4.2 Discrete local spaces . . . 114
6.4.3 Continuous latent spaces . . . 115
6.5 Experiments . . . 117
6.5.1 Experimental setup . . . 117
6.5.2 Scalability study . . . 119
6.5.3 Generalization study . . . 120
6.5.4 Dynamicity study . . . 122
6.6 Deployment-oriented study . . . 123
6.7 Summary . . . 125

Chapter 7
Multi-agent RL framework for secure virtual machine placement

7.1 Introduction . . . 128
7.2 Problem formulation . . . 129
7.3 Reinforcement learning extensions . . . 130
7.3.1 Solving multi-objective optimization with reinforcement learning . . . 130
7.3.2 Multi-agent reinforcement learning . . . 131
7.4 Existing solutions . . . 131
7.4.1 Modeling approaches . . . 131
7.4.2 Solutions for placement . . . 132
7.5 Methodology . . . 133

7.5.1 Graph representation . . . 133
7.5.2 Defender agent . . . 134
7.5.3 Latent spaces for the defender agent . . . 136
7.5.4 Simulation environment . . . 137
7.6 Experiments . . . 138
7.6.1 Experimental setup . . . 138
7.6.2 Threat model study . . . 140
7.6.3 Security–performance trade-off study . . . 141
7.6.4 Impact of cloud vulnerability landscapes . . . 141
7.6.5 Impact of initial allocation strategies . . . 142
7.6.6 Scalability study . . . 143
7.6.7 Replica pool study . . . 144
7.7 Deployment pipeline . . . 145
7.8 Summary . . . 146

Conclusions

Résumé en Français

Appendixs

Appendix A
Hyperparameter optimization and experimental configuration

A.1 Language models for CWE, CAPEC, and ATT&CK classification . . . 163
A.2 RL and auto-encoder models for GCO . . . 164
A.3 RL and auto-encoder models for attack path prioritization . . . 166

Appendix B
Graph combinatorial optimization benchmarks

B.1 Logic of GCO benchmarks . . . 169
B.2 Domain randomization . . . 171
B.3 Empirical heuristic exploration . . . 173

Appendix C
Attack path benchmark

C.1 Domain randomization . . . 175
C.2 Complexity metric . . . 175

Appendix D
Virtual machine placement benchmark

D.1 Domain randomization 177
D.2 Performance and security metrics 177

Bibliography

List of Figures **203**

List of Tables **205**

Acronyms **207**

INTRODUCTION

Over the last decades, networked infrastructures have grown significantly in both scale and complexity, becoming a fundamental component of modern organizations, industries, and critical systems. At the same time, the software layers and digital assets deployed across these infrastructures have also increased in complexity, leading to a rapidly expanding attack surface [1] and an unprecedented number of disclosed vulnerabilities [2, 3]. This continuous growth in the number of interconnected devices and deployed applications has made cybersecurity risk management increasingly challenging for large-scale infrastructures.

The growing number of disclosed vulnerabilities creates a major challenge for organizations: it is often impossible to mitigate every vulnerability immediately after disclosure. As a result, vulnerability prioritization has become a critical requirement [4, 5]. Existing vulnerability scoring systems and metrics, while widely adopted, generally rely on generic severity assessments and often fail to capture the context-specific characteristics of a particular network environment [5]. In practice, the actual criticality of a vulnerability strongly depends on factors such as network topology, host interconnections, attacker reachability, deployed services, and the strategic role of compromised assets within the infrastructure.

This thesis investigates the modeling of attacker behavior in networked infrastructures to support more effective vulnerability prioritization strategies. Rather than relying on classic static vulnerability metrics, the objective is to identify the vulnerabilities that are most critical within a given network context by considering how an attacker could realistically exploit them to achieve context-specific objectives [6]. Such an approach enables the development of context-aware security metrics that better capture the attack surface of individual infrastructures. Furthermore, identifying the most critical attack paths provides insight into the maximum potential impact that an attacker could achieve within a network.

Attack graphs have long been proposed as a means of modeling potential attack paths in networked infrastructures [7, 8]. These techniques represent how vulnerabilities and system states can be sequentially exploited by an attacker to progress toward a target objective. However, traditional attack graph approaches typically require enumerating all possible attack paths before prioritization algorithms can identify the most critical ones. As a result, they face significant scalability challenges when applied to large, complex infrastructures with numerous hosts, services, and vulnerabilities [9, 10, 11].

To address these limitations, recent research trends have focused on developing more scalable approaches that implicitly infer attacker behavior and identify critical attack paths without exhaustive enumeration. These methods typically aim to derive heuristics that approximate attackers' decision-making and highlight vulnerabilities most relevant for mitigation [12, 13, 14]. However, designing such heuristics manually remains difficult due to the complexity and diversity of modern infrastructures.

Consequently, recent advances increasingly leverage Machine Learning (ML), particularly Reinforcement Learning (RL) [15], to automatically learn such heuristics. RL is especially well-suited to this problem because it models sequential decision-making processes that closely resemble the step-by-step behavior of attackers selecting vulnerabilities and progressing through a network [16, 17]. By learning from interactions with simulated environments, RL-based approaches can discover efficient attack strategies and identify vulnerabilities that most significantly contribute to critical attack paths.

Nevertheless, even if effective heuristics can be learned automatically, important challenges

remain. Similar to broader challenges in the ML domain, generalization and scalability are central concerns [18]. Learned heuristics must generalize across different network scenarios and infrastructure configurations [19], capturing broader notions of critical attack path discovery that remain reusable across different scenarios. Furthermore, these approaches must scale to the size and complexity of modern networked infrastructures [20], which may contain large numbers of hosts and vulnerabilities. Addressing these challenges is essential to developing practical, robust ML-driven approaches to vulnerability prioritization in real-world networked environments.

Challenges and research questions

Achieving the objectives of this thesis involves several scientific and technical challenges that must be addressed to develop robust, scalable, and practical data-driven solutions for the problem of discovering critical attack paths.
RL approaches typically require thousands to millions of interactions with an environment to learn effective sequential decision-making policies [21]. Consequently, simulation environments play a fundamental role in enabling scalable and computationally efficient training processes [22]. However, such environments must accurately model attacker behavior, vulnerability exploitation dynamics, and the evolution of network states over time [23]. In addition, they must provide sufficient realism, diversity, and adaptability to support learning policies that generalize across heterogeneous network scenarios while accounting for partial observability [24] and the inherently sequential nature of vulnerability-exploitation dynamics.
A primary challenge, therefore, lies in the limited availability of simulation environments that simultaneously provide both realism and scalability. Existing solutions often struggle to generate large numbers of realistic and diverse network scenarios while accurately modeling the dynamic behaviors and interactions that characterize real-world infrastructures, all without requiring extensive manual engineering. This limitation hinders the training of RL agents, which must be exposed to a broad range of instances to learn robust, generalizable policies for discovering and prioritizing critical attack paths without repeated retraining.

Research Question 1 (RQ1)

How can existing vulnerability data sources be combined to generate realistic synthetic network scenarios and scalable simulation environments that accurately model attacker behavior and vulnerability exploitation dynamics?

The second challenge lies in the limitations of existing vulnerability data sources. Vulnerabilities disclosed in public databases often lack complete annotations, contextual information, and structured metadata necessary to properly characterize their impact, exploitability, and relationships with attacker objectives. Such limitations hinder the effectiveness of automated reasoning mechanisms and reduce the realism and expressiveness of simulation environments that rely on rich, structured representations of vulnerabilities to model vulnerability dynamics. Consequently, automating vulnerability understanding and metadata enrichment becomes a critical challenge.

Research Question 2 (RQ2)

How can the vulnerability annotation and metadata enrichment process be automated using ML techniques to improve the representation and understanding of vulnerabilities?

The third challenge concerns the design of scalable and generalizable heuristic learning strategies for attack path discovery across heterogeneous network configurations. Existing RL approaches often exhibit a strong dependence on the specific environments in which they are trained, limiting their ability to transfer knowledge and generalize to previously unseen configurations. These limitations may stem from both the parameterization of the learned policies and the inability of the underlying representations to capture structural properties that remain invariant across scenarios. This challenge is particularly significant in cybersecurity environments, where infrastructures are naturally represented as graphs whose topologies, vulnerabilities, and connectivity patterns can vary substantially while preserving common attack semantics.

Research Question 3 (RQ3)

How can RL policies, their parameterization, and their learning framework be designed to learn scalable and generalizable attack heuristics that remain effective across diverse network scenarios and vulnerability configurations?

Finally, an additional challenge lies in translating learned attacker behaviors into actionable security insights. While RL agents may successfully learn effective attack strategies and identify critical attack paths, it remains unclear how this knowledge can be systematically leveraged to support practical vulnerability prioritization, mitigation planning, and defensive decision-making. Bridging the gap between learned attack behaviors and actionable security recommendations is essential to ensure that such approaches provide tangible value in real-world applications.

Research Question 4 (RQ4)

How can attacker behaviors learned through RL policies be exploited to infer critical attack paths and support context-aware vulnerability mitigation strategies?

List of contributions

To address the research questions outlined in this thesis, a series of contributions are proposed, each specifically designed to tackle one or more of the identified research challenges and corresponding research questions.

- **C1 – An automated and more realistic simulation framework for critical attack path discovery (RQ1)**
  This contribution proposes a more realistic and scalable simulation framework for generating synthetic scenarios dedicated to attack path prioritization. Its key contribution is the integration and automation of empirically derived distributions from real-world environments into the scenario-generation process, reducing reliance on manually specified parameters while enabling greater control over the characteristics and diversity of generated infrastructures.
  The framework further incorporates publicly available vulnerability data to automatically

characterize vulnerabilities and their exploitation dynamics, enabling more data-driven modeling of attacker interactions. Building on `Microsoft CyberBattleSim`, the proposed `Continuous CyberBattleSim` extension addresses key limitations of existing cyber-range environments related to realism, scalability, scenario diversity, and vulnerability modeling, providing a flexible platform for training and evaluating RL-based attack path prioritization approaches.

Novelty. The novelty of this contribution lies in coupling an RL-oriented cyber simulator with empirical data not only to characterize the generated infrastructure graphs, but also to parameterize the simulator's state transition dynamics. This enables both the structure of the simulated environments and the evolution of their states to reflect empirical observations, addressing a limitation of the original CyberBattleSim and other existing cyber simulation environments.

Continuous CyberBattleSim
Franco Terranova, Abdelkader Lahmadi, Isabelle Chrisment.
GitHub Code: Continuous CyberBattleSim
Black Hat Arsenal Presentation (Asia 2026): blackhat.com/asia-26/arsenal

- **C2 – A multi-taxonomy vulnerability classification framework using language models (RQ2)**
  This contribution proposes a framework based on fine-tuned Language Models (LMs) for multi-taxonomy multi-label vulnerability classification. Given a vulnerability description, the framework automatically identifies associated weaknesses, attack patterns, and adversarial techniques across multiple relevant cybersecurity taxonomies. The primary objective is to automate vulnerability annotation and metadata enrichment, thereby facilitating the integration of newly disclosed vulnerabilities into the proposed simulation framework through taxonomy-based mappings. In addition, the framework serves as a decision-support tool for security analysts by accelerating annotation workflows and reducing the manual effort required to maintain large-scale vulnerability knowledge bases.

  Novelty. The novelty of this contribution lies in addressing multiple cybersecurity taxonomies jointly within a unified multi-label classification framework. The proposed methodology provides a common approach for leveraging the shared structure across taxonomies while ensuring high-performance predictions at scale for large multi-label problems involving extensive CVE datasets.

  Multi-Taxonomy Vulnerability Classification with Hierarchically Finetuned Language Models
  Franco Terranova, Sana Rekbi, Abdelkader Lahmadi, Isabelle Chrisment.
  DIMVA 2026 – Conference on Detection of Intrusions and Malware and Vulnerability Assessment, Jul 2026, Chania, Crete, Greece. [25]
  GitHub Code: CVE-LMTune
  Website: CVE-LMXplore
  Data Repository: Zenodo

- **C3 – Scalable and generalizable RL solutions for critical attack path discovery (RQ3)**
  This contribution investigates the design of scalable and transferable RL approaches for Graph Combinatorial Optimization (GCO) problems, with a particular focus on policy parameterization and efficient learning strategies. The objective is to develop policies capable of operating across diverse graph instances while improving scalability with respect to graph size and generalization to unseen scenarios. To address these challenges, this work proposes a novel RL reformulation based on latent decision spaces learned through Graph Neural Networks (GNNs), enabling agents to operate over compact representations rather than size-dependent problem formulations. The proposed methodology is first introduced as a general framework applicable to a broad class of GCO problems and is subsequently instantiated for the critical attack path discovery problem in networked infrastructures.

  Novelty. The main novelty of this contribution lies in a new policy parameterization for GCO, adapted from and extending approaches proposed in the existing literature. The study systematically analyzes the limitations of commonly used parameterizations and demonstrates how these limitations can affect generalization and scalability. The proposed parameterization is evaluated across multiple real-world GCO benchmarks, including the critical attack path discovery problem, which is characterized by a particularly large and high-dimensional action space. More broadly, the evaluation addresses networking and cybersecurity problems with large action spaces and strict inference-time constraints, requiring effective generalization and timely decisions, challenges largely overlooked in prior GCO studies.

Projecting Latent RL Actions: Towards Generalizable and Scalable Graph Combinatorial Optimization
Franco Terranova, Guillermo Bernardez, Albert Cabellos-Aparicio, Nina Miolane, Abdelkader Lahmadi.
Under Review. [26]
Github Code: LaGCO-RL
Data Repository: Zenodo

Scalable and Generalizable RL Agents for Attack Path Discovery via Continuous Invariant Spaces
Franco Terranova, Abdelkader Lahmadi, Isabelle Chrisment.
2025 28th International Symposium on Research in Attacks, Intrusions and Defenses (RAID), Oct 2025, Gold Coast, Australia, pp. 440–457. [27]
GitHub Code: Continuous CyberBattleSim
Data Repository: Zenodo

Leveraging Deep Reinforcement Learning for Cyber-Attack Paths Prediction: Formulation, Generalization, and Evaluation
Franco Terranova, Abdelkader Lahmadi, Isabelle Chrisment.
2024 27th International Symposium on Research in Attacks, Intrusions and Defenses (RAID), Sep 2024, Padua, Italy, pp. 1–16. [28]
Code Repository: Zenodo

Data Repository: Zenodo

- **C4 – A multi-agent RL framework for security-aware virtual machine placement** (**RQ4**) This contribution demonstrates how learned attacker policies can be leveraged to support practical mitigation and defense mechanisms through a representative use case: virtual machine placement (VMP). To this end, a multi-agent RL (MARL) framework is proposed, in which attacker and defender agents interact in a shared environment to address a multi-objective optimization problem that balances security and performance requirements. The attacker agent models adversarial behavior and identifies potential attack opportunities, while the defender agent dynamically adapts the infrastructure configuration to mitigate security risks while preserving operational efficiency. By integrating attack-path reasoning into the decision-making process, this contribution demonstrates how learned attack strategies can be transformed into actionable insights that enable automated, adaptive, and security-aware infrastructure management pipelines.

  Novelty. The main novelty of this contribution lies in extending the use of RL policies from attack-path discovery to security-aware mitigation and defensive decision-making. The proposed approach transfers the information learned by the attacker agent into a defender's decision-making process and instantiates this paradigm for VMP. In contrast to existing VMP formulations that primarily optimize security metrics with low realism, the proposed framework explicitly incorporates security metrics derived from adversarial behavior directly into the placement decision.

  When Performance Meets Security: Reducing Attack Paths in Virtual Machine Placement via Reinforcement Learning
  Franco Terranova, Noura Limam, Abdelkader Lahmadi, Isabelle Chrisment, and Raouf Boutaba.
  AISec 2026 - 19th ACM Workshop on Artificial Intelligence and Security, Nov 2026, The Hague, Netherlands. [29]
  Code Repository: Zenodo
  Data Repository: Zenodo

Taken together, the contributions form a unified framework that jointly characterizes the learning agent and its environment, providing an end-to-end approach for critical attack-path discovery.

Novelty. The main novelty of the proposed framework lies in its end-to-end capability to: (i) control the realism, diversity, and variability of generated scenarios in simulation, thereby enabling representative and controllable RL training environments for critical attack path discovery; (ii) incorporate real-world vulnerabilities, including vulnerabilities that have not yet been explicitly identified or annotated by security experts, into the simulation and systematically model their potential impact during RL training and evaluation; (iii) provide techniques for developing more generalizable and scalable RL agents, together with a systematic methodology for assessing their performance across diverse scenarios and problem settings; and (iv) establish a methodology for evaluating the practical deployment of these agents in conjunction with defensive mitigation mechanisms, thereby bridging the gap between attack-path discovery in simulation and its use within realistic cybersecurity defense workflows.

Thesis organization

The manuscript is organized into seven chapters, excluding the introduction and conclusion. Figure 1 provides a schematic overview of the organization, illustrating the relationships among the research questions, the proposed contributions, and the overall progression of the thesis.
Part I of the manuscript focuses on the background and related work, establishing the foundations of the thesis through a comprehensive overview of the state of the art related to the research topics addressed in this work. Chapter 1 presents existing vulnerability data sources and current approaches to vulnerability annotation and the automation of taxonomy linkage. Chapter 2 reviews attack graph and attack path planning methodologies together with existing automated solutions based on ML techniques. The subsequent parts of the thesis present the main research contributions.
Part II of the thesis focuses on the development of a learning framework for attack path planning. Chapter 3 introduces the proposed simulation framework, which generates realistic synthetic network scenarios and models network-level attack planning and vulnerability exploitation dynamics for critical attack path discovery. Subsequently, Chapter 4 presents a multi-label classification framework for automated vulnerability taxonomy annotation and metadata enrichment using LMs. More specifically, the proposed approach learns mappings between vulnerability descriptions and MITRE taxonomies, enabling the automatic integration of recently disclosed and non-annotated vulnerabilities into the simulation framework introduced in Chapter 3.
Part III of the thesis focuses on RL methods for GCO. Chapter 5 proposes scalable and generalizable RL methodologies for GCO problems independently of the specific application task. The chapter primarily compares RL architectures, parameterizations, and learning strategies, and proposes a new solution to improve scalability and transferability across heterogeneous graph instances. Building upon these foundations, Chapter 6 adapts and evaluates this approach for critical attack path discovery and attacker behavior modeling, using the simulation framework proposed in Chapter 3 as the target benchmark environment.
Finally, Chapter 7 extends this work toward practical defense applications by proposing a multi-agent RL framework for optimizing security-aware virtual machine placement. This chapter builds upon both the attacker modeling methodologies introduced in Chapter 6 and the scalable and generalizable RL strategies proposed in Chapter 5.
The manuscript concludes with a synthesis of the main findings, discussing their implications, limitations, and potential directions for future research.

Context

This thesis is conducted within the framework of the project *SuperViz* (ANR-22-PECY-0008 PECY-2022), funded by the French National Research Agency (ANR) under the France 2030 program and supported by the Université de Lorraine. The research activities are carried out at the Inria Center at Université de Lorraine and at the LORIA laboratory (Laboratoire lorrain de Recherche en Informatique et ses Applications), within the RESIST research team.
Several parts of this thesis were also developed through international research collaborations. The work presented in Chapter 5 was carried out in collaboration with the Universitat Politècnica de Catalunya, particularly with the Barcelona Neural Networking Center, under the supervision of Prof. Albert Cabellos. This collaboration was supported through a research internship funded by the DREAM program of the Université de Lorraine within the French PIA project "Lorraine Université d'Excellence" (ANR-15-IDEX-04-LUE).

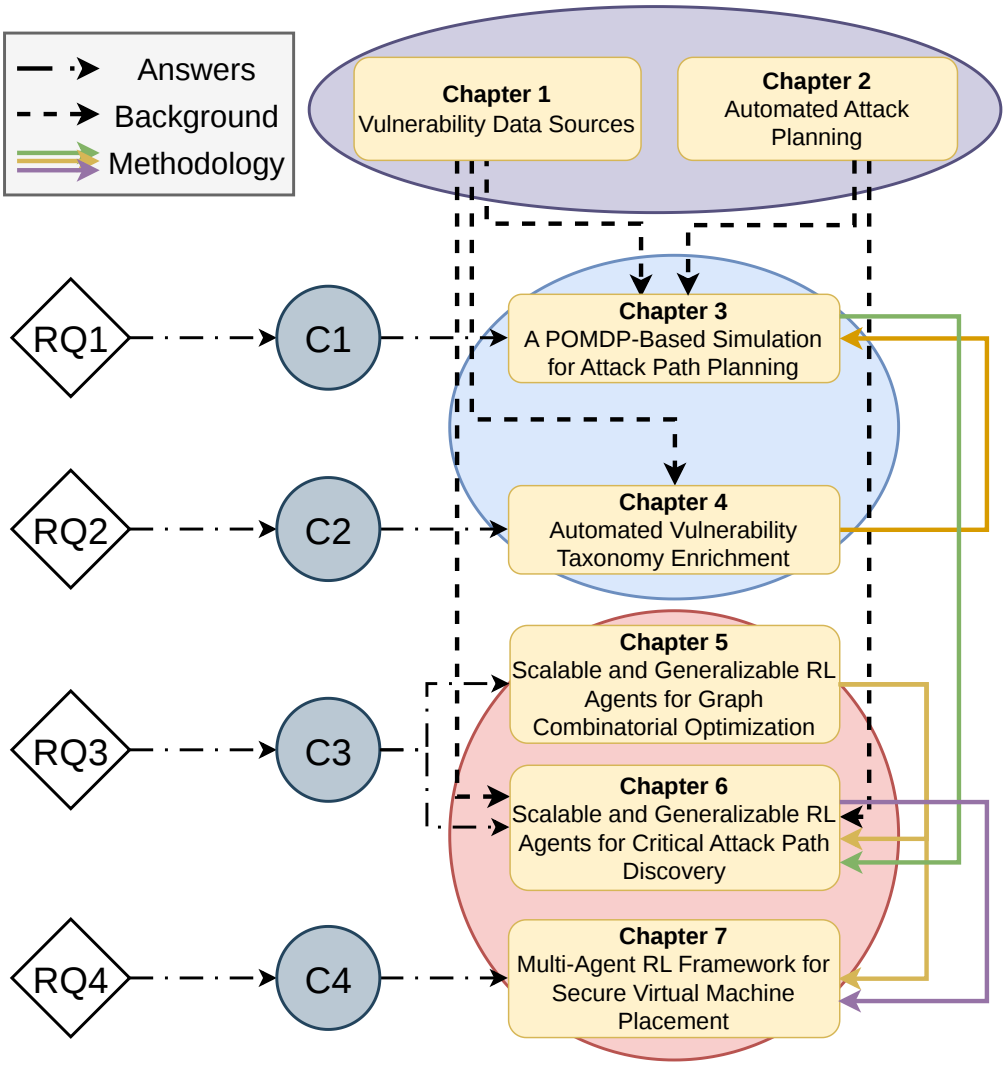


Figure 1: Thesis organization into chapters with links to research questions and contributions.

This international collaboration was further extended through joint research activities with the University of California, Santa Barbara, within the Geometric Intelligence Lab, in collaboration with researcher Guillermo Bernárdez and Prof. Nina Miolane.
In addition, the work presented in Chapter 7 on Multi-Agent RL for Virtual Machine Placement was conducted during research visits at the University of Waterloo, in collaboration with Prof. Noura Limam and the research group of Prof. Raouf Boutaba.

Part I

Background and related work

Chapter 1

VULNERABILITY DATA SOURCES

Summary: *This chapter introduces the main categories of cybersecurity data sources relevant to this work, ranging from vulnerability databases and structured vulnerability taxonomies to cyber threat intelligence sources capturing real-world activity. It also discusses existing efforts to establish semantic relationships among these heterogeneous sources, a challenge that is one of the central contributions of this thesis.*

Contents

1.1 Introduction . . . 13
1.2 Vulnerability databases . . . 14
1.2.1 Common vulnerabilities and exposures . . . 14
1.2.2 Exploit database. . . . 16
1.2.3 Vulnerability disclosure process . . . 16
1.3 Common vulnerability scoring system . . . 18
1.4 MITRE frameworks . . . 20
1.4.1 Common weakness enumeration . . . 20
1.4.2 Common attack pattern enumeration and classification . . . 21
1.4.3 Adversarial tactics, techniques, and common knowledge . . . 22
1.5 Cyber-threat intelligence . . . 23
1.6 Filling the gaps through automation . . . 24
1.7 Summary . . . 25

1.1 Introduction

Vulnerability research and practice rely heavily on structured, high-quality data sources to understand, model, and mitigate threats [30]. These data sources provide standardized metadata describing known vulnerabilities, software weaknesses, attack patterns, and adversarial behaviors, thereby enabling consistent analysis across tools, organizations, and research efforts. In data-driven cybersecurity solutions, the effectiveness and reliability of learning models are strongly influenced by the quality, quantity, and realism of the training data [31]. Consequently, the standardization and proper structuring of vulnerability information play a key role in supporting the development of automated systems for proactive risk assessment.
Over the past two decades, several initiatives led by organizations such as the MITRE Corporation [32] and the National Institute of Standards and Technology (NIST) [33] have established widely adopted frameworks and databases that now form the backbone of modern cybersecurity knowledge [34].
These resources serve multiple complementary purposes: (1) they provide a common vocabulary that facilitates communication among researchers, practitioners, and policymakers; (2)

they enable the systematic categorization of security issues at different granularities, ranging from low-level software weaknesses to higher-level attack strategies; (3) they establish structured relationships between different types of taxonomies, such as linking vulnerabilities to the techniques that adversaries may exploit [35].
The following sections provide an overview of the main cybersecurity data ecosystems used throughout this thesis, ranging from vulnerability repositories to structured knowledge bases and cyber threat intelligence (CTI) sources [36]. Particular attention is given to their structure, complementary roles, and the growing body of related research aimed at establishing semantic links between these heterogeneous sources.

1.2 Vulnerability databases

Vulnerability databases are centralized repositories that catalog publicly disclosed security flaws in software, hardware, and networked systems [37]. They play a critical role in the cybersecurity ecosystem by enabling the systematic identification, tracking, and remediation of vulnerabilities. These databases provide structured, standardized information for each entry, detailing affected products and versions, severity metrics, and references to external advisories. Importantly, vulnerability databases do not exist in isolation but are part of a broader ecosystem of interconnected knowledge bases. Entries are often enriched with standardized taxonomies, severity scoring systems, and references to related weaknesses, allowing for multi-level analysis of security risks.
Beyond their operational use, vulnerability databases constitute a foundational resource for cybersecurity research. They enable large-scale empirical analyses of vulnerability trends [38] and support the development of data-driven methods for risk assessment and threat modeling [39]. Their structured nature also enables seamless integration with automated systems such as vulnerability scanners, intrusion detection systems, and security information platforms [40].
Among the most prominent vulnerability databases is the National Vulnerability Database (NVD) [33], maintained by NIST. The NVD builds upon publicly disclosed vulnerabilities and provides additional layers of structured metadata that support automated analysis. In particular, the NVD enriches vulnerability entries with the following metadata (as shown in Figure 1.1):

1. **Severity scores**, primarily based on the Common Vulnerability Scoring System (CVSS) [41], including base, temporal, and environmental metrics
2. **Reference links** to vendor advisories, exploits, patches, and external analyses
3. **Weakness mapping**, linking vulnerabilities to their underlying enabling software weaknesses, through the Common Weakness Enumeration (CWE) identifier [42]
4. **Affected product**, represented through the Common Platform Enumeration (CPE) identifier [43], which standardizes the representation of affected products and versions

This enriched structure creates a structured dataset that can be readily consumed by automated tools and analytical pipelines. As a result, the NVD is widely used in both industrial and academic contexts for several cybersecurity applications.

1.2.1 Common vulnerabilities and exposures

Another key component of the vulnerability ecosystem is the Common Vulnerabilities and Exposures (CVE) system [44], which provides a standardized method for identifying and referencing publicly known vulnerabilities.

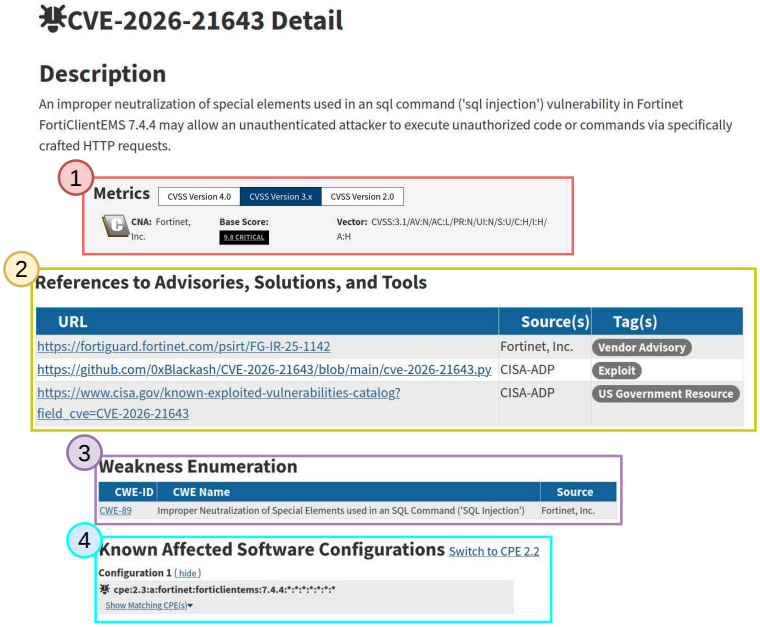


Figure 1.1: NVD entry highlighting its main components, including CVSS metrics, vulnerability references such as exploit links and vendor advisories, CWEs, and known affected products.

Rather than serving as a fully detailed database, CVE acts as a unifying indexing system that ensures consistency across complementary information from sources such as the NVD, vendor advisories, exploit databases, and higher-level knowledge bases, thereby supporting a unified and interoperable view of vulnerabilities [45].

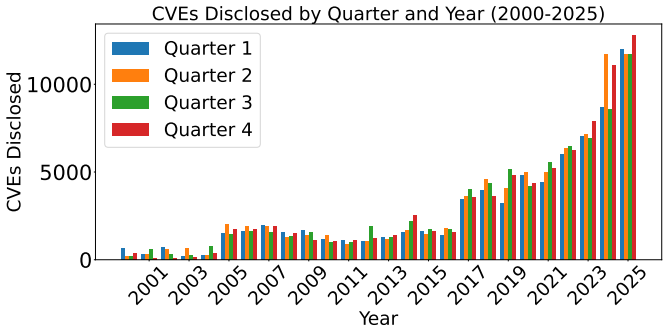


Figure 1.2: CVE counts per quarter and year from 2000 to Q1 2025.

Figure 1.2 illustrates the evolution of the number of CVEs disclosed between 2000, shortly after the establishment of the CVE program, and 2025. The figure highlights a sustained and rapid increase in vulnerability disclosures, rising from 1,438 CVEs in 2000 to 48,244 in 2025, corresponding to an average annual growth rate of 22.1%. This trend reflects the continuous expansion of the adoption of this vulnerability identification ecosystem.
Each vulnerability is assigned a unique identifier of the form:

$$\texttt{CVE-YYYY-NNNN}$$

where `CVE` is the Common Vulnerabilities and Exposures identifier, while:

$$\texttt{YYYY} \rightarrow \text{year of assignment or publication}$$
$$\texttt{NNNN} \rightarrow \text{unique sequential vulnerability identifier}$$

This standardized format enables consistent and unambiguous referencing of vulnerabilities across security datasets, vendors, and cybersecurity tools.

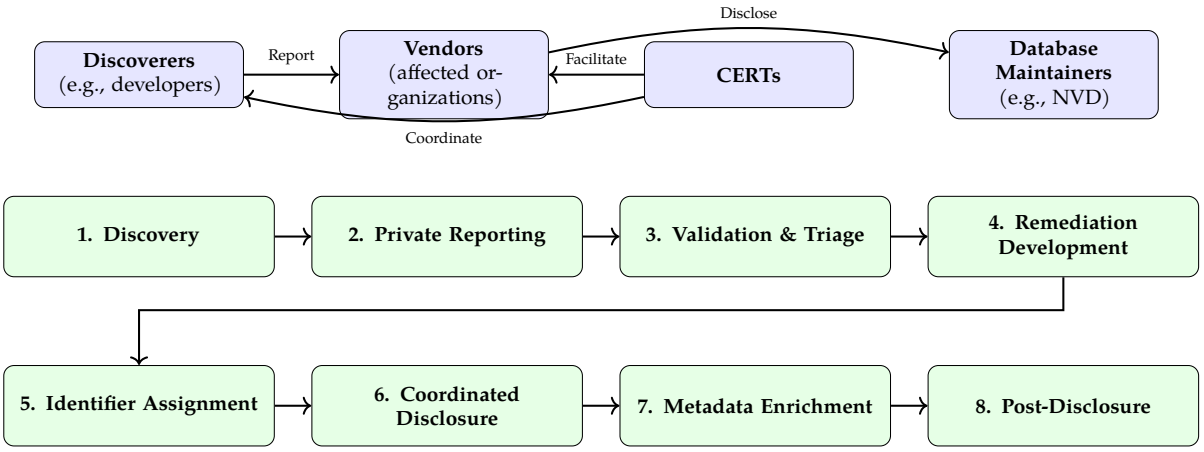


Figure 1.3: Overview of the stakeholders involved in coordinated vulnerability disclosure, their respective roles and interactions, and the temporal lifecycle of disclosure. [48]

CVE entries are curated by a distributed network of authorized organizations known as CVE Numbering Authorities (CNAs)[46]. These entities are responsible for assigning identifiers, validating vulnerability reports, and ensuring the consistency and quality of entries. This decentralized governance model enables the system to scale efficiently as the volume of disclosed vulnerabilities grows, while maintaining a high level of reliability and standardization.

1.2.2 Exploit database.

Another widely used resource in the vulnerability ecosystem is the Exploit Database (ExploitDB) [47], maintained by Offensive Security. ExploitDB is a public archive of Proof-of-Concept (PoC) exploits and vulnerability demonstrations targeting publicly disclosed vulnerabilities. Unlike other vulnerability databases, which primarily focus on vulnerability documentation and metadata enrichment, ExploitDB provides executable exploit artifacts and technical demonstrations for security research, penetration testing, and reproducibility.
A significant portion of ExploitDB entries are explicitly linked to CVE identifiers, enabling direct associations between vulnerabilities documented in the NVD and publicly available exploit implementations. This linkage plays an important role in practical risk assessment, as the existence of a public exploit is often considered a strong indicator of exploitability and real-world threat exposure.

1.2.3 Vulnerability disclosure process

The vulnerability disclosure process defines the lifecycle through which a security flaw is discovered, reported, analyzed, and ultimately disclosed to the public.

Stakeholders involved. The disclosure process typically involves several key actors:

- **Discoverers**: security researchers, developers, or analysts who identify the vulnerability.
- **Vendors**: organizations responsible for the affected software or system.
- **Coordinating entities**: organizations such as Computer Emergency Response Teams (CERTs) [49] that facilitate communication and coordination.

- **Database maintainers**: entities responsible for publishing vulnerability records, such as the NVD.

Temporal disclosure lifecycle. The disclosure process can be described as a sequence of distinct phases [50], each associated with specific actions and information flows (Figure 1.3):

1. **Discovery phase.** A vulnerability is identified through activities such as security testing, code auditing, fuzzing, or real-world exploitation. At this stage, knowledge of the vulnerability is typically limited to the discoverer.
2. **Private reporting phase.** The discoverer reports the vulnerability to the vendor or a coordinating entity. The report generally includes technical details, PoC exploits, and potential impact assessments. Confidentiality is maintained to prevent premature exposure of vulnerabilities and is critical for enabling affected vendors to develop and deploy remediation measures [51] and minimizing the window of opportunity for malicious exploitation [46, 52].
3. **Validation and triage phase.** The vendor evaluates the report to confirm the vulnerability's existence, assess its severity, and identify affected products or versions. This phase may involve iterative communication with the discoverer.
4. **Remediation development phase.** The vendor develops and tests a fix, patch, or mitigation strategy. The duration of this phase depends on the complexity of the vulnerability and the affected systems.
5. **Identifier assignment and initial enrichment.** Following validation, the vulnerability may be assigned a unique identifier through standardized naming systems such as the CVE program. At this stage, the initial disclosure record generally contains mandatory descriptive information, including a technical description of the vulnerability [53] and the affected products or software versions, commonly represented using CPE identifiers.
6. **Coordinated public disclosure phase.** After a predefined disclosure deadline — or, preferably, once a patch or mitigation becomes available — the vulnerability is publicly disclosed. The initial public disclosure primarily aims to communicate the existence and technical nature of the vulnerability and to enable affected parties to deploy mitigation and remediation measures.
7. **Post-disclosure metadata enrichment.** Following the initial disclosure, additional metadata may progressively be added by vulnerability databases and security organizations. This enrichment process may include severity assessment through CVSS scoring, weakness categorization using taxonomies such as CWE, exploitability analysis, and references to related advisories. Since these analyses often require expert assessment or coordination across multiple entities, their publication may be significantly delayed after the initial disclosure [54].
8. **Post-disclosure phase.** After disclosure, the vulnerability may be integrated into security tools, exploit databases, and threat intelligence feeds. Monitoring continues for exploitation in the wild and patch adoption rates.

Overall, the vulnerability disclosure process constitutes a fundamental mechanism for maintaining the security of modern systems. Its effectiveness relies on well-defined procedures, standardized identifiers, and coordinated interactions among stakeholders, ensuring that vulnerabilities are managed in a controlled and transparent manner.

1.3 Common vulnerability scoring system

The CVSS is a standardized framework for assessing the severity of vulnerabilities. It produces a numerical severity score ranging from 0 to 10, as well as additional discrete qualitative indicators across several security dimensions. These scores are designed to provide a consistent and comparable measure of vulnerability impact across different systems and contexts.
Several versions of CVSS have been proposed over time, with improvements in expressiveness and accuracy [55]. The most widely adopted version at the time of writing is CVSS v3.1, which refines earlier versions (v2.0 and v3.0) by improving metric definitions and scoring consistency.

Base metrics and vector representation. CVSS v3.1 primarily relies on a set of *Base Metrics* that capture the intrinsic technical characteristics of a vulnerability that remain constant over time and across deployment environments. These metrics constitute the core severity assessment commonly reported in vulnerability databases (e.g., NVD).
A key feature of CVSS is its standardized vector string representation, which encodes the values of the base metrics in a compact and machine-readable format:

```
CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H
```

Each component corresponds to a specific base metric:

- **AV (Attack Vector)**: {N (Network), A (Adjacent), L (Local), P (Physical)}
  Defines how far an attacker can exploit the vulnerability remotely.
- **AC (Attack Complexity)**: {L (Low), H (High)}
  Captures the number of conditions beyond the attacker's control that must be satisfied for effective exploitation.
- **PR (Privileges Required)**: {N (None), L (Low), H (High)}
  Describes the level of privileges required prior to exploitation.
- **UI (User Interaction)**: {N (None), R (Required)}
  Indicates whether exploitation depends on user actions (e.g., clicking a link).
- **S (Scope)**: {U (Unchanged), C (Changed)}
  Reflects whether exploitation impacts resources beyond the initially vulnerable component (scope change).
- **C (Confidentiality Impact)**, **I (Integrity Impact)**, **A (Availability Impact)**: {N (None), L (Low), H (High)}
  Represents the vulnerability's potential impact on the system. These metrics quantify the consequences of a successful exploit on data confidentiality, system integrity, and service availability.

Each qualitative value is mapped to a numerical weight defined in the CVSS specification, enabling a subsequent quantitative score computation.

Base score computation. The CVSS base score aims to capture the intrinsic severity of a vulnerability through heuristic formulas [56] that combine the aforementioned elements into two complementary aspects: how easily it can be exploited (exploitability sub-score) and the magnitude of its impact (impact sub-score).
The **exploitability score** reflects the effort required for a successful attack, and it is computed as in Equation 1.1.

$$\text{Exploitability} = 8.22 \times AV \times AC \times PR \times UI \tag{1.1}$$

Each factor represents a normalized weight for its corresponding metric. For instance, vulnerabilities exploitable over the network (AV:N) with no required privileges (PR:N) and no user interaction (UI:N) yield higher exploitability values. This sub-score captures the likelihood of exploitation under typical conditions.
On the other hand, an impact sub-score (Equation 1.2) quantifies the potential consequences of a successful exploit on the system, and it is used to determine the overall **impact score** (Equation 1.3).

$$ISC_{\text{Base}} = 1 - (1 - C)(1 - I)(1 - A) \tag{1.2}$$

$$\text{Impact} = \begin{cases} 6.42 \times ISC_{\text{Base}} & \text{if } S = U \\ 7.52 \times (ISC_{\text{Base}} - 0.029) - 3.25 \times (ISC_{\text{Base}} - 0.02)^{15} & \text{if } S = C \end{cases} \tag{1.3}$$

This formulation ensures that the combined impact increases as more security properties are affected. The distinction between unchanged and changed scope reflects whether the vulnerability can affect components beyond its original security boundary, introducing a non-linear amplification of the impact.
The final **base score** combines both sub-scores into a single component, as defined in Equation 1.4.

$$\text{Base Score} = \begin{cases} 0 & \text{if Impact} \leq 0 \\ \text{RoundUp}(\min(\text{Impact} + \text{Exploitability}, 10)) & \text{if } S = U \\ \text{RoundUp}(\min(1.08 \times (\text{Impact} + \text{Exploitability}), 10)) & \text{if } S = C \end{cases} \tag{1.4}$$

This aggregation balances exploitability and impact, ensuring that both dimensions contribute to the final severity assessment. The use of scaling factors and non-linear transformations encodes expert-driven heuristics for ranking vulnerabilities relative to one another.

Temporal and environmental metrics. Beyond the base metrics and their associated base score, CVSS defines two complementary metric groups that refine the severity assessment according to evolving conditions and deployment-specific contexts.

- **Temporal metrics** account for factors that evolve over time [57], including exploit code maturity, remediation availability, and report confidence.
- **Environmental metrics** adapt the severity assessment to a specific operational environment [58], accounting for asset criticality, deployment configurations, and organization-specific impact considerations.

These additional metric groups enable organizations to derive context-aware risk assessments rather than relying solely on the generic base severity score.

Practical considerations and limitations. In practice, CVSS scores are typically assigned by security experts, vendors, or database maintainers. This manual process can introduce delays and inconsistencies due to subjective interpretation of metrics [54]. Moreover, CVSS evaluates vulnerabilities as independent entities and does not account for interdependencies or sequential attack strategies. As such, while it provides a standardized and widely adopted severity measure, it may not fully capture system-level risk in complex environments.

1.4 MITRE frameworks

In addition to vulnerability databases, the ecosystem relies on structured knowledge bases that provide higher-level abstractions of security issues, primarily developed and maintained by the MITRE Corporation.
Unlike vulnerability databases, which focus on specific instances of security flaws, MITRE frameworks aim to capture generalizable concepts and relationships. As such, they play a critical role in bridging low-level vulnerability data with higher-level threat modeling and analysis.
A common characteristic of these knowledge bases is their hierarchical organization in the form of structured Directed Acyclic Graphs (DAGs) [59]. In such representations, nodes correspond to security concepts, while edges encode semantic relationships between them. The acyclic property ensures that relationships form hierarchical or dependency structures without cyclic loops, enabling consistent reasoning, inheritance, abstraction, and traversal operations.
These graph structures support multiple levels of abstraction, allowing high-level concepts to be progressively refined into more specialized or concrete instances, facilitating both human interpretability and automated analysis.

1.4.1 Common weakness enumeration

CWE is a standardized taxonomy of software and hardware weaknesses. It provides a structured classification of the underlying causes of vulnerabilities, rather than the vulnerabilities themselves. As such, CWE serves as a foundational layer for understanding why vulnerabilities arise. Each CWE entry represents a specific type of weakness, such as improper input validation, buffer overflows, or incorrect access control mechanisms.
These entries typically include the following information [60]:

- A unique identifier (e.g., `CWE-79`)
- A name and description
- Common consequences (e.g., impacts on confidentiality, integrity, or availability)
- Potential mitigations and detection methods
- Additional contextual information such as modes of introduction or likelihood of exploit

CWE organization is structured as a DAG, enabling a weakness to participate in multiple semantic relationships simultaneously rather than belonging to a strict tree hierarchy. Among these relationships, the primary structural relation is the *Parent-Child* (*Is-A*) relationship, which captures abstraction and specialization levels between weaknesses. Parent nodes represent more general weakness categories, whereas child nodes correspond to increasingly specific refinements or concrete manifestations of these weaknesses.
The CWE (Is-A) hierarchy is structured around several abstraction categories:

- **Pillar**: highest-level abstraction representing a broad class of weaknesses,
- **Class**: abstract groups of related weaknesses sharing common properties,
- **Base**: weaknesses that are sufficiently specific to support detection and mitigation guidance,
- **Variant**: specialized forms of weaknesses tied to particular technologies, languages, or implementations,
- **Compound**: weaknesses emerging from the combination or interaction of multiple weaknesses.

A *Pillar* node always appears at the top level of the taxonomy. Beneath it, entries from the set

{*Class, Base, Variant, Compound*} can be recursively nested to form flexible multi-level hierarchies with no fixed depth. Because the structure is graph-based rather than purely hierarchical, a single CWE entry may inherit from multiple parent weaknesses and therefore appear at different abstraction levels depending on the semantic relationship considered.
The latest CWE taxonomy at the time of writing (v4.17) contains 943 unique entries connected through 1,310 parent-child relationships. The distribution of entries across hierarchy levels is as follows:

- Level 0: 10 entries,
- Level 1: 178 entries,
- Level 2: 445 entries,
- Level 3: 428 entries,
- Level 4: 119 entries,
- Level 5: 24 entries.

This hierarchical and multi-relational organization enables CWE to represent both broad conceptual weakness families and highly specialized implementation-specific weaknesses within a unified semantic framework.

Example: CWE-284: Improper Access Control (Pillar) → CWE-285: Improper Authorization (Class) → CWE-552: Files or Directories Accessible to External Parties (Base) → CWE-219: Storage of File with Sensitive Data Under Web Root (Variant)

1.4.2 Common attack pattern enumeration and classification

The Common Attack Pattern Enumeration and Classification (CAPEC) framework [61] provides a structured catalog of attack patterns that describe how adversaries exploit weaknesses to achieve malicious objectives. While CWE focuses on weaknesses, CAPEC focuses on attacker behavior.
Each CAPEC entry describes a specific attack pattern, including:

- A unique identifier (e.g., `CAPEC-66`)
- A detailed description of the attack
- Preconditions required for execution
- Step-by-step execution flow
- Required attacker skills and resources
- Typical consequences and impacts

CAPEC attack patterns are designed to capture reusable adversarial strategies applicable across different technologies, systems, and operational contexts.
Similarly to CWE, CAPEC entries are organized as a DAG, with the primary structural relationship being the *generalization–specialization* relationship, which captures different levels of abstraction between attack patterns. General patterns describe broad attack strategies, while specialized child patterns refine them into increasingly precise exploitation procedures or implementation-specific attacks. This CAPEC hierarchy is structured around several abstraction categories:

- **Category**: high-level thematic grouping of related attack patterns,
- **Meta attack pattern**: highly abstract attack strategies independent of specific technologies,
- **Standard attack pattern**: commonly observed attack procedures describing typical adver-

sarial behavior,

- **Detailed attack pattern**: highly specialized attack techniques tied to precise implementation contexts or exploitation details.

The typical abstraction flow follows:

$$\textit{Category} \rightarrow \textit{Meta Attack Pattern} \rightarrow \textit{Standard Attack Pattern} \rightarrow \textit{Detailed Attack Pattern}$$

The latest CAPEC taxonomy at the time of writing (v3.9) contains 552 unique attack patterns connected through 533 parent-child relationships. The distribution across hierarchy levels is as follows:

- Level 0: 53 entries,
- Level 1: 189 entries,
- Level 2: 316 entries.

This hierarchical and multi-relational organization enables CAPEC to represent both broad adversarial strategies and highly specialized attack procedures within a unified framework.

Example: CAPEC-152: Inject Unexpected Items (Category) → CAPEC-137: Parameter Injection (Meta Attack Pattern) → CAPEC-6: Argument Injection (Standard Attack Pattern)

Importantly, CAPEC can be explicitly linked to CWE entries, indicating which weaknesses are typically exploited by a given attack pattern and creating a bridge between the two dimensions.

1.4.3 Adversarial tactics, techniques, and common knowledge

The MITRE Adversarial Tactics, Techniques, and Common Knowledge (ATT&CK) framework [35] provides a comprehensive knowledge base of adversarial tactics and techniques derived from real-world observations. It models how attackers operate across the different stages of an intrusion, offering behavioral and operational perspectives on cybersecurity threats.
ATT&CK is structured around three hierarchical concepts:

- **Tactics**: high-level adversarial objectives representing the "why" of an attack (e.g., initial access, persistence, lateral movement).
- **Techniques**: specific methods used to achieve these objectives, representing the "how" of an attack (e.g., phishing, command execution, credential dumping).
- **Sub-techniques**: more fine-grained variations of techniques, providing additional detail on how a technique can be implemented (e.g., different forms of phishing such as spearphishing attachments or links).

Each ATT&CK technique (and sub-technique) is documented with rich contextual information:

- A detailed description of the adversarial behavior
- Procedure examples observed in real-world campaigns
- Detection methods and relevant data sources
- Mitigation strategies and defensive recommendations
- Associated software tools and threat actor groups

The ATT&CK framework is commonly represented as a matrix, where tactics define the columns and techniques are organized within each column (with sub-techniques underneath). To be precise, ATT&CK is divided into several domain-specific matrices reflecting different operational environments:

- **Enterprise**: the largest and most widely used matrix, covering adversarial behaviors targeting traditional IT environments such as Windows, Linux, macOS, cloud infrastructures, and enterprise networks.
- **Industrial Control Systems (ICS)**: focuses on attacks targeting operational technologies and industrial environments, including critical infrastructure sectors such as energy or manufacturing.
- **Mobile**: models adversarial techniques targeting Android and iOS ecosystems, including malicious applications, device exploitation, network interception, and mobile-specific techniques.

Inside each matrix, ATT&CK defines the same hierarchical relationship *Tactic* → *Technique* → *Sub-technique*, which just changes per matrix in terms of existing components for each level.
As of ATT&CK v17 (April 2025), the framework presents the following statistics across the three operational domains:

- Enterprise ATT&CK: 14 tactics, 211 techniques, and 468 sub-techniques,
- Mobile ATT&CK: 12 tactics, 75 techniques, and 46 sub-techniques,
- ICS ATT&CK: 12 tactics, 83 techniques, and no defined sub-techniques.

Example — *Enterprise*: Credential Access (Tactic) → T1110 — Brute Force (Technique) → T1110.002 — Password Cracking (Sub-technique)

ATT&CK complements CWE and CAPEC by focusing on observed adversarial behavior rather than abstract weaknesses or attack patterns. While CWE describes *what is wrong* in a system and CAPEC describes *how it can be exploited*, ATT&CK captures *how attackers actually operate in practice*. As previously stated, CWE entries can be mapped to CAPEC attack patterns that describe the exploitation methods associated with specific weaknesses, while CAPEC patterns can, in turn, be linked to ATT&CK techniques that represent real-world adversarial procedures. Its grounding in operational attack data therefore makes ATT&CK particularly valuable for bridging theoretical vulnerability models and practical cybersecurity operations.

1.5 Cyber-threat intelligence

CTI refers to the collection, aggregation, and analysis of information about real-world threats, exposed systems, adversarial campaigns, and attacker behavior. In contrast to structured knowledge bases such as vulnerability databases or MITRE frameworks, which primarily provide relatively stable and curated representations of security concepts, CTI emphasizes the empirical observation of actively evolving threats and operational attacker activity. As such, CTI offers a dynamic and continuously updated perspective on the cybersecurity landscape.
A key category of CTI sources is based on *internet-wide scanning platforms*, such as Shodan [62] and Censys [63]. These platforms continuously probe the public Internet to identify reachable devices and services, enabling large-scale observation of the global attack surface [64].
These platforms operate by systematically exploring IP address spaces and attempting connections over a predefined set of protocols and ports. For each responsive host, they collect raw data, including service banners, protocol responses, TLS certificates, and metadata exposed during the handshake phase.
This process is typically distributed and continuous, allowing platforms to maintain up-to-date snapshots of exposed systems. The collected data is then normalized and indexed, transforming raw network responses into structured records that can be queried efficiently.

Table 1.1: Overview of automated cybersecurity knowledge linking approaches grouped by source and target knowledge domains.

Works	Input	Output	Paradigm
CVE → CVSS			
[65, 66, 67, 68, 69, 70, 71]	CVE description	CVSS components	Single-label classification
[72]	CVE description	CVSS base score binned	Single-label classification
[73, 71]	CVE description	CVSS base score	Regression
CVSS x.y → CVSS w.z			
[74, 75]	CVE description	CVSS w.z	Single-label classification
CVE → Exploit Likelihood			
[76, 77, 78]	CVE entry	Soon to be exploited	Binary classification
CVE → MITRE			
[79, 80, 81, 82, 83, 84, 85, 86]	CVE description	CWE	Multi-label classification
[87, 88]	CVE description	CAPEC	Multi-label classification
[89, 3, 90, 91, 92, 93]	CVE description	ATT&CK	Multi-label classification
[94, 95]	CVE + historical mappings	Correct CWE	Multi-label classification
ExploitDB → CVE			
[96]	ExploitDB post	CVE entry	Generation
CVE → Exploit			
[97, 98, 99]	CVE entry	POC Exploit	Generation
CVE → Fix commits			
[100]	CVE entry	Ranked candidate fix commits	Ranking
Commit → Suspicious			
[101]	Commit	Vulnerability contributing	Classification
CVE → Cluster			
[102]	CVE	Insightful clusters	Clustering + Dimensionality reduction

Shodan, in particular, builds upon this scanning pipeline by organizing collected data into searchable entities, where each record corresponds to a service instance observed on a given host, enabling easy custom aggregations for statistical analysis.
Users can query the indexed data through structured search filters such as `product:Apache`, `port:22`, `country:FR`, or vulnerability-specific CVE queries, enabling fine-grained exploration of the global Internet attack surface. These querying capabilities support empirical analysis of exposed services, technologies, and vulnerable systems at Internet scale. Consequently, Shodan is widely used to characterize exposure trends and estimate the distribution of vulnerabilities across heterogeneous classes of Internet-connected devices and infrastructures.
Despite challenges such as data heterogeneity, measurement bias, noise, and temporal variability, CTI remains essential for capturing how vulnerabilities and attack techniques manifest in practice, thereby complementing more static aspects of vulnerability.

1.6 Filling the gaps through automation

As discussed in the previous sections, a major challenge associated with cybersecurity knowledge bases lies not only in maintaining and updating individual taxonomies, but also in establishing and preserving semantic relationships across heterogeneous data sources. Furthermore, many vulnerability records remain incomplete at the time of disclosure, necessitating methods that can infer or generate missing information. Examples include identifying the most likely CWEs associated with a vulnerability, linking vulnerabilities to relevant attack techniques, or predicting

exploit- and patch-related metadata.
These tasks are inherently complex and often require extensive expert analysis, manual curation, and domain-specific reasoning. As a result, the enrichment of vulnerability records is frequently delayed, as illustrated by processes such as CVSS vector assignment [67]. Such delays can significantly impact downstream applications that depend on timely and structured security intelligence.
To address these limitations, a growing body of research has explored the use of ML, and, more recently, LMs to automate the linking and enrichment of vulnerability knowledge bases. Table 1.1 summarizes existing works in this area, categorizing them according to the linking or generation task they address, the inputs they leverage, the outputs they produce, and the ML paradigms employed to model the input–output mapping. A common characteristic across these works is the reliance on textual information contained in vulnerability descriptions as input, which serves as the primary source of contextual information, as this metadata is mandatory at the time of vulnerability disclosure.
The main body of the literature focuses on tasks such as CVSS vector prediction and CWE/ATT&CK technique prediction, as these can be formulated using comparatively simpler paradigms, such as classification, which has contributed to their broader adoption and attention. In contrast, more complex and innovative tasks often require more advanced paradigms and modeling strategies, resulting in a comparatively smaller body of literature.
Despite the progress achieved with ML and LM-based methods, several challenges remain open. Many existing approaches address these tasks with limited granularity, for example by focusing on a restricted subset of labels rather than covering the complete CWE or ATT&CK taxonomies. Similarly, several studies rely on relatively small datasets, which may not fully represent the diversity and complexity of real-world vulnerability data. These limitations can affect the generalizability of the proposed methods and highlight the need for larger-scale evaluations, broader taxonomy coverage, and approaches capable of handling the evolving nature of cybersecurity knowledge.

1.7 Summary

This chapter presented the main categories of cybersecurity data sources that underpin modern security analysis. Vulnerability databases provide structured repositories of disclosed flaws, supported by standardized identifiers such as CVE and severity assessments via CVSS. The vulnerability disclosure process ensures the controlled publication and dissemination of this information. Complementing these sources, MITRE frameworks such as CWE, CAPEC, and ATT&CK offer higher-level abstractions that capture weaknesses, attack patterns, and adversarial behaviors. Finally, CTI sources, including internet-wide scanning platforms and threat aggregation systems, provide empirical insights into real-world exposure and attack activity.
Together, these data sources form a complementary ecosystem, bridging low-level technical vulnerabilities with high-level threat understanding. Their combination enables both systematic analysis and realistic modeling of cybersecurity risks, forming a foundational basis for data-driven approaches in cybersecurity. Despite numerous standards and structured knowledge bases, significant gaps, delays, and missing links persist across these heterogeneous data sources. As a result, a substantial body of research has focused on automating the enrichment and linking of cybersecurity knowledge to improve consistency, scalability, and interoperability among security frameworks.

Link to the thesis objectives

As this thesis focuses on critical attack path discovery through RL, whose training process typically relies on large-scale interactions with simulated environments rather than real-world infrastructures, it requires accurate environment models that represent vulnerabilities with their associated metadata and relationships to attacker-centric taxonomies. In this context, the automated mapping of vulnerabilities to cybersecurity taxonomies, together with the inference of missing vulnerability-related information, plays a crucial role in improving the completeness and usability of vulnerability knowledge bases. Such capabilities help reduce the delays, costs, and limitations associated with manual expert analysis, thereby facilitating the timely integration of newly disclosed vulnerabilities into simulation environments. Addressing this challenge constitutes one of the key contributions of this thesis and is explored in detail in Section 4.

Chapter 2

Automated attack path analysis

Summary: *This chapter introduces the problem of automated attack path planning for proactive network security assessment. It presents the fundamental concepts underlying attack path discovery and examines the key challenges faced by existing approaches, with particular emphasis on scalability limitations in large, complex infrastructures. The chapter then discusses the growing shift from exhaustive attack graph analysis toward heuristic- and learning-based methods to identify critical attack paths more efficiently. Finally, it reviews the current state of the art and motivates the methodologies and contributions proposed throughout this thesis.*

Contents

2.1 Introduction **27**
2.2 Attack graphs **29**
2.2.1 Attack graph representations 30
2.2.2 Logical attack graphs 31
2.2.3 Required knowledge and inputs 31
2.2.4 Remaining challenges 32
2.3 From exhaustive to critical attack paths **32**
2.4 Critical attack path discovery as a combinatorial optimization **34**
2.5 From exact optimization to heuristic search **36**
2.6 Learning paradigms **37**
2.6.1 Supervised learning 37
2.6.2 Unsupervised learning 37
2.6.3 Reinforcement learning 38
2.7 Heuristic learning approaches **39**
2.7.1 One-shot optimization 39
2.7.2 Sequential optimization 40
2.8 Existing heuristics solutions **41**
2.9 Summary **43**

2.1 Introduction

As computer networks become increasingly critical to modern infrastructure, organizations face the growing challenge of evaluating their systems' resilience to real-world cyberattacks. To address this issue, cybersecurity practices have evolved to incorporate adversarial security assessments, such as penetration testing (PT), including advanced forms such as red-teaming exercises, alongside defensive approaches such as blue teaming [103]. Among these methodologies, PT has become one of the most widely adopted approaches for assessing the security

posture of networked systems by emulating realistic attack behaviors and identifying attack strategies before they can be taken by malicious actors [104, 105].
PT was defined by Denis et al. as *"a simulation of an attack to verify the security of a system or environment to be analyzed ... through physical means utilizing hardware, or through social engineering"* [106]. More specifically, at the network level, PT involves evaluating the security of interconnected systems by designing and executing potential attack paths across hosts, exploiting known vulnerabilities in operating systems (OSs) and applications [107].
Broadly, PT can be divided into three main phases [108, 106]: (*i*) scoping and reconnaissance, which involve defining the scope of the assessment and collecting information about the target environment through network scanning and enumeration; (*ii*) vulnerability analysis, during which known or potentially unknown vulnerabilities are identified and assessed for potential exploitation; and (*iii*) exploitation and reporting, which consist of executing attacks using payloads or custom scripts, gathering evidence, and documenting the findings together with their security implications.
While these phases are often presented sequentially, PT in networked infrastructures is inherently iterative. Successful exploitation may reveal new hosts, services, credentials, or network paths that were previously inaccessible, triggering additional rounds of reconnaissance and vulnerability analysis. Consequently, attackers repeatedly cycle through these phases as they progress through the infrastructure, gradually constructing multi-step attack paths toward their objectives.
Designing and executing attack paths is inherently complex, time-consuming, and heavily dependent on specialized expertise acquired through years of practical experience [109, 110]. As a result, comprehensive PT is therefore frequently reserved for major organizational milestones [111].
The automation of PT has consequently emerged at different levels of abstraction. At the operational level, existing approaches primarily automate individual tasks within the PT workflow. Reconnaissance, for example, can be automated through network discovery and scanning tools such as `nmap` [112]. Vulnerability assessment can be supported by scanners such as `Nessus` [113] and `OpenVAS` [114], while exploitation can be facilitated by frameworks such as `Burp Suite` [115] and `Metasploit` [116]. These approaches automate the execution of specific PT operations, but generally leave the higher-level reasoning about which actions to perform, and in what order, to the human operator.
Despite these advances, a fundamental challenge remains: automatically determining how an attacker should combine individual actions to construct an effective attack path toward a given objective [117]. Automating the execution of a scan or an exploit does not, by itself, address the problem of deciding which vulnerability should be pursued next, how compromising one system may enable access to another, or which sequence of actions is most promising for reaching a particular objective. This higher-level decision-making problem is commonly referred to as *attack planning*, and seeks to automatically generate and optimize attack strategies to improve the speed, scalability, and effectiveness of security assessments while enabling greater levels of autonomous PT [118].
Attack planning can operate at different levels of abstraction. While some approaches ground attack steps in concrete exploits, commands, or tools, higher-level approaches reason directly over vulnerabilities, their dependencies, and resulting system states, without tying each action to a specific implementation. This makes the planning process less dependent on the availability or implementation of particular exploits and tools. This thesis focuses on this higher-level formulation, investigating the automatic identification and prioritization of promising attack paths based on vulnerabilities and system states. The objective is not to automate exploit

execution, but to automate the reasoning required to determine *which vulnerabilities should be pursued and in what order* to efficiently reach an attacker objective.
This section presents the background of existing automated attack-planning approaches, starting with classical attack graph-based methods and progressing to more recent solutions that focus on identifying only critical attack paths rather than exhaustively enumerating all possibilities. It further discusses how the problem has increasingly been formulated as a CO task, where heuristic methods are employed to overcome the scalability limitations of exact approaches. The chapter concludes by presenting existing ML frameworks suited to heuristic learning strategies, including the RL paradigm used in this thesis, followed by a review of related work on the full set of ML-based approaches for automating attack planning.

2.2 Attack graphs

Real-world intrusions rarely rely on a single exploit or attack step. Instead, adversaries typically advance through a network by chaining multiple actions, with each successful compromise creating further opportunities for exploitation. These sequences are commonly referred to as *attack paths*, while the set of all feasible attack paths within a target environment constitutes an *attack graph*, a formalism introduced to model adversarial progression through networked systems [7, 8]. By definition, an *attack graph* is a tuple $G = (S, \tau, S_0, S_g)$, where:

- S denotes the set of security states representing the possible configurations or privileges reachable by an attacker within the target environment;
- $\tau \subseteq S \times S$ denotes the transition relation describing how an attacker can move from one state to another through the execution of an action or exploitation step;
- $S_0 \subseteq S$ denotes the set of initial states corresponding to the attacker's starting conditions or initial access capabilities;
- $S_g \subseteq S$ denotes the set of goal or success states representing critical attacker objectives, such as privilege escalation or compromise of sensitive assets.

As a consequence, attack graphs aim to model the security states reachable by an attacker, the transitions between these states, and the overall alternative paths leading to critical objectives or goals. By explicitly representing these relationships, they provide a structured foundation for automated security assessment, risk evaluation, and adversarial planning.
In the context of this thesis, considering a granularity based on network nodes and vulnerability exploitation actions, each transition $(s_i, s_{i+1}) \in \tau$ represents an attacker movement within the network topology enabled by the exploitation of a specific vulnerability. For example, an attack path across a simplified network can be represented as:

$$s_0(\text{Web Server}) \xrightarrow{\text{Credential Theft}} s_1(\text{Database Server}) \xrightarrow{\text{Privilege Escalation } (v_2)} s_2(\text{Domain Controller}),$$

where each state corresponds to the attacker's capabilities on a specific network node. The initial state s_0 represents the attacker's position reached after compromising a vulnerable public-facing service. The attacker can then move laterally to internal nodes, such as a database server, by exploiting obtained credentials or additional weaknesses. Finally, $s_3 \in S_g$ represents the goal state where the attacker compromises a critical asset and achieves elevated privileges. Therefore, attack paths can be used to capture both the sequence of vulnerability exploitations and the progression of the attacker through the underlying network topology.

2.2.1 Attack graph representations

Over time, several attack graph representations have been proposed, each differing in how states and dependencies are modeled. The most common characterization distinguishes them among state-enumeration graphs [8], exploit-dependency graphs [119], and host-based attack graphs [11]. Table 2.1 summarizes these three major categories of attack graph representations based on their space complexity, characteristics, and limitations. These approaches differ primarily in their level of abstraction, scalability, and representation of attacker progression. In the complexity expressions, N denotes the number of hosts or network entities, while M denotes the number of vulnerabilities or exploits considered in the environment.

Table 2.1: Comparison of the main attack graph representations based on space complexity, characteristics, and limitations.

Type	Space Complexity	Characteristics	Limitations
State-Enumeration	$O(2^{MN})$	Finite-automata-based representation explicitly modeling attack states and transitions; strongly related to Markov-chain formulations.	Poor scalability due to combinatorial state explosion.
Exploit-Dependency	$O(N^2 + MN)$	AND/OR-graph representation modeling dependencies between exploits and privileges.	Reduced expressiveness for complex global state interactions.
Host-Based	$O(N)$	Network-centric representation focusing on attack propagation between hosts.	Lower exploit-level granularity and less detailed attack semantics.

State-enumeration graphs explicitly model all reachable security states and transitions between them, providing a highly expressive representation of attacker behavior. However, their exponential complexity makes them difficult to scale to realistic enterprise networks. In contrast, exploit-dependency graphs abstract attack progression by modeling causal relationships among exploits, privileges, and preconditions using AND/OR dependency structures, thereby significantly improving scalability while preserving detailed exploit semantics. Host-based attack graphs further simplify the representation by focusing on attack propagation between hosts and network assets rather than modeling individual exploits in detail. This abstraction significantly improves scalability and enables the analysis of larger infrastructures, albeit at the cost of reduced exploit-level granularity. Such fine-grained reasoning can subsequently be delegated to specialized models or analysis components operating on top of the host-level representation when more detailed exploitation analysis is required.

2.2.2 Logical attack graphs

Among the various attack graph representations, Logical Attack Graphs (LAGs) [120], which belong to the exploit-dependency family, have emerged as one of the most widely adopted approaches due to their ability to compactly capture causal relationships between vulnerabilities, attacker capabilities, privileges, and exploit dependencies while remaining scalable enough for medium-sized environments. Unlike state-enumeration solutions, which explicitly enumerate global network states, LAGs model the logical conditions under which exploits become feasible and how newly acquired privileges enable subsequent attack steps.
In LAGs, the environment is generally represented as a directed graph where:

- Leaf nodes represent factual information about the environment, such as network reachability, host configurations, or known vulnerabilities.
- Intermediate nodes represent attacker actions, exploit executions, or newly obtained privileges.
- Goal nodes represent critical attacker objectives, such as compromising sensitive hosts or obtaining administrative privileges.

One of the most influential frameworks based on logical attack graphs is Multihost, Multistage Vulnerability Analysis Language (MulVAL) [10]. MulVAL leverages Datalog-based logical reasoning to automatically generate attack graphs from network configurations, vulnerability databases, and security policies. By encoding attack preconditions and consequences as logical inference rules, MulVAL efficiently derives attack paths while avoiding a combinatorial explosion. In addition to MulVAL, several other attack graph generation frameworks are widely adopted, including TVA [11] and NetSPA [9]. All these approaches share the objective of improving the scalability of attack graph generation by avoiding the exponential complexity of state-enumeration methods. More specifically, they rely on higher-level abstractions and dependency-based reasoning mechanisms to reduce attack graph construction and analysis to polynomial time in many practical scenarios [23], making them sufficiently scalable for medium-sized networks.

2.2.3 Required knowledge and inputs

Constructing attack graphs requires extensive prior knowledge of the target environment, its assets, and the conditions that enable attacker progression. Although the exact requirements depend on the underlying framework, most approaches generally rely on a combination of infrastructure, vulnerability, connectivity, and security-related information to model feasible attack paths.
In general, most attack graph generation frameworks require the following input knowledge:

- **Network infrastructure**: information about hosts, devices, services, OSs, software versions, and network topology.
- **Connectivity and reachability**: communication links between assets, routing information, firewall rules, access-control policies, and network segmentation constraints.
- **Vulnerability intelligence**: known vulnerabilities, exploit availability, vulnerability severity scores, exploit preconditions, and affected software components.
- **Attacker capabilities and privileges**: initial attacker locations, accessible credentials, privilege levels, and assumptions regarding attacker knowledge or resources.
- **Security policies and defense mechanisms**: authentication requirements, privilege escalation constraints, and mitigation policies.

- **Attack logic and reasoning rules**: formal descriptions of exploit dependencies, attack propagation conditions, privilege transitions, and causal relationships between steps.

Together, these inputs form a structured representation of the environment from which feasible attack paths and adversarial progression strategies can be derived automatically.

2.2.4 Remaining challenges

Despite recent improvements in scalability through novel attack graph frameworks, which reduced the complexity of attack graph generation from exponential to polynomial time in many practical settings, several important limitations remain:

- **Residual scalability limitations**: although improved, polynomial-time graph generation can still become computationally expensive for large-scale infrastructures.
- **Dependence on handcrafted rules**: reasoning engines rely on manually designed interaction and propagation rules that must be continuously updated as systems, vulnerabilities, and attack techniques evolve [121].
- **Strong assumptions regarding environment knowledge**: these approaches typically assume complete, accurate, and up-to-date knowledge of the target infrastructure. Such assumptions rarely hold in real-world settings, where attackers operate under partial, uncertain, and continuously evolving observations of the environment.
- **Static modeling assumptions**: most traditional attack graph approaches are designed for relatively static infrastructures and have limited ability to capture dynamic changes in the network topology or system configurations [122]. Consequently, significant environmental modifications often require costly regeneration and reanalysis of the attack graph.

These limitations motivated a shift toward more scalable and adaptive attack-planning approaches that may alleviate the aforementioned issues.

2.3 From exhaustive to critical attack paths

Generating the complete set of attack paths provides a principled, complete way to assess the security criticality of a network. However, recent research has increasingly shifted from exhaustive attack-graph generation to identifying only the most critical attack paths, using them as indicators of the network's overall security posture [13, 12, 123]. This evolution reflects the practical observation that defenders are rarely interested in all theoretically possible attacks; instead, they primarily seek to identify the most realistic, impactful, or likely attack strategies. Rather than reasoning over the full attack graph, these approaches focus on worst-case attack scenarios to characterize the system's security level.

Early efforts in this direction relied on *symbolic planning* techniques, which represent cybersecurity scenarios as formal models composed of states, actions, and transition rules. A prominent example is the *Planning Domain Definition Language* (PDDL), a standardized language introduced for describing planning problems in automated reasoning. In this context, PDDL allows experts to specify the available actions an attacker can perform, their preconditions, and their resulting effects on the environment. By encoding vulnerabilities, system configurations, and exploitation steps as symbolic actions, automated planners can reason over the modeled environment and generate only the relevant portions of the attack graph corresponding to feasible attack paths [124]. While effective in constrained and well-defined scenarios, symbolic approaches inherit limitations from traditional attack-graph generation techniques: they require extensive manual

effort to define, update, and maintain domain-specific rules. Consequently, their scalability becomes increasingly challenging as network environments grow in size, complexity, heterogeneity, and dynamism, where vulnerabilities, configurations, and possible attack behaviors continuously evolve.

More recently, research has shifted toward *sub-symbolic* or data-driven attack-planning strategies, particularly those based on ML, which aim to identify critical attack paths directly from data. These approaches seek to approximate human decision-making heuristics by learning attack strategies from experience [16, 111]. This shift opens the door to adaptive planning methods that reduce reliance on manually crafted rules and expert-defined models while better coping with complex, evolving environments. Furthermore, learned heuristics can be readily adapted to incorporate newly discovered vulnerabilities, emerging attack techniques, or additional optimization objectives, without requiring the manual redesign of symbolic reasoning rules or attack models.

Nevertheless, regardless of whether the approach is symbolic or sub-symbolic, a fundamental question remains: *which type of attack graph representation should these methods operate on?*

In fact, starting from an initial node or attacker state, the prioritization algorithm must learn navigation strategies by selecting outgoing edges or by leveraging relevant node attributes to identify high-value attack paths for a given objective [125, 23].

However, more expressive attack graph formulations, such as state-enumeration and exploit-dependency graphs, differ fundamentally from host-based representations in the extent to which attack-related knowledge is explicitly encoded and precomputed. These richer formulations capture attacker states, privilege transitions, exploit dependencies, and causal relationships between attack steps directly within the graph structure. As a result, a significant portion of the attack reasoning process is embedded in the model itself, facilitating fine-grained attack-path analysis and prioritization.

In contrast, host-based attack graphs rely on a more abstract representation focused primarily on hosts and network connectivity. Rather than explicitly encoding complete attack chains or attacker progression, they precompute only the minimal information needed to describe the attack surface and potential interactions among assets. Consequently, attack paths and attacker behavior must be inferred dynamically by the reasoning or learning mechanism operating on top of the graph. This design substantially improves scalability and enables the analysis of large-scale infrastructures, although the richer attack semantics must then be reconstructed through additional mechanisms.

While expressive attack graph models offer a richer representation of attacker behavior and attack progression, they rely on strong assumptions and face several important practical limitations that hinder their applicability in large-scale and dynamic environments [23, 28, 122]:

(**1**) **Limited robustness to uncertainty.** State-aware and exploit-dependency attack graphs generally rely on a globally consistent and largely complete view of the target environment. As a result, they often require substantial recomputation whenever the network topology changes. Moreover, the semantics encoded in these models depend on attack states, privilege relationships, and exploit dependencies, all of which are assumed to be known in advance. In realistic settings, however, attackers operate under partial and evolving knowledge, progressively discovering hosts, services, credentials, and vulnerabilities through reconnaissance activities [118]. Consequently, incomplete or outdated information may invalidate significant portions of the precomputed attack graph and the reasoning derived from it.

In contrast, host-based representations encode only high-level information about discovered as-

sets and their connectivity, making them naturally compatible with incremental exploration of the environment. Newly acquired knowledge can be incorporated through localized graph updates as reconnaissance progresses, without requiring the reconstruction of complex attacker-state or exploit-dependency structures. This property makes host-based attack graphs particularly well suited for modeling attack planning under partial observability.

(2) **Limited robustness to environmental evolution.** Attack graphs are typically generated from a fixed snapshot of the environment. Changes such as hosts joining or leaving the network, service updates, or evolving security policies may invalidate previously computed attack paths, often requiring costly graph regeneration. This limitation becomes particularly problematic in highly dynamic environments such as Internet of Things (IoT) or software-defined networks, where changes occur continuously. Host-based representations, by contrast, can evolve incrementally with the infrastructure through localized graph modifications, due to the same advantages discussed in the previous point.

(3) **Scalability limitations.** Although exploit-dependency attack-graph frameworks significantly improve scalability compared to early state-enumeration approaches, the complexity of graph generation and optimization still increases rapidly with the number of hosts, services, vulnerabilities, and exploit dependencies [120, 126, 110, 127, 128, 108, 125], as modeling exploit relationships still leads to large and computationally expensive structures. Host-based formulations avoid explicit state enumeration and dependency expansion by directly representing the operational network, yielding substantially more scalable representations for large-scale environments.

In summary, modeling the environment directly at the operational network level using host-based graphs enables the formulation of attack planning as a partially observable, evolving decision-making process. By relying on a higher level of abstraction and deferring detailed attack reasoning to the planning or learning mechanism, this representation avoids many of the scalability, rigidity, and observability limitations associated with more expressive attack graph formulations. At the same time, it preserves the essential structural information needed to identify and prioritize critical attack paths, making it particularly well-suited to modeling adaptive, realistic attacker behavior in large-scale, dynamic environments.

2.4 Critical attack path discovery as a combinatorial optimization

As discussed in Section 2.2.4, the increasing scale, heterogeneity, and dynamicity of modern network infrastructures make exhaustive attack path enumeration rapidly impractical. In realistic environments, the number of reachable attack states, exploit combinations, privilege transitions, and lateral movement opportunities grows combinatorially with the network size. Consequently, security assessment efforts generally focus not on enumerating all possible attacks, but rather on identifying the most critical, realistic, or impactful ones. This naturally motivates formulating critical attack path discovery as a CO problem, typically performed using the host-based attack graph formulation to avoid the limitations discussed in Section 2.3.
CO [129] refers to a class of optimization problems in which the objective is to identify the best solution within a finite, yet potentially extremely large, discrete search space. Formally, a generic CO problem can be expressed as in Equation 2.1.

$$
\begin{aligned}
x^* = \arg\max_{x \in \mathcal{X}} \quad & f(x) \\
\text{s.t.} \quad & g_i(x) \leq 0, \quad \forall i \in \{1, \ldots, m\}, \\
& h_j(x) = 0, \quad \forall j \in \{1, \ldots, n\},
\end{aligned}
\tag{2.1}
$$

where:

- $\mathcal{X}$ denotes the set of feasible solutions,
- $f(x)$ represents the objective function evaluating the quality of a candidate solution x,
- $g_i(x)$ and $h_j(x)$ define inequality and equality constraints, respectively,
- x^* denotes the optimal solution according to a predefined optimization criterion.

Attack path planning as CO. In the context of critical attack path discovery, the elements of Equation 2.1 can be interpreted as follows:

- $\mathcal{X}$ represents the solution space of candidate attack paths. Depending on the level of abstraction adopted, a solution may correspond to a sequence of compromised hosts, a sequence of exploited vulnerabilities, a sequence of attacker actions, or a more detailed combination of network positions and vulnerabilities exploited throughout the attack process.
- $f(x)$ measures the criticality of an attack path. This objective may be conditional on a threat model and incorporate factors such as the value and sensitivity of compromised assets, the privileges obtained along the path, the amount of data exposed, the operational impact on the infrastructure, or other security-related risk metrics.
- $g_i(x)$ and $h_j(x)$ represent constraints governing the feasibility of attack progression. These constraints may encode prerequisites for exploitation, privilege requirements, network reachability, authentication conditions, firewall restrictions, credential dependencies, or any other conditions that must hold for an attack path to be executable.
- x^* corresponds to an optimal attack path according to the selected security objective and threat model. Depending on the definition of attack criticality, multiple attack paths may achieve the same optimal objective value, resulting in several equally critical solutions.

Importantly, the exact definition of these elements depends on the attack graph representation, attacker model, and security objectives under consideration. As a result, critical attack path discovery encompasses a broad family of optimization problems.

Attack path planning as graph CO. When extending attack planning to the network level, the optimization problem becomes significantly more challenging because it must operate over graph-structured environments, as network infrastructures can naturally be represented as graphs.
More generally, this problem falls within the class of GCO, in which the optimization process is defined directly over a graph $G = (V, E)$, where V denotes the set of entities and E the relationships between them. The decision variables are therefore graph elements themselves, such as nodes, edges, paths, or subgraphs. In the context of attack planning, a candidate solution is typically represented as a traversal or sequence of graph components $P = (v_1, v_2, \ldots, v_T)$, where each element $v_t \in h(G)$ and $h(G)$ denotes the set of feasible graph elements (e.g., nodes) or graph element attributes (e.g., vulnerabilities in nodes). The objective is to identify an optimal traversal strategy that maximizes the attacker's objective while satisfying the constraints imposed

by the problem and the graph structure.
Compared to classical CO problems, GCO in networking applications introduces several additional challenges [19]. First, network topologies vary significantly in size and structure, causing both the decision variables and solution space to change across instances. Second, the search space grows combinatorially with the number of hosts, vulnerabilities, and possible attack paths. Finally, attack planning is often performed in dynamic, partially observable environments, where newly discovered information continually alters the set of feasible actions and attack opportunities. Consequently, optimization methods must be scalable, adaptable, and capable of generalizing across heterogeneous graph instances.

2.5 From exact optimization to heuristic search

Exact optimization methods provide the standard approach for solving CO problems. However, their applicability is limited by the combinatorial explosion of the search space, as these methods typically require exploring the entire solution space, a challenge similar to that encountered in attack graph generation (see Section 2.2.1). In attack path planning, the number of possible attack paths grows exponentially with the number of hosts and vulnerabilities within the network, making exhaustive optimization rapidly intractable in realistic large-scale networks [130].
Consequently, practical attack planning systems typically rely on two complementary strategies: (i) heuristic search methods rather than exact optimization techniques [19], and (ii) compact, non-exhaustive underlying attack graph representations (see Section 2.3). Instead of exhaustively evaluating all possible attack paths, heuristic approaches guide the search toward the most promising solutions according to predefined notions of attack criticality [131]. When combined with compact graph representations, this significantly reduces the computational complexity of the problem and enables more scalable attack-path analysis in large networked environments.
The most well-established approaches rely on heuristics manually engineered by cybersecurity experts through handcrafted rules and scoring mechanisms. These heuristics are often derived from external security frameworks such as CVSS scores or MITRE ATT&CK techniques. For example, vulnerabilities associated with high CVSS exploitability or severity scores may be prioritized to guide the search toward potentially critical attack paths.
Although these heuristic-driven approaches significantly improve scalability by reducing the effective search space, they still introduce several important limitations:

1. **Designing manually** effective heuristics for realistic environments remains highly challenging. Modern infrastructures are heterogeneous, dynamic, and characterized by complex dependencies among vulnerabilities, services, privileges, and topology. As network complexity continues to evolve, heuristics frequently require manual redesign and recalibration to remain effective.
2. The optimization process ultimately depends on layered human-designed abstractions and assumptions. Frameworks such as CVSS are themselves expert-defined approximations of security risk, meaning that attack planning often relies on **multiple levels of handcrafted reasoning**.
3. Handcrafted heuristics often **generalize poorly** across different network environments and threat models due to the highly relational and dynamic nature of attack planning.

These limitations naturally motivate the following question: rather than manually engineering heuristic functions, can we automatically learn effective heuristic strategies [132] that, in turn, identify the most critical attack paths?

2.6 Learning paradigms

ML aims to learn predictive or decision-making functions directly from data rather than relying on manually designed rules. Depending on the nature of the available information and the learning objective, ML methods are commonly categorized into three main paradigms: supervised learning, unsupervised learning, and reinforcement learning.
All three paradigms rely on parameterized models f_θ, where θ denotes a set of learnable parameters. Learning involves optimizing these parameters with respect to an objective function that captures the model's desired behavior. Although the optimization objectives differ across paradigms, the general goal is to identify parameters that maximize predictive performance, uncover meaningful structures in data, or improve decision-making capabilities.

2.6.1 Supervised learning

Supervised learning addresses problems for which labeled examples are available during training. Given a dataset $\mathcal{D} = (x_i, y_i)_{i=1}^N$, where x_i denotes an input sample and y_i its associated target output, the objective is to learn a function $f_\theta : \mathcal{X} \to \mathcal{Y}$ capable of generalizing to previously unseen inputs. Depending on the structure of the target space $\mathcal{Y}$, supervised learning tasks are commonly categorized into three main classes: *classification*, where the objective is to assign inputs to one of several discrete categories; *regression*, where the objective is to predict continuous numerical values; and *ranking*, where the objective is to learn an ordering or preference relation among a set of instances according to a given criterion.
Training is typically formulated as an empirical risk minimization problem (Equation 2.2).

$$\theta^* = \arg\min_\theta \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}\big(f_\theta(x_i), y_i\big), \tag{2.2}$$

$\mathcal{L}$ denotes a loss function measuring the discrepancy between model predictions and target values. Common examples include cross-entropy loss for classification and mean squared error for regression. The resulting model learns to approximate the mapping between inputs and outputs observed during training.

2.6.2 Unsupervised learning

Unsupervised learning operates without explicit target outputs. Given a dataset $\mathcal{D} = {x_i}_{i=1}^N$, the objective is to discover meaningful patterns, structures, or latent representations directly from the observed data.
Representative unsupervised learning tasks include clustering, dimensionality reduction, representation learning, and generative modeling. Rather than predicting predefined targets, the model seeks to capture statistical regularities present in the dataset.
Training is commonly formulated as in Equation 2.3, where the loss function depends only on the observed data and the learned representation.

$$\theta^* = \arg\min_\theta \sum_{i=1}^{N} \mathcal{L}(x_i; \theta), \tag{2.3}$$

The exact form of $\mathcal{L}$ varies according to the task. For example, autoencoders minimize reconstruction errors, while clustering methods optimize measures of cluster compactness and separation.

2.6.3 Reinforcement learning

RL differs fundamentally from supervised and unsupervised learning in that learning occurs through interactions with an environment rather than from a fixed dataset. Instead of receiving target outputs, an agent learns by selecting actions and observing the resulting rewards.
This interaction process is commonly modeled as a Markov Decision Process (MDP) [133], typically defined as

$$\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, T, R, \rho_0 \rangle,$$

where:

- $\mathcal{S}$ denotes the state space, representing the set of possible environment configurations;
- $\mathcal{A}$ denotes the action space, corresponding to the set of decisions available to the agent;
- $T(s' \mid s, a)$ denotes the transition function, defining the probability of reaching state $s' \in \mathcal{S}$ after executing action $a \in \mathcal{A}$ in state $s \in \mathcal{S}$;
- $R(s, a)$ denotes the reward function, providing a scalar feedback signal guiding the optimization objective;
- ρ_0 denotes the initial state distribution from which training episodes are initialized.

At each timestep t, the agent observes the current state s_t, selects an action a_t, transitions to a new state $s_{t+1} \sim T(\cdot \mid s_t, a_t)$, and receives a reward $r_t = R(s_t, a_t)$, which guides the learning process. The objective is to learn a policy $\pi_\theta(a \mid s)$, parameterized by θ, that maximizes the expected cumulative reward over time (Equation 2.4), where $\gamma \in [0, 1]$ is a discount factor controlling the importance of future rewards.

$$\theta^* = \arg\max_\theta \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t\right], \tag{2.4}$$

In practice, in Deep RL (DRL), policies are typically implemented using Deep Neural Networks (DNNs) that either output action logits, converted into probabilities through a `softmax` function and sampled accordingly, or estimate Q-values defined as

$$Q^\pi(s, a) = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \,\middle|\, s_t = s, a_t = a\right],$$

which represent the expected discounted return of executing action a in state s. Q-values enable direct comparison between actions, commonly selecting actions using strategies such as `argmax` over Q-values.
Unlike supervised learning, RL does not rely on explicit target outputs. Instead, the learning signal is provided indirectly through rewards, requiring the agent to balance the exploration of new behaviors with the exploitation of previously acquired knowledge.
In this formulation, the agent attempts to infer the underlying network state from the sequence of observations and interactions. This capability is particularly relevant for automated attack surface discovery, where the complete attack surface is initially unknown and must be progressively identified through intelligent exploration.

Partially observable Markov decision processes. In many real-world settings, the complete environment state cannot be directly observed. Instead, decisions must be made under uncertainty using only partial and progressively acquired information. To model such settings, RL problems are often formulated as Partially Observable MDPs (POMDPs) [24].

Formally, a POMDP extends the previous MDP formulation as

$$\mathcal{P} = \langle \mathcal{S}, \mathcal{A}, \mathcal{O}, T, Z, R, \rho_0 \rangle,$$

where $\mathcal{S}$, $\mathcal{A}$, T, R, and ρ_0 retain the same meaning as in the MDP formulation, while:

- $\mathcal{O}$ denotes the observation space accessible to the agent;
- $Z(o \mid s', a)$ denotes the observation function, defining the probability of receiving observation $o \in \mathcal{O}$ after transitioning to state $s' \in \mathcal{S}$ following action $a \in \mathcal{A}$.

At each timestep, the agent no longer directly observes the true environment state s_t, but instead receives an observation $o_t \sim Z(\cdot \mid s_t, a_{t-1})$ providing only partial information about the underlying system. Consequently, optimal decision-making often depends not only on the current observation, but also on the history of previous interactions.

2.7 Heuristic learning approaches

Recent advances in ML have led to growing interest in replacing manually engineered heuristics for CO with learned optimization strategies. In these approaches, heuristics are no longer explicitly designed by human experts, but instead learned automatically from data, graph representations, or simulated interactions with the environment. Given a problem instance I, a learning-based heuristic can be represented as a parametric function

$$h_\theta(I, s) \rightarrow \{x^*, x_i^*\}$$

where s denotes the current optimization state and θ the learnable parameters of the model. Depending on the optimization strategy, the model may either directly generate a complete solution x^* in a single step or iteratively construct it through a sequence of decisions x_i^*. This distinction gives rise to the two main paradigms commonly adopted in learning-based CO approaches: *one-shot optimization* and *sequential optimization*.

2.7.1 One-shot optimization

One-shot optimization approaches aim to solve the optimization problem globally within a single decision step. Instead of progressively constructing a solution through sequential decisions, these methods directly infer a complete solution structure satisfying the optimization objective and associated constraints. Formally, the objective consists in directly identifying the optimal solution x^*, which may correspond to a graph structure such as a subgraph, a path, or, in the context of our application, a critical attack path.

Mapping to attack path planning. In attack path planning, one-shot optimization commonly appears in shortest-path formulations, graph optimization methods, or global reasoning frameworks where the entire attack strategy is computed simultaneously from a static representation of the environment. Such approaches can provide stronger guarantees regarding global optimality and are generally effective in relatively small or fully observable environments. However, their applicability rapidly becomes limited in large-scale graph environments due to the combinatorial explosion of feasible graph structures.

Learning-based approaches. Learning-based one-shot optimization methods are typically divided into supervised and unsupervised approaches.

- **Supervised methods** rely on datasets containing expert demonstrations or solutions generated by optimization solvers, enabling models to learn optimization policies through imitation. Although this can produce effective heuristics, generating training labels for large-scale instances requires computationally expensive generations [134]. As a consequence, in practice, labels are typically generated using existing heuristics, introducing an inherent performance ceiling: models are trained to mimic potentially suboptimal solutions and cannot surpass this upper bound [135, 136]. Supervised approaches have been extensively studied in the literature, including recent end-to-end methods that directly generate complete solutions, such as diffusion-based models [137], as well as neural improvement heuristics [138]. The latter departs from full solution construction by adopting an iterative refinement strategy in which existing solutions are progressively improved through local decisions. This shift simplifies the learning problem while still achieving competitive performance.
- In contrast, **unsupervised approaches** avoid explicit supervision by directly optimizing differentiable relaxations of the discrete optimization problems via concave surrogate losses, which are optimized in an unsupervised manner [136, 139]. These approaches have been explored using various architectures, including GNNs [140], hypergraph neural networks [141], physics-informed neural networks [142], and variational annealing [136]. While these methods reduce dependence on labeled data and expensive solvers, they often struggle to incorporate complex combinatorial constraints [134], typically rely on approximations that may weaken guarantees regarding feasibility and optimality [135], and result in models that may converge to suboptimal local minima [141].

2.7.2 Sequential optimization

In sequential optimization, the solution is constructed incrementally through a sequence of decisions. At each step, the optimizer selects new decision variables, which in GCO settings typically correspond to graph elements such as nodes or edges. The solution (e.g., a path) therefore evolves progressively during the optimization process.

Mapping to attack path planning. In the context of network-level attack path planning, this corresponds to modeling the attack path as a progressive selection of vulnerabilities and hosts. Successful choices modify the operational graph by revealing new attack opportunities, privileges, or reachable hosts. Sequential optimization is therefore particularly well-suited for dynamic and partially observable environments, where future decisions strongly depend on previous attack outcomes and newly discovered information, since choices are deferred step by step.

Learning-based approaches. Among learning-based optimization approaches, RL has emerged as a dominant paradigm for solving sequential CO problems.
In CO problems, the underlying state generally encodes the current partial solution and the problem structure, while actions correspond to iterative optimization decisions that progressively construct the final solution. Depending on the application, the environment representation may range from fully enumerated state-transition systems to more abstract graph-based formulations.

Rather than explicitly exploring the exponentially large solution space, the learned policy approximates a heuristic that sequentially navigates toward high-quality solutions through reward-driven optimization. This formulation is particularly advantageous in our context, as the policy inherently restricts the search to feasible transitions without requiring the explicit generation of all possible attack paths or graph states. In contrast to other approaches, RL optimizes the objective through sequential decision-making, relying on step-wise evaluative feedback rather than fixed supervision over entire trajectories, as in supervised learning or imitation learning [143]. Instead of relying on static labels for complete solutions, RL uses reward signals to assess solution quality over time. This framework enables agents to go beyond the limitations imposed by predefined datasets and known reference solutions.

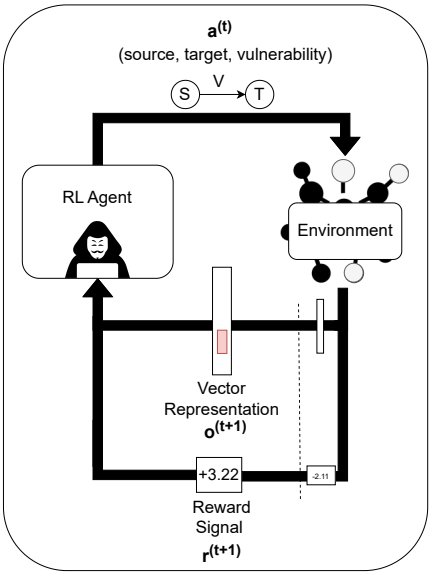


Figure 2.1: Example of RL-driven attack path planning using a POMDP.

However, the key challenge of RL lies in the inherently sparse reward structure of GCO problems, where meaningful evaluations are typically only available at the end of an episode, making training difficult [136, 134]. Prior work has addressed this limitation by introducing intermediate reward signals to alleviate sparsity and improve learning efficiency [144].

In attack-path planning with RL, the POMDP formulation is particularly relevant because attackers typically operate with incomplete and evolving knowledge of the target environment. The agent progressively discovers information about hosts, services, vulnerabilities, credentials, and network configurations through reconnaissance and exploitation actions. Therefore, the complete underlying attack graph is not initially available, and the agent must continuously infer hidden aspects of the environment while adapting its strategy over time.

As illustrated in Figure 2.1, partial observability can be represented through the limited visibility of graph nodes and their attributes, where only a subset of the network topology and associated information is accessible at each timestep. Moreover, actions can be defined at the granularity of graph transitions, where the agent selects an attack step composed of a source node, a target node, and an exploitation technique or vulnerability. This GCO formulation can allow the RL agent to learn attack strategies by exploring the graph structure and identifying effective sequences of actions leading towards specific objectives.

2.8 Existing heuristics solutions

Existing approaches for attack graph analysis are summarized in Table 2.2, which categorizes methods into symbolic and learning-based approaches. The table further highlights representative works according to their underlying graph representation and primary optimization objective. Traditional symbolic approaches for attack graph analysis typically rely on manually designed algorithms and heuristics, such as shortest-path computations or minimum-label cut techniques. While effective in constrained settings, these methods often struggle to scale or

Table 2.2: Categorization of heuristic and learning-based approaches for attack graph analysis and critical component identification.

References	Type	Graph Representation	Objective
Manual / Analytical Heuristics			
[12, 13]	Minimum Label Cut	Custom Attack Graph Representation	Critical Attack Set
[145]	Shortest Path	Exploit Dependency Graph	Target-Reaching Path
[14]	Monte Carlo Search	Host-Based Attack Graph	Critical Attack Set
[146]	Road Sorting Algorithm	Custom Attack Graph Representation	Complex Path Discovery
[147]	Depth-First Backward Search	Exploit Dependency Graph	Hidden Path Discovery
[148]	Probabilistic Planners	Custom Attack Graph Representation	Optimal Path Discovery
Learning-Based Heuristics			
[149, 150, 151, 110, 152]	Ranking	Exploit Dependency Graph	Critical Asset Ranking
[153, 125, 128, 127]	RL	Exploit Dependency Graph	Target-Reaching Path
[154, 155]	RL	Custom Attack Graph Representation	Full Network Compromise
[154]	RL	Custom Attack Graph Representation	Optimal Path Discovery
[156]	RL	Host-Based Attack Graph	Target-Reaching Path
[157]	RL	Host-Based Attack Graph	Full Network Compromise
[158]	RL	Network Traffic Derived Attack Graph	Valid Paths Generation
[159]	PageRank	Exploit Dependency Graph	Defendable Node Ranking

adapt to large, dynamic, and heterogeneous network environments. To address these limitations, recent research has increasingly explored learning-based approaches that automatically discover attack paths, prioritization strategies, and other attack behaviors directly from attack graph representations.

Interestingly, unlike many other GCO applications, neither unsupervised learning nor supervised classification approaches have been extensively explored for critical attack path identification. This is largely due to the combinatorial explosion in the number of feasible attack paths, which renders direct one-shot classification intractable. Instead, the problem is more naturally formulated as a ranking task, in which candidate paths are ordered by their estimated criticality. Such formulations are more scalable, as they only require learning the relative importance of candidate paths. Alternatively, RL methodologies, which currently dominate the literature, address the problem by modeling attack-path discovery as a sequential decision-making process. Beyond naturally capturing attacker behavior and attack progression, RL approaches can accommodate partial observability and decompose complex attacks into intermediate steps, as discussed previously. Our work is positioned alongside RL-based approaches because of their inherent

advantages. Furthermore, we focus on host-based attack graphs, which reduce environmental complexity while preserving the fundamental challenges outlined in Section 2.3.

2.9 Summary

This section introduced the problem of automated attack path planning and the motivations for developing scalable approaches that operate in modern network infrastructures. We discussed the limitations of traditional attack graph methodologies, particularly their difficulty in scaling to large, heterogeneous, dynamic, and partially observable environments due to their full enumeration and granularity.
To address these limitations, attack path planning was reformulated as a GCO problem, where the objective is to identify the most critical or impactful attack strategies within attack graphs. Particular emphasis was placed on low-granularity graph representations, which are better suited to supporting dynamic environments and partially observable settings.
The chapter then examined the limitations of exact optimization methods, highlighting how their computational complexity rapidly becomes prohibitive in realistic attack graph environments. This challenge motivates the adoption of heuristic-based search approaches. However, manually designed heuristics are often difficult to engineer, maintain, and generalize across evolving infrastructures and threat landscapes. These shortcomings motivate the use of ML techniques to automatically learn optimization heuristics from data or simulated interactions. Existing studies demonstrating the application of ML-based heuristics to attack path planning were reviewed and discussed. In particular, RL was identified as a particularly promising approach, as it naturally formulates attack path planning as a sequential decision-making problem, closely mirroring the step-by-step nature of real-world attack strategies.

Link to the thesis objectives

Before introducing the approaches proposed in the subsequent chapters, this chapter established the foundations that motivate the modeling choices adopted throughout this thesis. In particular, it justified the use of a POMDP in conjunction with host-based attack graphs, as well as the formulation of critical attack-path discovery as a GCO problem. The chapter also reviewed existing approaches to attack-path prioritization, covering the different methods and the objective functions commonly employed in the literature. These concepts provide the foundation for the solution developed in this thesis, which is presented in detail in Chapter 3. There, we introduce a POMDP-based framework for RL-driven attack path planning and establish the methodological and experimental choices that underpin the study presented in Chapter 6, where RL approaches are investigated to learn effective policies for critical attack-path discovery.

Part II

A learning framework for attack path planning

A POMDP-BASED SIMULATION FOR ATTACK PATH PLANNING

Summary: *This chapter introduces the first contribution of this thesis: a simulation framework for RL training and evaluation that models attack-path planning as a POMDP. The framework is designed to capture the sequential nature of attack-path decision-making, enable the effective prioritization of critical attack paths, and support automated scenario instantiation and more realistic state transitions through the integration of expert knowledge and data-driven cybersecurity intelligence.*

Contents

3.1 Introduction **47**
3.2 Existing cyber-range platforms **49**
3.3 Methodology **51**
3.3.1 POMDP design 51
3.3.2 Automated POMDP environment construction 57
3.3.3 Domain randomization 61
3.4 Implementation **62**
3.5 Summary **64**

3.1 Introduction

The success of an RL agent in performing a given task depends critically on how accurately the underlying MDP represents the real-world system in which the agent is expected to operate. In other words, it is necessary to create representations that are well-grounded in reality, a challenge commonly referred to as the *grounding problem* [23].
In an ideal scenario, an RL agent would be trained directly in the real application setting, learning through trial-and-error interactions until it converges on policies that solve the task (sub-)optimally [160]. However, in the automated attack planning framework, such direct training is infeasible due to several fundamental limitations: (1) executing real exploits would require a fully operational infrastructure composed of configurable and intentionally vulnerable machines, together with an actuation layer capable of translating the agent's decisions into concrete actions, resulting in substantial deployment and maintenance costs as well as requiring continuous updates; (2) the action execution and reset time associated with each training episode would make the millions of interactions required by RL algorithms impractical in effective training conditions [161]; and (3) achieving effective generalization would require the generation of thousands of diverse network scenarios featuring varying topologies, configurations, and

vulnerabilities, a requirement that is difficult to satisfy in practice without incurring substantial costs.
These challenges motivate the development of simulation or emulation environments that approximate the real-world dynamics of attack path planning and prioritization as an MDP. While emulation can more readily capture certain aspects of real-world network behavior, it remains constrained by high action execution times. As a result, it still struggles to support the large number of interactions and training iterations that RL algorithms require to learn effective policies.
Consequently, simulation is generally preferred for RL training, provided that efforts are made to minimize the *sim-to-real gap* [160, 162]. In this paradigm, the computational burden of large-scale interactions needed during training is handled by fast interactions, while real-world deployment—if feasible—is typically limited to (post-training) inference over a limited number of episodes. However, several challenges must be addressed when designing a simulation-based MDP that accurately captures the characteristics and constraints of the attack-path planning problem. [163, 16, 164, 165, 166]. Specifically, the realism gap is driven by three main factors: (1) identifying the right level of detail and granularity for state and action representations to make learning meaningful [165]; (2) deriving appropriate dynamics for state transitions and reward functions; and (3) generating network scenarios that represent real-world topologies, vulnerability distributions, and their allocations.

Contributions

To address the challenges associated with designing a realistic and scalable POMDP capable of supporting the learning of sequential decision-making strategies for attack-path prioritization, this chapter introduces three main contributions:

- An enhanced **POMDP formulation** of the attack-path planning problem based on host-level attack graphs, which strikes a balance between modeling fidelity and scalability while incorporating expert knowledge through vulnerability metadata to enable meaningful state transitions.
- An **automated scenario generation pipeline** capable of producing diverse scenario instances by integrating real-world vulnerability data and configurable realism parameters, thereby reducing the manual effort required to construct training and evaluation scenarios.
- A **domain randomization framework** designed to generate extensive scenario variability across topologies, configurations, and vulnerability distributions, promoting policy generalization and enabling more robust RL evaluation.
- **Continuous CyberBattleSim** (*C-CyberBattleSim*), a simulation framework built as an extension of Microsoft's *CyberBattleSim* [167], capable of simulating the designed POMDP and supporting the proposed enhancements.

Together, these contributions aim to improve the realism, scalability, and practical relevance of RL training and validation scenarios, while maintaining the computational efficiency required for large-scale learning.

3.2 Existing cyber-range platforms

A *cyber-range platform* is a controlled and reproducible virtual environment designed to emulate or simulate computer networks, systems, and cybersecurity scenarios for training, experimentation, and evaluation purposes. These platforms enable the simulation of attacker and defender behaviors by providing configurable network topologies, vulnerabilities, services, and security mechanisms.
Several cyber-range platforms and their corresponding MDP formulations have been proposed to support the learning of sequential decision-making strategies that prioritize attack paths. While these approaches share the common goal of automating proactive attacker behavior, they differ significantly in the level of abstraction at which they operate and the execution model used within these platforms.

Action-space granularity. A first major differentiating factor concerns the *granularity of the actions* represented in the MDPs underlying these platforms. Some approaches model attack-path planning at the level of concrete tools and frameworks, such as Metasploit, where the agent selects from specific exploits, payloads, or commands provided by the framework [166]. Such simulators or emulators are designed to train agents whose decisions can be more readily translated into real-world deployments. However, they remain tightly coupled to the capabilities, assumptions, and limitations of a particular toolset.
In contrast, other approaches abstract away from individual tools and represent actions at a higher level, enabling agents to select generalized attack steps, such as MITRE ATT&CK techniques or vulnerabilities present within the network, independently of the availability or implementation details of specific exploits. This higher level of abstraction promotes more general attack planning and improves transferability across different frameworks, infrastructures, and toolchains. The trade-off, however, is a reduced connection to deployment feasibility, as an additional execution layer is required to translate abstract actions into concrete real-world operations when suitable implementations exist.

Execution model. A second key dimension of differentiation concerns whether the framework is based on emulation or simulation as the execution model [166].
Network *emulators* consist of interconnected virtual machines (VMs), hosted locally or in cloud-based infrastructures, that communicate over virtual networks. In these settings, actions selected by the agent are executed directly on the virtual environment, resulting in high-fidelity behavior that closely mirrors the real world. Consequently, agents trained in emulated environments tend to exhibit better performance when deployed in operational scenarios [166]. However, this realism comes at the cost of high computational overhead and limited scalability. RL typically requires thousands to millions of interaction steps to converge, making training in such frameworks time- and resource-intensive and, in turn, limiting extensive experimentation of advanced AI methods.
Scenario *simulators*, on the other hand, rely on abstract models in which the effects of actions are represented through predefined transition functions and human-designed rules. Rather than reproducing the behavior of the underlying infrastructure, simulators directly model the state changes resulting from an action. Although defining such transition models can be challenging and may oversimplify real-world dynamics [16], since many low-level system interactions, network behaviors, and implementation-specific details are abstracted away, simulation-based environments offer substantially faster execution and lower resource requirements than emulation-

Table 3.1: Classification of existing cyber-range platforms by execution model and action-space granularity.

Execution Model	**Action-Space Granularity**		
	Tool-Level	**Technique-Level**	**Vulnerability-Level**
Emulation	–	CyGIL [168] BRAWL [169]	–
Simulation	–	NASim [170] CAGE [171]	CyberBattleSim [167]
Sim. + Emu.	NASimEmu [172] Cyborg [164] PenGym [16] CRLF [173]	CyberWheel [165] FARLAND [174]	–

based alternatives. Consequently, they are particularly well suited for RL research, where agents typically require millions of interactions with the environment, enabling rapid experimentation, large-scale training, and efficient evaluation of novel methodologies [168].
Only more recently, cyber-range environments have aimed to support both paradigms, using simulation for scalable and efficient training and emulation for high-fidelity testing and validation prior to deployment.
Table 3.1 summarizes representative cyber-range platforms that target multi-host, multi-stage attack planning along two dimensions: action-space granularity and the underlying execution model.
It is worth noting that numerous other cyber-range training and evaluation platforms exist beyond those included in Table 3.1. However, many of these platforms do not adopt a standardized or clearly defined granularity for action spaces, making them difficult to categorize within the taxonomy presented in this chapter. For example, platforms such as CSLE [175] expose highly simplified traffic-level (continue/stop) actions, whereas CyGym [176] employs an action space consisting of only three high-level actions (e.g., probe, attack, and pass).
To maintain continuity with the attack-path planning literature, which predominantly relies on attack-graph representations in which attack paths are modeled as sequences of vulnerability exploitation, the vulnerability level was selected as the reference action-space granularity for this work, remaining sufficiently abstract to avoid dependence on specific offensive tools, exploits, or attack frameworks.
Furthermore, accurately emulating a large and continuously evolving set of real-world vulnerabilities presents significant practical challenges. Consequently, a simulation-based environment was adopted as the foundation of this work, with `Microsoft CyberBattleSim` serving as the baseline platform for developing the proposed extensions. Simulation also offers several advantages that make it well-suited for RL research, including extensive customization, rapid execution, and low computational overhead. These characteristics facilitate large-scale experimentation and provide the flexibility required to investigate the advanced modeling and learning methodologies proposed throughout this thesis.

3.3 Methodology

This section presents the proposed POMDP for attack path planning. It details the formulation of its transition dynamics, the data-driven pipeline for generating realistic scenarios, and the randomization mechanisms introduced to enhance diversity.

3.3.1 POMDP design

A primary contribution of this thesis is the development of a novel POMDP formulation for attack-path prioritization that explicitly captures the challenges of partial observability and progressive network discovery encountered in real-world attack planning. The formulation is further grounded in real-world vulnerability data, leveraging expert-curated metadata to derive realistic action outcomes and state transitions.
To achieve this objective, the proposed framework incorporates several key design choices aimed at balancing realism and scalability:

- The POMDP **transition** and **reward functions** are derived from both network characteristics and vulnerability metadata obtained from established cybersecurity databases, including standardized assessment frameworks such as CVSS. By grounding environment dynamics in expert-curated information, the proposed formulation provides a more realistic approximation of attack progression and attacker decision-making. Unlike prior simulators, in which the effects of individual vulnerabilities were manually specified by the environment designer, the proposed approach limits manual modeling to high-level network interaction rules [23], while vulnerability-specific behavior is inferred directly from associated expert-annotated metrics.
- A set of tunable parameters is introduced to **control** multiple dimensions of **scenario complexity**, allowing the simulation training and validation scenarios to be calibrated to different levels of structural difficulty representative of the operational environment of interest.
- While most prior work typically formulates attack path planning as the search for a sequence of actions leading to the compromise of a specific target asset [148], we adopt a **target-agnostic perspective**. The objective is to reach the maximum attainable level of network penetration across all hosts [11], thereby capturing a more general notion of network security and overcoming the inherently target-specific nature of these formulations.
- The agent's objective is generalized through a **customizable attacker profile** (i.e., threat model), enabling flexible definitions of desired security impact and allowing the environment to reflect diverse adversarial goals.
- We **relax** the common **monotonicity assumption** present in existing attack graph models—where attackers never lose capabilities and never need to backtrack [11, 120]. By allowing disruptive actions and non-monotonic state transitions, the proposed formulation more accurately reflects attacker behavior in realistic campaigns.

POMDP components

Let $G = (\mathcal{N} \cup \mathcal{A} \cup \mathcal{V} \cup \mathcal{O}, \mathcal{E} \cup \mathcal{D} \cup \mathcal{F})$ be a multipartite graph whose nodes are partitioned into disjoint subsets, where:

- $\mathcal{N}$ denotes the set of host nodes,
- $\mathcal{A}$ denotes the set of vulnerable assets (e.g., services),

- $\mathcal{V}$ denotes the set of vulnerability nodes,
- $\mathcal{O}$ denotes the set of obstacles, inspired by recent work on cyber terrain modeling [127]
- $\mathcal{D} \subseteq \mathcal{A} \times \mathcal{V}$ encodes the assignment of vulnerabilities to assets,
- $\mathcal{E} \subseteq \mathcal{N} \times \mathcal{A}$ encodes the assignment of assets to hosts,
- $\mathcal{F} \subseteq \mathcal{N} \times \mathcal{O}$ encodes the assignment of obstacles to hosts.

The resulting POMDP scenarios are constructed as a function of the graph instance, an attacker profile, and a difficulty matrix:

$$\mathcal{M} = f(G, \Pi, D_M).$$

The *difficulty matrix* (D_M) parameterizes the ease or difficulty of state transitions. This matrix serves as a calibration mechanism to regulate the overall complexity of the scenarios and to control the stochastic dynamics of action outcomes.
We further define an *attacker profile* Π [7] as a function that specifies the adversarial behavior that the RL agent is expected to learn, expressed as a set of step-level rewards or penalties associated with achieving specific attack objectives (e.g., credential access). Rather than explicitly defining a sequential attack strategy, the attacker profile provides local guidance on the expected impact of individual actions, allowing the agent to learn effective strategies through interaction with the environment. Therefore, the attacker profile directly influences the reward structure and, consequently, the optimization objective of the POMDP.
The model dynamics are defined by several components, following the classical decomposition introduced in Section 2.6.3:

$$\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{O}, T, Z, R, \rho_0 \rangle,$$

Each component of the POMDP (with their interactions represented in Figure 3.1) is systematically derived from (i) the structural properties of the graph, (ii) the behavioral specification encoded in the attacker profile, and (iii) the transition calibration induced by the difficulty matrix, as detailed in the following paragraphs.

State space ($\mathcal{S}$). Each node x_i in the multipartite graph, with $x_i \in \mathcal{N} \cup \mathcal{V} \cup \mathcal{A} \cup \mathcal{O}$, is associated with a feature vector

$$x_i = (f_{i,1}, f_{i,2}, \ldots, f_{i,K_i}) \in \mathbb{X}^{K_i},$$

where the features encode the specific characteristics of the corresponding node type. For host nodes $\mathcal{N}$, these features include, for instance, the operational status, compromise state, or discovery state of the host. For vulnerability nodes $\mathcal{V}$, the feature vector encodes vulnerability metadata, such as CVSS components (e.g., attack vector, attack complexity, required privileges, and impact metrics), as well as the set of outcomes enabled by that vulnerability. For asset nodes $\mathcal{A}$ and obstacle nodes $\mathcal{O}$, the features describe their type and relevant characteristics (e.g., service version, firewall rule characteristic). All these features represent the state at a given timestep and can be modified by the agent through its actions, with their evolution encoding the temporal progression of the episode.
The global state is defined as the Cartesian product of the feature spaces of all nodes, together with a representation of the edge relations:

$$s = (x_1, \ldots, x_{|\mathcal{N}|+|\mathcal{V}|+|\mathcal{A}|+|\mathcal{O}|}, \mathcal{D}, \mathcal{E}, \mathcal{F}) \in \mathcal{S}.$$

In practice, the state can be represented by any encoding (e.g., concatenation) of these feature vectors. Edge information can either be encoded explicitly (e.g., via adjacency representations) or, more compactly, incorporated into the node feature vectors themselves, thereby capturing

relational information without explicitly storing the full adjacency structure in the state. Moreover, a host-based attack graph representation can be derived from this POMDP formulation by embedding vulnerabilities, assets, and obstacles directly into host-node attributes rather than representing them as separate node types in the graph. In this formulation, entities associated with multiple hosts may be replicated as host-local attributes when necessary, while the environment dynamics continuously update these attributes throughout the attack progression.

Action space ($\mathcal{A}$). We consider a family of action spaces that support different levels of granularity and modeling fidelity, enabling a customizable trade-off between required complexity and granularity.

1. **Node–Vulnerability.** The simplest action space is

   $$\mathcal{A}_1 = \mathcal{N} \times \mathcal{V},$$

   where an action $a = (n, v)$ denotes attempting to exploit vulnerability v on target node n, abstracting away the source of the attack.
2. **Source–Target–Vulnerability.** To explicitly model both the target of the vulnerability and the source pivoting the attack, we define

   $$\mathcal{A}_2 = \mathcal{N} \times \mathcal{N} \times \mathcal{V},$$

   where $a = (n_s, n_t, v)$ represents exploiting vulnerability v from source node n_s toward target node n_t. This follows the logic of the original `CyberBattleSim` simulator and enables distinguishing between local and remote exploits based on whether the source and target coincide.
3. **Source–Target–Vulnerability–Outcome.** Finally, to further enhance realism and express attacker intent, we define

   $$\mathcal{A}_3 = \mathcal{N} \times \mathcal{N} \times \mathcal{V} \times \Omega.$$

   While the previous representations implicitly consider the exploitation of all outcomes enabled by a vulnerability, this extension makes the outcome dimension explicit, which is more realistic since, in practice, some vulnerabilities may enable many outcomes or even general code execution, and the attacker chooses one at each transition. The set Ω is a finite set of possible customizable outcomes (e.g., inspired by MITRE ATT&CK tactics or techniques).

In the remainder, we denote by $\mathcal{A}$ any of the above action spaces, a dimension which can be parameterized depending on the desired modeling fidelity and decision complexity.

Initial state distribution (ρ_0). The initial state distribution ρ_0 encodes the attacker's initial foothold and knowledge at the beginning of each episode. We assume that the attacker initially controls exactly one host $n_0 \in \mathcal{N}$, which serves as the entry point into the network, modeling either an externally compromised host or a host controlled by an internal user launching the attack.
Formally, we first sample an initial host n_0 according to a distribution p_0 over $\mathcal{N}$, and then initialize the corresponding host features to a discovered and compromised state while all other hosts are initialized to a baseline configuration:

$$\rho_0(s) = \sum_{n \in \mathcal{N}} p_0(n), \mathbb{I}[s = s_0^{(n)}],$$

where $s_0^{(n)}$ denotes the initial state with host n compromised. A common choice is a degenerate distribution that assigns probability 1 to a fixed start node, but this encourages learning specific patterns tied to that start node. A uniform distribution $p_0(n) = 1/|\mathcal{N}|$ yields a more independent evaluation and enhances generalization capabilities during training. More generally, p_0 can be parameterized to exclude hosts that are not externally reachable or that cannot serve as entry points under specific threat models or network assumptions.

State transition function ($T(s' \mid s, a)$). The state transition function determines whether the preconditions for exploiting a vulnerability are satisfied in the current environment state. If these conditions are met, the corresponding postconditions are applied to generate the resulting successor state. To do so, it inspects attributes of hosts, vulnerabilities, assets, and obstacles, as well as their relations in the graph. The following description assumes the most expressive action space $\mathcal{A}_3$ and distinguishes between *deterministic pre-conditions* for an action to be feasible and *stochastic factors* for an action to be effective. Table 3.2 lists conditions and factors indicating from which components of the graph they are derived.

- **Deterministic pre-conditions.** An action $a = (n_s, n_t, v, \omega)$ is considered feasible only if all deterministic pre-conditions listed in Table 3.2 are satisfied. The source host must be controlled and operational; the target host must be discovered and operational; the selected vulnerability must be present on the target; and the intended outcome must be supported by that vulnerability. Required privileges are also verified by comparing vulnerability requirements with the current state of the target host. The compatibility between source and target is enforced by the CVSS attack vector component of the vulnerability, indicating whether the vulnerability can be exploited locally; in that case, it requires $n_s = n_t$. The node features are also used to validate outcomes involving collection or exfiltration (e.g., whether relevant data exists and is accessible). Obstacles such as firewalls may block incompatible remote exploits based on their configuration and placement in the topology. In particular, if a vulnerability is associated with a service running on a port where also a blocking firewall rule is present, and the attack is remote, the obstacle blocks the exploitation.
- **Stochastic factors.** If all deterministic conditions are satisfied, the actual outcome of the action is governed by a set of stochastic factors representative of some probabilistic aspects. First, the probability of success of an exploitation attempt is modeled under the simplifying assumption of independence between actions, following [148]. If the goal is only to enumerate possible attacks under idealized conditions, this stochastic effect can be disabled, effectively assuming a "perfect" attack behavior [7]. Second, for vulnerabilities involving information disclosure, the amount of new knowledge revealed (e.g., newly discovered nodes or data) is also modeled stochastically and derived from the CVSS *confidentiality impact* metric. Both stochastic aspects are parameterized by D_M, which maps discrete CVSS attributes (*attack complexity* and *confidentiality impact* discrete values) to probabilities or expected information gains.

When an action succeeds, the postconditions update host features to reflect the outcome selected for the vulnerability exploitation (e.g., denial-of-service disrupting a node, data collection, or control changing its status), thereby producing the next state s'. If an action is infeasible or fails, the state remains unchanged, i.e., $s' = s$.

Episode termination is also evaluated within the transition function. To model threat model completion, we pre-compute the maximum attainable impact under the given attacker profile from the graph G [11], starting from the initial entry host of the episode $derivedby(\rho_0)$, and terminate the episode when this level is reached (e.g., when all hosts that are controllable under

Condition / Factor	Type	Derived from
Source node is controlled and not disrupted	Deterministic	Host features in $\mathcal{N}$
Target node is discovered and not disrupted	Deterministic	Host features in $\mathcal{N}$
Vulnerability present on target	Deterministic	Edge sets $\mathcal{D}$ and $\mathcal{E}$
Outcome supported by vulnerability	Deterministic	Vulnerability features in $\mathcal{V}$
Required privileges satisfied	Deterministic	Vulnerability & host features ($\mathcal{N}$ and $\mathcal{V}$)
Attack vector compatibility (local/remote)	Deterministic	Vulnerability features in $\mathcal{V}$
Data available for collection/exfiltration	Deterministic	Node features $\mathcal{N}$
Obstacles (e.g., firewall) blocking exploit	Deterministic	Obstacle & host features ($\mathcal{O}$, $\mathcal{N}$, and $\mathcal{F}$)
Exploit success probability	Stochastic	CVSS attack complexity (in $\mathcal{V}$)
Information disclosure magnitude	Stochastic	CVSS confidentiality impact (in $\mathcal{V}$)

Table 3.2: Deterministic and stochastic factors enforced by the state transition function in the proposed POMDP.

that starter host are effectively controlled, in a *control* attacker profile).

Observation function (Z). To model partial observability, the agent does not have direct access to the complete environment state s_t. Instead, the observation function

$$o_t = Z(s_t)$$

provides a partial representation of the underlying graph scenario at a given timestep. This observation is determined by the current knowledge acquired by the agent and is influenced by the visibility constraints imposed by the cyber environment.
First, only the subset of hosts that are currently discoverable according to the agent's knowledge state is revealed. Associated information, including vulnerabilities, assets, and environmental characteristics, is included in the observation only for visible hosts. Second, even when a host is discovered, additional uncertainty may remain due to obstacles or terrain constraints that hide or limit access to specific attributes. Such obstacles can represent realistic cybersecurity conditions, including access-control policies restricting communication between hosts, encrypted services concealing application details, or insufficient privileges preventing the enumeration of system configurations and vulnerabilities. Consequently, feature representations may remain partially observed or obfuscated until the agent performs appropriate discovery actions, obtains additional privileges, or achieves specific interaction outcomes.

Reward function ($R(s, a, s')$). The reward function maps state transitions to real values and is parameterized by the attacker profile Π. The attacker profile assigns bonuses or penalties to outcomes, thereby encoding the attacker's objectives. Formally, we define

$$R(s, a, s') = \sum_{\omega \in \Omega} \Pi(\omega) \cdot \Delta_\omega(s, a, s'),$$

where $\Pi(\omega) \in \mathbb{R}$ denotes the reward weight associated with outcome ω, and $\Delta_\omega(s, a, s')$ indicates whether and to what extent outcome ω has been realized in the transition from s to s'.
This parameterization allows a human designer to specify high-level preferences through the outcome dimension of Π, resulting in different reward functions and, consequently, different learned attacker behaviors. For example, under a control-oriented profile, credential access may receive a high positive reward, while denial-of-service outcomes incur a penalty. Instead, if the objective is network exploration, discovery actions may be rewarded while outcomes deemed irrelevant receive zero reward.

Optionally, when relative importance varies across hosts, rewards are further scaled using node values inspired by the concept of crown jewels [128], for instance, derived from the assets present on each host. Let $V(n)$ denote the value associated with host n. The final reward can then be written as

$$R(s, a, s') = \sum_{n \in \mathcal{N}} V(n) \sum_{\omega \in \Omega} \Pi(\omega) \cdot \Delta_{\omega,n}(s, a, s'),$$

where $\Delta_{\omega,n}(s, a, s')$ indicates that outcome ω has affected host n in the transition. For outcomes involving multiple hosts (e.g., discovery revealing several neighbors), the reward naturally accumulates over the set of affected nodes. This formulation preserves a target-agnostic objective centered on overall network compromise while allowing different graph components to contribute unequally to the reward through host-specific weighting factors.

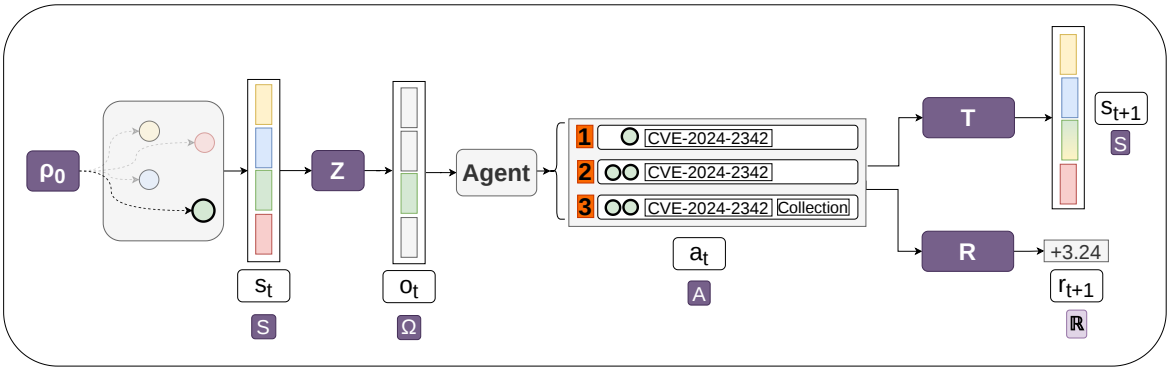


Figure 3.1: Overview of the proposed POMDP design and interaction loop between agent and environment, including the roles of the observation function O, transition dynamics T, reward function R, and initial state distribution ρ_0.

Learning in the proposed POMDP

To summarize, Figure 3.1 illustrates a single timestep of interaction with the proposed POMDP, highlighting the roles of the main components ρ_0, Z, T, R and their relationship with the state, action, observation, next state, and reward. The figure provides an end-to-end view of how the environment dynamics, partial observability, and reward assignment interact to drive the learning process.

The proposed POMDP formulation reframes attack-path prioritization as a sequential decision-making problem, eliminating the need for explicit attack-graph construction and analysis. Rather than enumerating and ranking attack paths, an RL agent learns a policy that guides exploration toward high-value regions of the state space through reward-driven interactions. As training progresses and the policy learns, the agent implicitly traverses and prioritizes the most relevant portions of the underlying attack graph, focusing computation on promising attack strategies while naturally disregarding low-value or unproductive branches. The learned policy therefore acts as an implicit pruning mechanism, concentrating inference-time exploration on the most critical attack paths.

Moreover, based on the characteristics discussed above, this design enables the agent to behave similarly to a real-world attacker operating under partial and evolving visibility. This is achieved by: (1) initializing each episode with only a single revealed host according to the initial state distribution; (2) representing subsets of hosts and their features (e.g., vulnerabilities or services)

as partially observable and progressively revealed through state transitions; and (3) allowing vulnerability actions whose outcomes include the discovery of new hosts or additional features of already discovered hosts.

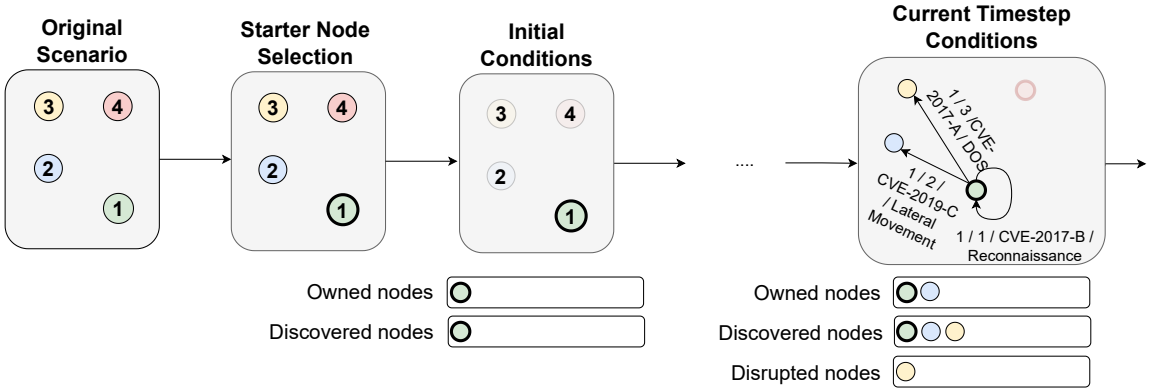


Figure 3.2: Example of a POMDP episode illustrating discovery, exploitation, and impact-driven decision making.

In addition, backtracking is naturally supported by allowing actions that may lead to negative outcomes, which are discouraged through large penalties rather than being explicitly forbidden. This better reflects realistic attacker behavior, in which failing or suboptimal actions are possible but costly. Furthermore, by modeling the initial state distribution as stochastic and keeping both the transition and reward functions target-agnostic, the POMDP approximates the network's average security level rather than optimizing for a single fixed entry point or target, yielding a more robust and general assessment of network exposure than attack graphs can provide.
With this POMDP formulation, a typical episode can be represented at a high level as illustrated in Figure 3.2. The environment logic first selects a starting node according to the initial state distribution, after which the attacker agent begins with a single discovered host and may choose to exploit a local vulnerability. Such an action can produce a reconnaissance outcome that reveals additional hosts. Subsequently, the agent may perform further exploitation actions against the newly discovered hosts, potentially leading to different attack objectives, such as credential access or denial-of-service outcomes that disrupt target nodes. This example illustrates how reconnaissance, exploitation, and impact objectives are naturally interleaved within the proposed framework. Moreover, it highlights the agent's decision-making process, which involves selecting as an action type the tuple composed of source node, target node, vulnerability to exploit, and intended objective of exploitation. The episode is then reset to the initial conditions, enabling further training iterations or evaluation across multiple independent episodes.
The trajectory illustrated in Figure 3.2 represents one possible execution path rather than a predefined sequence of actions. The actual episode evolution depends on the policy learned by the RL agent and its ability to optimize the reward structure defined by the attacker profile. Thus, the framework allows the agent to discover and execute different combinations of reconnaissance, exploitation, and impact actions without explicitly encoding a sequential attack strategy.

3.3.2 Automated POMDP environment construction

The POMDP introduced in Section 3.3.1 can be instantiated directly from a graph (G) comprising hosts, assets, obstacles, and vulnerabilities, along with their associated features, relations, and

allocations. While the complexity mapping and attacker profile can be easily specified by a human expert—through the definition of high-level numerical weights and probabilities—the construction of realistic graph representations and their associated components remains a significant modeling challenge, particularly when a large number of scenarios is required, as is typical in RL for both generalizable learning and robust evaluation.
Indeed, key elements of G include: (i) host characteristics, such as number, type, and value; (ii) the selection and placement of vulnerabilities across hosts; and (iii) the distribution and nature of assets and obstacles, which directly influence the complexity of the decision-making process faced by the agent. Designing such graphs manually is labor-intensive and difficult to scale.
When targeting a specific real-world network scenario, these parameters could, in principle, be derived from scanning tools and data-collection pipelines, thereby enabling the construction of a simulation-based digital twin. However, privacy concerns typically arise when scanning and replicating real infrastructures for training purposes. These concerns are further amplified by the need to scan and generate many scenario instances to ensure robust, generalizable policies. As a result, relying on real-world data introduces privacy and scalability challenges, and a mechanism is required to generate scenario graphs that are both realistic and sufficiently diverse, without depending on sensitive real-world data or excessive manual configuration.
To address these limitations, this thesis proposes an automated pipeline for POMDP parameterization to enable scalable, realistic scenario generation. The objective is to enhance realism while ensuring full automation of graph construction, requiring human input only for a limited set of high-level parameters. In contrast to the original `CyberBattleSim` design, this approach eliminates the need for simulation developers to hard-code graph characteristics. Such manual processes have historically limited both the diversity and scalability of simulated environments. For example, the original `CyberBattleSim` provides only four environment graph scenarios, which are widely reused in the literature, each containing fewer than ten unique vulnerabilities hard-coded in the topologies due to the effort required for manual selection and placement.
In particular, the proposed approach introduces:

1. An **automated vulnerability selection** pipeline, removing the need to manually determine which vulnerabilities to allocate. Given the vast and continually growing number of disclosed vulnerabilities [3], manual selection is, in fact, impractical. Moreover, real-world operational environments often exhibit very characteristic vulnerability distributions. By leveraging automated statistical analyses via internet-wide scanning platforms, the selection process can be aligned with realistic, real-world, and domain-specific patterns.
2. An **automated allocation** solution, enabling systematic and realistic placement of vulnerabilities, obstacles, and assets, together with consistent feature assignment for each host. Vulnerabilities are typically not distributed randomly but are correlated with the assets and software stacks present on a host. Automated mechanisms can enforce such dependencies and perform validity checks to ensure coherent and realistic configurations.

Automated vulnerability selection

The scenario generation pipeline begins by constructing a database of real-world vulnerabilities to populate the graph-based scenarios G_i. As previously discussed, manual vulnerability selection does not scale to the volume and pace of modern disclosures. Moreover, appropriate grouping logic is required to ensure that selected vulnerabilities are representative of specific operational target environments. To address these limitations, we adopt an automated approach in which the user provides only a human-readable query describing the target context, while scanning

tools are used to gather aggregated and anonymized statistics on real-world Internet-connected assets and hosts that match the specified query.
In particular, CTI platforms such as *Shodan* (Step 1 in Figure 3.3) can be used to identify the most common assets in a given context, including services and their versions, OSs, and other relevant host attributes. Example queries may target specific geographic regions, IP ranges, cloud providers, or industry sectors. These tools provide statistical prevalence information—such as device counts per asset version—thereby offering a practical approximation of real-world asset distributions. Such statistics can subsequently guide the placement of assets (and hence their associated real-world vulnerabilities) in the generated graph, ensuring that simulated scenarios reflect realistic distributions.
For instance, a query may focus on hosts deployed under a particular cloud provider within a defined geographic region. The resulting output reports how many hosts run each OS or service version. These frequency distributions can then be used to generate multiple scenarios featuring diverse yet statistically grounded combinations of these assets, while aligning to the real-world statistical frequencies.
Once assets and their versions have been identified, a vulnerability repository such as the NVD is queried to retrieve vulnerabilities associated with each asset-version pair. Along with vulnerability identifiers, we collect CVSS vectors, textual descriptions, and CWE identifiers. CWE mappings can later be used to infer possible ATT&CK techniques and tactics, as discussed in Section 1.4.3. This metadata is stored to parameterize state transitions and reward structures within the POMDP framework (see Section 3.3.1).
As a result, for each asset and its specific version, the pipeline produces: (i) a statistical frequency reflecting its prevalence in the target context, (ii) an associated set of vulnerabilities, and (iii) structured metadata describing their characteristics for proper modeling of transition functions. Importantly, this pipeline preserves privacy by relying exclusively on publicly available, aggregated Internet-scale data. It operates at the asset-distribution level rather than modeling individual real-world hosts, thereby avoiding the replication of sensitive infrastructures while still enabling the construction of realistic vulnerability sets derived from trusted sources.

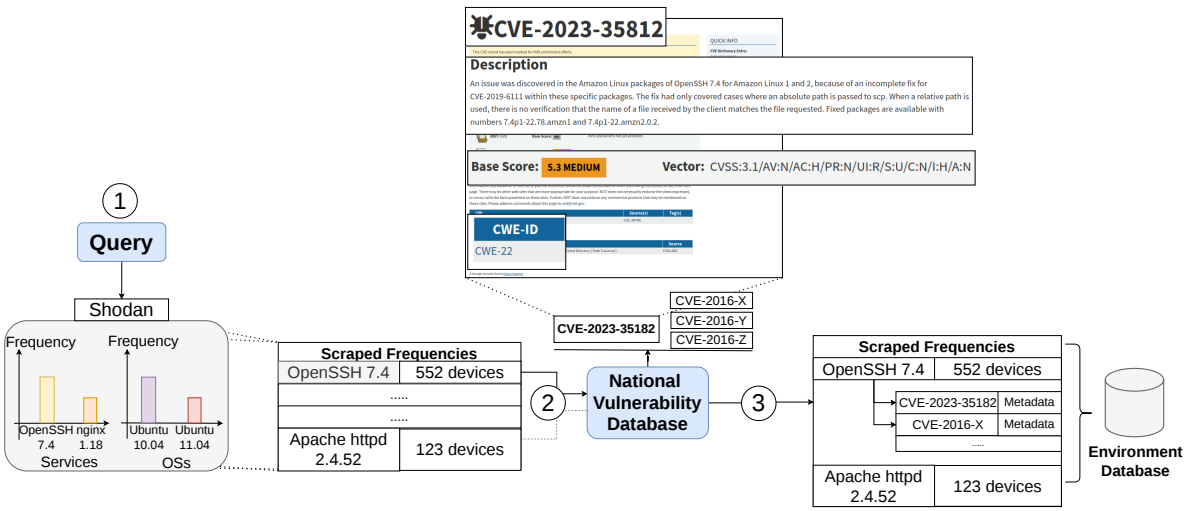


Figure 3.3: Overview of the automated scenario generation pipeline, where assets and their associated vulnerabilities are collected from a CTI tool and vulnerability database, then stored in the environment database for scenario generation.

Automated vulnerability allocation

The second stage of the pipeline focuses on generating the concrete scenario graph by leveraging the asset statistics and vulnerabilities collected in the previous step, along with a controlled set of high-level parameters that the user can configure. The objective is to systematically populate the graph-based scenario without manual configuration, except for a small set of intuitive controls, and to ensure that the resulting distributions reflect realistic deployment conditions.

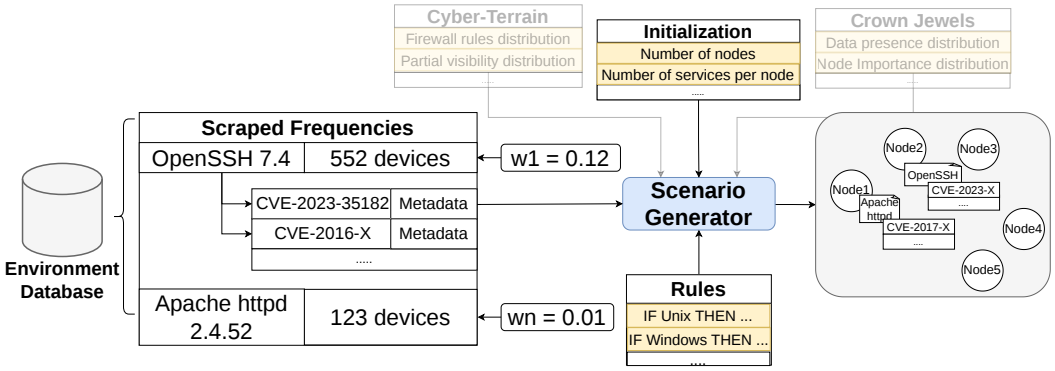


Figure 3.4: Overview of the automated graph scenario construction pipeline: graph initialization and vulnerability allocation.

As illustrated in Figure 3.4, the process begins with a set of initialization parameters, such as the *number of nodes*, which are used to create an empty graph. Each node is then assigned a host type (e.g., *Unix-only host*) and a capacity specification, such as the maximum number of services it can run, the number of guest OS instances it can support, or its data storage capacity.
To ensure variability across hosts, the simulator allows node-dependent parameters (e.g., the number of services per host, the host type distribution) to be modeled as proper distributions rather than fixed quantities. This enables users to control proportions at a high level, for example, by specifying the probability of different host types, while maintaining diversity across generated instances.
Next, asset versions and their associated vulnerabilities are assigned to hosts. This automated assignment is designed to both satisfy host-specific compatibility constraints and approximate the real-world distributions derived from the CTI query:

- A set of **validity constraints** (rules) ensures that assets are assigned only to compatible hosts (e.g., *Ubuntu* OS on Unix-compatible hosts, or *Microsoft Outlook* service on hosts with *Windows* as OS). These rules are integrated into the assignment procedure, restricting the set of assets (and hence vulnerabilities) that can be sampled for each host.
- Proper **realism features** through weighted sampling are achieved by biasing the selection among valid assets according to their observed prevalence. The associated weights are derived from device counts returned by the CTI query, which are always greater than zero for the considered assets, and are mapped through configurable functions. For instance, a logarithmic transformation can be applied as

$$w_i = 1 + \log(\text{device count}_i)$$

to reduce the dominance of highly prevalent assets while preserving their higher likelihood of selection. This approach ensures that commonly deployed services and platforms are

sampled more frequently, while maintaining sufficient diversity in the generated scenarios. The specific weighting function can be tuned to control the extent to which generated environments reflect real-world deployment distributions, allowing a trade-off between realism and scenario variability.

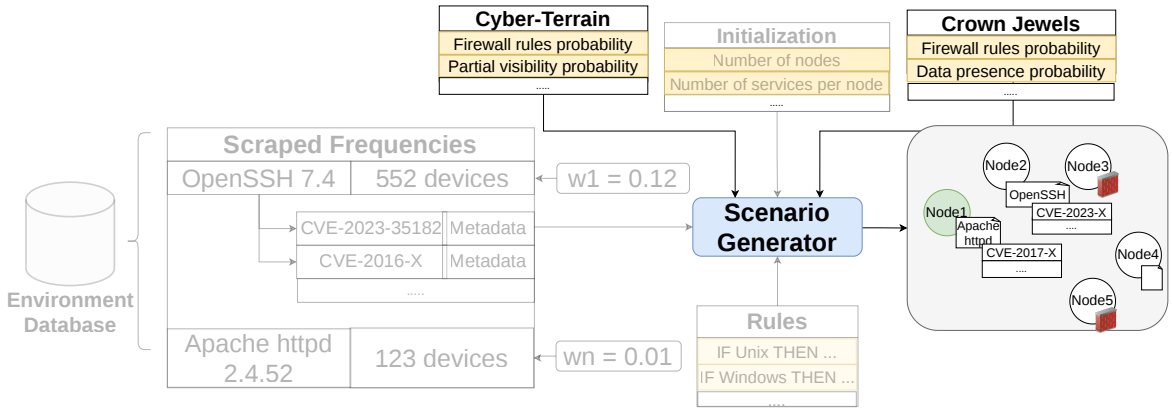


Figure 3.5: Overview of the automated graph scenario construction pipeline: cyber-terrain and crown jewels allocation.

Cyber-terrain features, such as firewalls and access control characteristics, are incorporated in a subsequent step (Figure 3.5). For example, firewall configurations can be instantiated based on the ports exposed by the sampled services. Their presence and impact can again be determined according to user-defined distributions, enabling high-level control over both realism and diversity.
Additional host-level attributes may include default user privileges upon lateral movement, visibility constraints, or privilege escalation requirements. These cyber-terrain features directly affect which vulnerabilities are exploitable (e.g., services filtered by firewalls), whether additional actions (e.g., privilege escalation) are required before exploiting some vulnerabilities, and consequently influence agent interactions with the POMDP scenarios, including the state transitions described in Section 3.3.1.
Finally, nodes are assigned attributes related to their importance (noted as "crown-jewel" assets). These include a *data presence* indicator, which specifies whether a host contains assets that yield rewards upon collection, and a *node importance value*, a continuous metric representing the potential operational impact of compromising that host. This metric can be used to enable proper reward scaling (see Section 3.3.1) and to fine-tune the placement and concentration of high-value targets within the graph.
Through this overall automated parameterization process, the resulting graph scenario reflects realistic network characteristics while preserving the flexibility and scalability required for large-scale scenario generation. Human input is limited to high-level scalar parameters or probability distributions, ensuring easier and broader control over the scenario instances.

3.3.3 Domain randomization

The scenario generator pipeline is extended with a domain randomization mechanism [177] that enables controlled multi-instance generation. Rather than producing a single graph realization from fixed parameters, the generator operates over a parameter space, sampling a new configuration at each instantiation. This transforms training and evaluation from being performed on a

fixed simulation instance to being conducted over a distribution of stochastic scenarios.
This extension can serve for two reasons: (1) training over multiple sampled scenarios reduces structural overfitting, encouraging the agent to learn strategies that depend on underlying attack principles rather than memorizing specific scenario characteristics or vulnerability placements; (2) evaluation over a large set of scenarios provides more reliable performance estimates, mitigating the risk that results are biased by a particular or a limited number of graph realizations.
The domain randomization strategy is implemented by associating each high-level configuration parameter with a probability model rather than a fixed value (see Figure 3.6). For parameters already defined at the node level through probability distributions (e.g., number of services per host, probability of firewall presence), randomization is lifted one level higher by sampling the parameters of those distributions. In other words, instead of fixing a single distribution across all scenarios, we introduce a hierarchical formulation in which distributional parameters are sampled at the scenario level and subsequently used to sample node-level attribute characteristics.
The randomization process remains constrained within the single-instance generation pipeline described in Section 3.3.2. All sampled configurations must satisfy the same compatibility rules, structural constraints, and statistical alignment mechanisms defined previously, ensuring that the generated scenario family remains consistent with the required characteristics of realistic scenarios.
Furthermore, the availability of explicit parameter distributions enables additional experimental methodologies, including: (i) *systematic stress-testing*, achieved by deliberately adjusting distribution ranges or skewness (e.g., increasing asset concentration or raising the prevalence of firewall rules) to evaluate agent robustness under more challenging conditions; (2) *controlled ablation studies*, performed by narrowing, widening, or fixing selected parameter distributions to isolate the impact of specific structural or security-related factors on agent performance.

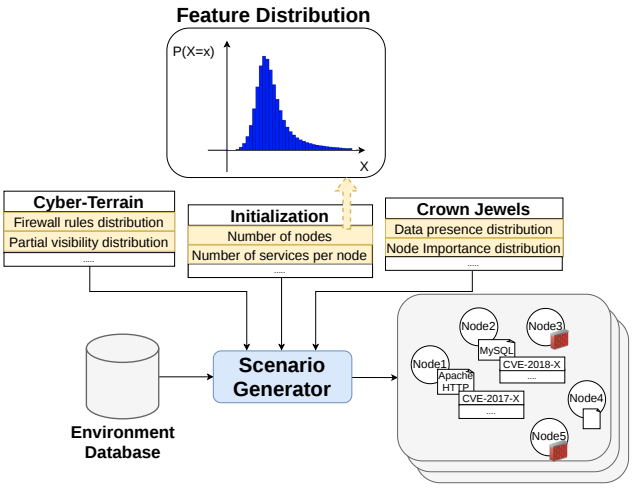


Figure 3.6: Domain randomization strategy supported by proper distributions for each high-level parameter.

3.4 Implementation

The proposed POMDP formulation and automated scenario-generation pipeline are implemented within a simulation framework named `Continuous CyberBattleSim`, released as an open-source project on GitHub. The framework extends the original Microsoft *CyberBattleSim* simulator by providing a more general and configurable environment for critical attack path discovery, while preserving compatibility with the standard `OpenAI Gym` interface.
Gym provides a unified abstraction for RL environments through a minimal API centered around three core functions: `__init__`, `reset()`, and `step()`. This standardization enables compatibility with existing RL libraries and training pipelines, allowing the proposed simulator to be directly integrated with off-the-shelf RL algorithms.

For a given scenario instance, the simulator behavior is defined through the following components.

Initialization (`__init__`). The `__init__` function constructs the complete underlying graph representing the environment according to the parameters or distributions provided by the scenario generator. This includes creating host nodes, assigning features and assets, integrating obstacles, and initializing auxiliary structures required to track the evolution of the episode, such as compromise status and visibility information. The resulting graph represents the complete ground-truth environment state, which remains partially hidden from the agent during interaction.
Extension with respect to Microsoft CyberBattleSim. The proposed implementation extends the original graph representation by supporting additional node-level features, asset types, and obstacles, enabling the simulation of richer attack scenarios with higher customization.

Reset function (`reset()`). The `reset()` function initializes a new episode by sampling from the initial state distribution. This includes selecting the initial node and constructing the initial *graph overlay*, which represents the partial observation available to the agent. The overlay contains only the initially discovered portion of the environment and is progressively updated as new information becomes available. Episode-level variables are reset to their initial conditions.
Extension with respect to Microsoft CyberBattleSim. The proposed simulator introduces randomized initialization of episodes, including the selection of the starting node. This removes the dependency on a fixed initial configuration and enables the generation of diverse trajectories from the same scenario distribution, improving the ability of RL agents to learn transferable policies.

Step function (`step(action)`). The `step()` function implements the POMDP transition dynamics. Given an action, the environment evaluates its validity according to the transition rules defined in Section 3.3.1 and applies the corresponding state transition. The function updates the environment state, including node attributes, compromise conditions, privilege levels, and the agent's graph overlay. It then computes the resulting observation and reward returned to the agent.
Extension with respect to Microsoft CyberBattleSim. The transition mechanism is extended to support a broader range of attack outcomes, including a larger set of MITRE ATT&CK-based behaviors, as well as the additional environment features and obstacles introduced in the proposed formulation. Moreover, the reward mechanism is redesigned as a customizable function, enabling different objectives and optimization criteria to be defined depending on the considered scenario, providing more flexibility than the static reward function incorporated in the original simulator.

Randomization module. A dedicated domain-randomization module is introduced on top of the Gym environment to automatically generate diverse scenario instances. The module reads configuration files describing parameter distributions and samples new environment configurations, allowing multiple independent scenarios to be generated from the same underlying distribution.
Extension with respect to Microsoft CyberBattleSim. While the original CyberBattleSim implementation focuses primarily on predefined individual scenarios, the proposed framework exposes scenario generation as a configurable process. This enables systematic evaluation across multiple

environment instances and supports the study of scalability, robustness, and generalization of RL agents.

3.5 Summary

This chapter formulated attack-path planning as a POMDP and motivated the use of simulation-based cyber-range environments with vulnerability-level action modeling for RL-based discovery of critical attack paths. Building on this foundation, we introduced a data-driven POMDP framework in which transition and reward dynamics are parameterized using expert-annotated vulnerability metadata and network-level features.
The chapter further presented an automated scenario-generation pipeline capable of producing realistic and customizable scenarios while preserving structural and statistical consistency with real-world deployments. By automating vulnerability selection, feature placement, and domain randomization, the framework enables scalable generation of diverse training and evaluation instances while reducing manual modeling effort. A practical Gym-based implementation was also introduced to ensure compatibility with standard RL pipelines.
Overall, this chapter also highlighted the fundamental trade-off between realism and scalability that characterizes cybersecurity training and evaluation platforms, particularly those based on simulation and aimed at automated attack planning. It emphasized the importance of carefully selecting abstractions that preserve the essential characteristics of real-world attack dynamics while remaining computationally tractable for large-scale experimentation through RL.

Link to the thesis objectives

The proposed POMDP framework, supported by the implemented `Continuous CyberBattleSim` platform, serves as the foundation for several subsequent contributions of this thesis. First, it incorporates the ability to model recently disclosed vulnerabilities that are not yet fully annotated, leveraging the automated classification framework introduced in Chapter 4. It is then used in Chapter 6 to generate realistic scenarios and model the sequential decision-making process required for RL agent training and evaluation on the task of critical attack path discovery. Finally, the framework is extended in Chapter 7 to incorporate virtualization dynamics and additional infrastructure-level characteristics, enabling the training and evaluation of mitigation strategies in virtualized environments, with a particular focus on the VMP problem.

Chapter 4

Automated vulnerability taxonomy enrichment

Summary: *This chapter presents the second major contribution of this thesis: a multi-label classification module designed to predict the CWE, CAPEC, and ATT&CK associations of vulnerabilities with meaningful predictive performance. The task is addressed by using vulnerability descriptions as input and an LM as the underlying prediction model. The primary objective is to facilitate the integration of newly disclosed or not yet annotated vulnerabilities into our simulation framework, in which mappings to these taxonomies are essential for realistically modeling vulnerability behavior. Beyond this specific application, the proposed approach contributes to the broader challenge of automating vulnerability annotation, a high-cardinality and labor-intensive task that requires significant expert effort and domain knowledge.*

Contents

4.1 Introduction **65**
4.2 Text-based multi-label classification **67**
4.2.1 Language models 68
4.2.2 Language models for multi-label classification 69
4.3 Existing approaches **69**
4.3.1 Limitations of manual mappings 70
4.3.2 Automated mapping strategies 71
4.4 Methodology **72**
4.4.1 Data collection 73
4.4.2 Hierarchical classification 74
4.5 Experiments **75**
4.5.1 Experimental setup 76
4.5.2 Experimental results 77
4.6 Environment integration **83**
4.7 Summary **84**

4.1 Introduction

A fundamental challenge in designing realistic cybersecurity simulators is the impracticality of explicitly modeling the precise behavior and dynamics of every individual vulnerability. Consequently, simulators focused on attack-path planning must rely on higher-level abstractions that characterize vulnerabilities in terms of the outcomes they enable. This allows state transitions to capture the security-relevant effects of successful exploitation rather than the underlying

technical details of each vulnerability. Such an abstraction is both necessary for scalability and aligned with the primary objective of attack-path planning, where the consequences of exploitation are more relevant than the exploitation mechanisms themselves. In the framework proposed in Chapter 3.3.1, whenever the agent exploits a vulnerability, the transition function updates both the target host and the surrounding network according to the simulated outcome. This raises a first central question: how should exploitation outcomes be represented?
As introduced in Chapter 1.4, MITRE taxonomies such as CWE, CAPEC, and ATT&CK provide complementary abstractions that link vulnerabilities to weaknesses, attack patterns, and adversarial techniques. Among them, ATT&CK is particularly well-suited for modeling exploitation outcomes, as it defines a finite set of attacker tactics and techniques that describe how attackers operate in practice. Rather than modeling the behavior of each vulnerability individually, the simulator can model the ATT&CK techniques enabled by a vulnerability, thereby significantly improving scalability while preserving meaningful representations of attack dynamics [80, 72]. CWE and CAPEC mappings can further refine this representation by incorporating information about underlying weaknesses and known attack patterns.
However, these mappings are largely assigned manually and often suffer from substantial delays and incompleteness. CVSS vectors, which are also leveraged by the simulator, are typically published shortly after vulnerability disclosure, with a median annotation delay of approximately one day [54]. This rapid availability is facilitated by the task's relatively low complexity, as CVSS scoring is a single-label assessment with limited cardinality. In contrast, mappings to CWE, CAPEC, and ATT&CK involve high-cardinality classification problems that often require substantial expert analysis [178, 67, 179]. As a result, these annotations may only become available weeks or months after disclosure, while a significant proportion of vulnerabilities remain permanently unmapped [80, 72]. As a result, a large portion of recently disclosed vulnerabilities cannot be immediately integrated into taxonomy-aware simulation environments. This limitation constrains the applicability of our framework, as the simulation relies on taxonomy-based annotations to represent and model vulnerability effects. Consequently, when CWE, CAPEC, or ATT&CK mappings are unavailable, recently disclosed vulnerabilities cannot be readily incorporated into the simulation. In particular, the framework is intended to support vulnerability prioritization and patch management decisions as new vulnerabilities are disclosed. For instance, newly reported vulnerabilities could be integrated into simulated network twins where they are likely to appear, enabling the system to analyze how frequently they could contribute to critical attack paths under representative threat models. However, to maximize its practical value, such prioritization should be performed as close as possible to the time of disclosure, rather than weeks or months later when taxonomy-based annotations eventually become available. Vulnerabilities that are more likely to play a significant role in high-impact attack paths could then be prioritized for remediation [67, 72].
To enable the integration of vulnerabilities into our framework, this chapter proposes an automated, data-driven approach to infer vulnerability annotations across the ATT&CK, CWE, and CAPEC taxonomies without human intervention, thereby removing the dependence on delayed manual analysis. The approach leverages information available at disclosure time, in particular human-written vulnerability descriptions [72, 92].
More importantly, the proposed model can act as a general automated labeling system for vulnerability databases, providing reliable provisional taxonomy labels for newly disclosed CVEs immediately upon publication, with the potential for future integration into existing vulnerability management and reporting platforms.
Because vulnerabilities are often associated with multiple elements within each taxonomy, this problem naturally takes the form of a multi-label classification task. This task can be addressed

using supervised learning by leveraging historically labeled CVEs and learning patterns between textual vulnerability descriptions and provided taxonomy classes. However, training models to perform classification across the extensive class sets of these taxonomies requires large amounts of labeled data, and no standardized benchmark dataset currently exists for multi-taxonomy mapping [81]. Existing studies are also fragmented, as most focus on a single taxonomy and rely on inconsistent datasets, experimental protocols, and evaluation metrics, which limits meaningful comparison.

Contributions

To enable effective multi-label classification of vulnerability descriptions into the aforementioned taxonomies, this chapter introduces three main contributions:

- We propose a strategy to scrape a continuously updated **dataset** containing approximately 228k CVEs as of submission, spanning the CWE, CAPEC, and ATT&CK taxonomies (680, 442, and 173 classes, respectively). To the best of our knowledge, this is the largest multi-label benchmark to date, substantially surpassing the previously reported largest labeled dataset, with 94k samples focused solely on CWE classification [83].
- We propose a **hierarchical classifier** [180] that decomposes each multi-label classification task into a cascade of taxonomy-aligned stages, improving performance on these high-cardinality problems while maintaining low computational overhead through a *shared-embedding strategy*.
- We present a novel **unified framework** for fine-tuning diverse LMs on CWE, CAPEC, and ATT&CK at all levels of granularity, providing standardized strategies and metrics that address multi-label scenarios and class imbalance.

4.2 Text-based multi-label classification

Multi-label classification is a supervised learning task in which each instance can be associated with multiple labels simultaneously [181]. Unlike single-label classification, where each sample belongs to exactly one class, multi-label settings allow arbitrary combinations of labels to co-occur. This formulation is particularly relevant in domains where objects naturally exhibit multiple properties or categories simultaneously.
Formally, let $\mathcal{X}$ denote the input space and $\mathcal{Y} = \lambda_1, \lambda_2, \ldots, \lambda_C$ the label set containing C possible classes. The objective is to learn a function

$$f : \mathcal{X} \rightarrow 0, 1^C,$$

which maps an input $x \in \mathcal{X}$ to a binary vector $\mathbf{y} = (y_1, y_2, \ldots, y_C)$, where $y_i = 1$ indicates the presence of label λ_i. The resulting vector represents the set of labels associated with the instance.
Multi-label classification exhibits several distinctive characteristics compared to traditional classification tasks. First, the number of possible label combinations grows exponentially with the number of classes, leading to a very large output space. Second, labels are often correlated: the presence of one label may increase or decrease the likelihood of another [182]. Third, datasets frequently follow a long-tail distribution in which a small subset of labels appears frequently while many others occur rarely [183].
Proper evaluation is also critical in multi-label classification, as traditional single-label metrics cannot capture partially correct predictions [184]. Unlike single-label tasks, predictions may

correctly identify only a subset of the true labels, requiring evaluation at both the label and instance levels. Consequently, multi-label classification systems are typically evaluated using complementary metrics that capture prediction accuracy, robustness to class imbalance, exact label-set matching, and label-ranking quality.

4.2.1 Language models

LMs are neural architectures designed to encode natural language into continuous vector representations capturing semantic and contextual information [185]. By mapping text into dense embeddings within a shared semantic space, they enable models to capture relationships between concepts and generalize across downstream tasks [186].
Given an input sequence of tokens $(x_1, x_2, \ldots, x_T)$, an LM produces contextualized representations in which each token embedding is influenced by its surrounding context. Modern LMs are typically based on the transformer architecture [187], whose self-attention mechanism allows the model to capture dependencies between all tokens in a sequence. As a result, textual inputs with similar meanings tend to be mapped to nearby regions of the embedding space, facilitating transfer across tasks such as classification, retrieval, reasoning, or decision-making.
Large-scale pre-training is central to LMs' ability to learn generalizable representations. During this stage, models are exposed to large collections of unlabeled text and typically trained using self-supervised objectives such as masked token prediction or next-token prediction [188]. These objectives encourage the model to learn representations that capture general linguistic patterns and semantic structure. Because the training data spans diverse domains and tasks, the resulting embeddings often encode reusable knowledge that can be adapted to new problems with minimal task-specific supervision.
Architecturally, LMs can be broadly divided into two categories based on how they process and utilize these representations: generative models and discriminative models.

- **Generative models:** Generative LMs focus on producing text sequences through autoregressive token prediction. This category includes decoder-only architectures such as Generative Pretrained Transformer (GPT) [189] and encoder–decoder models such as Text-to-Text Transfer Transformer (T5) [190]. Although primarily designed for text generation, these models learn rich semantic representations during training, enabling them to perform a wide range of downstream tasks that can be formulated as text-generation problems. Furthermore, instruction tuning allows generative models to interpret task descriptions expressed in natural language, enabling flexible task generalization through prompting while relying on shared underlying language representations.
- **Discriminative models:** Discriminative models are primarily designed for text understanding and categorization tasks. They typically rely on encoder architectures, such as Bidirectional Encoder Representations from Transformers (BERT) [191], which are optimized to produce high-quality contextual embeddings of input text that can subsequently be used to distinguish between semantic categories. These models commonly employ bidirectional training objectives, such as masked language modeling, where missing tokens are predicted using both left and right context. This encourages the encoder to learn representations capturing the full semantic meaning of the input sequence, which can then be leveraged for downstream discriminative tasks.

4.2.2 Language models for multi-label classification

Pre-trained LMs have become powerful tools also for multi-label classification tasks involving textual data [192]. Through large-scale pre-training on diverse corpora, given an input text $x \in \mathcal{X}$, an LM can transform the sequence into contextual embeddings using its internal architecture, and this can be either leveraged as is or further adapted to predict multiple relevant labels simultaneously based on proper transformations expressed as additional neural layers or specific operations. Two main strategies are commonly used to apply LMs to multi-label classification:

- **Text-generation approaches:** In this paradigm, the classification task is reformulated as a text generation problem. Instead of directly predicting label probabilities, the model generates the relevant labels as part of an output sequence conditioned on the input text. This approach typically relies on generative architectures such as decoder-only models or encoder–decoder models. While this formulation allows a single model to handle many tasks through prompting or instruction tuning, it introduces additional challenges. The model must generate the correct textual representation of labels, which becomes increasingly difficult as the number of possible classes grows.
- **Fine-tuned encoder-based discriminative models:** Encoder-based models can be adapted for multi-label classification by attaching a classification head to the final embedding representation of the input text. The encoder produces a contextual embedding that summarizes the input's semantic content, and additional classification layers map this embedding to the label space. Typically, a sigmoid activation function is used to produce independent probability estimates for each label, enabling simultaneous prediction of multiple labels. Fine-tuning strategies vary depending on dataset size and domain similarity, ranging from updating only the classification head to jointly training the entire model. This approach provides direct probability estimates for each label and avoids the need to interpret generated text.

Whether classification is performed in the token space through text generation or directly in the label space through discriminative prediction depends on the characteristics and requirements of the target task. Text-generation approaches are particularly appealing because they enable a single model to address a wide range of tasks through prompting or instruction tuning, offering greater flexibility and generalization across domains. In contrast, fine-tuned encoder-based discriminative models operate directly in the class space, producing explicit probability estimates for each label. Although these models are generally more specialized and task-specific, they are often lighter, more efficient, and easier to deploy in practical settings.

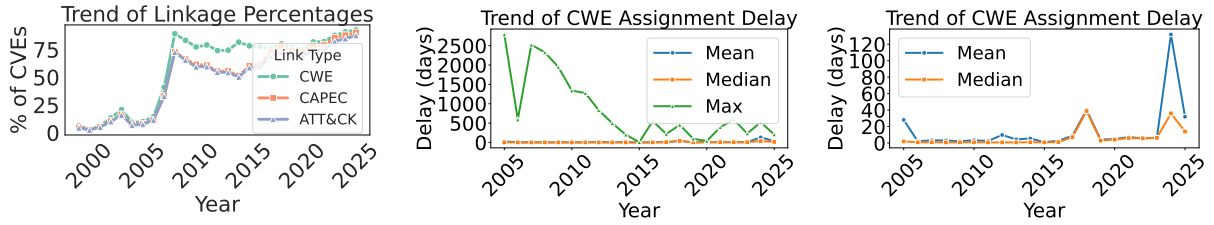
4.3 Existing approaches

This section reviews related work on mapping vulnerabilities to the aforementioned MITRE taxonomies. We first present knowledge bases that rely on manually constructed mappings and symbolic relationships between taxonomies to derive semantic links for vulnerabilities. We then discuss more recent automated approaches, particularly those based on ML and LMs, which infer these mappings directly from vulnerability descriptions. Finally, we highlight the limitations of current methods and the open challenges motivating the contributions of this work.

4.3.1 Limitations of manual mappings

Several manual approaches have been proposed to map vulnerabilities along the pipeline *CVE* → *CWE* → *CAPEC* → *ATT&CK* using predefined knowledge bases manually constructed by security experts [91, 193]. These approaches rely on the existing relationships between taxonomies, linking them through curated mappings to indirectly infer the complete set of taxonomy labels for a vulnerability from labels already assigned in other taxonomies.
These approaches assume the availability of a CWE identifier manually assigned to each CVE. Once the CWE is available, predefined mapping tables are used to link the CWE to CAPEC attack patterns and subsequently CAPEC patterns to ATT&CK techniques. While conceptually straightforward, this pipeline suffers from several limitations:

1. **Manual CWE annotation:** A CVE must first be manually analyzed and assigned a CWE by security experts. This step is both time-consuming and prone to delays from the moment of disclosure. Furthermore, many CVEs remain unlabeled, such as those marked *NVD-CWE-noinfo* in the NVD, indicating that sometimes no CWE can be assigned due to insufficient information. This lack of labeling blocks downstream mapping [193], even when relevant attack patterns and techniques could potentially be associated with these vulnerabilities based on their textual descriptions. To ensure comprehensive coverage and prevent such bottlenecks, it is therefore essential to also explore direct mappings to CAPEC attack patterns and ATT&CK techniques, bypassing intermediate dependencies when necessary.
2. **Maintenance of knowledge bases:** The mappings between CWE → CAPEC and CAPEC → ATT&CK must be carefully curated and updated as these taxonomies continuously evolve. Each taxonomy regularly introduces new entries, modifies hierarchical relationships, or deprecates outdated elements. For instance, the ATT&CK matrices are updated approximately every six months; the latest update (ATT&CK v19, April 2026) introduced 9 new *Enterprise* techniques and 5 new *ICS* techniques, as well as 199 major sub-technique changes. Maintaining consistency across interconnected taxonomies therefore requires continuous manual effort and expert oversight, which may not scale efficiently as the number of vulnerabilities grows and taxonomy classes change.
3. **Incomplete coverage tables:** Even with continuous updates, many CWE entries remain unmapped to CAPEC attack patterns, and many CAPEC patterns similarly lack corresponding ATT&CK mappings. This results in gaps in the mapping pipeline [193] that hinder the classification process when relying solely on knowledge bases. In practice, this means that even if a vulnerability has a CWE assigned, it may still fail to propagate through the pipeline because no predefined mapping exists between the corresponding taxonomy elements, even when the vulnerability exhibits clear attack patterns and techniques.



(a) % of CVEs manually linked to taxonomies.

(b) CWE assignment delay trend (max, mean, median).

(c) CWE assignment delay trend (mean and median).

Figure 4.1: Overview of CVE linkage and CWE assignment delay trends across years.

To illustrate the current state of knowledge bases, we refer to the widely used *CVE2CAPEC* [194] project. From 1999 to 2025, the percentage of CVEs manually linked to at least one CWE, CAPEC, or ATT&CK element increased from 6%, 5%, and 4% to 92%, 91%, and 89%, respectively, reflecting sustained efforts to improve manual mappings (Figure 4.1a). Despite this progress, CWE assignment remains a major bottleneck, as CAPEC and ATT&CK mappings are typically derived from manually assigned CWE labels. Consequently, delays in CWE annotation propagate to the entire mapping process.
Statistics derived from the NVD by measuring the delay between vulnerability publication and the transition to the *Analysis* phase, during which a CWE is assigned, show an average delay of 24 days, a median delay of 11 days, and a maximum delay of 170 days in 2025. Figure 4.1b reports the complete distribution, including maximum values, to highlight the difference between average, median, and extreme cases, while Figure 4.1c excludes outliers to provide a clearer view of the typical assignment delays. The average and median CWE assignment delays shown in Figure 4.1c do not exhibit a consistent decreasing trend over time. Although the maximum delay has decreased compared with earlier years, recent vulnerability disclosures can still remain uncategorized for several months, with some cases requiring more than five months before a CWE assignment is provided.
These results highlight the persistent lag in fully integrating CVEs into higher-level taxonomies when relying on manual CWE assignment as a prerequisite for mapping to CAPEC and ATT&CK. As the number of reported vulnerabilities continues to grow each year, manual workflows increasingly struggle to keep pace with disclosure rates. Consequently, there is a clear need for approaches that can automate parts of this pipeline.

4.3.2 Automated mapping strategies

This section reviews existing ML-based approaches targeting the automated mapping of vulnerability descriptions to CWE, CAPEC, and ATT&CK taxonomies.
A primary limitation of existing automated vulnerability classification studies is their narrow focus on a single taxonomy. Most approaches address CWE, CAPEC, or ATT&CK in isolation, rather than proposing a unified framework that leverages the shared challenges across these taxonomies through a single methodology.
Table 4.1 categorizes existing studies according to the target classification taxonomy (CWE, CAPEC, or ATT&CK) and provides additional details on the approaches evaluated. These works correspond to the CVE $\rightarrow$ CWE, CVE $\rightarrow$ CAPEC, and CVE $\rightarrow$ ATT&CK mappings presented in Table 1.1, while providing a more detailed analysis of their methodological and experimental settings. In particular, the table reports the dataset size ($\|D\|$), the granularity level considered within each taxonomy (e.g., tactics, techniques, or sub-techniques in ATT&CK), and the number of classes ($\|C\|$) included at that level.
This distinction is crucial, as the coverage of the target taxonomy can significantly influence the difficulty and applicability of the proposed approaches. In practice, many datasets do not contain sufficient samples to provide at least one labeled instance for every possible class, particularly for rare or highly specialized classes that are infrequently associated with publicly disclosed vulnerabilities. Consequently, several studies restrict their evaluation to subsets of the taxonomy, which may simplify the classification task but limit the extent to which the proposed methods can support comprehensive vulnerability knowledge enrichment.
The table also reports the number of traditional ML methods (e.g., decision trees) or LMs evaluated in each study. Early works predominantly relied on classical ML techniques applied to textual features extracted from vulnerability descriptions. More recent approaches increasingly

Table 4.1: Comparison of existing automated mapping approaches across key evaluation and training criteria: MS (multiple splits), SS (stratified splits), MM (multi-label metrics), CI (class imbalance handling), and H (hyperparameter optimization/exploration). A dash ("–") denotes missing information.

Taxonomy	Study	$\|D\|$	Level	$\|C\|$	ML	LMs	Standard ML Practices				
							MS	SS	MM	CI	H
CWE	[79]	82k	Full	124	1	2	✓	✓			✓
	[80]	29k	Base CWE	19	6	0	✓			✓	✓
	[81]	4k	Full	25	1	4		✓	✓		
	[82]	50k	Full	364	1	0				✓	
	[83]	94k	Full	9	3	0				✓	
CAPEC	[87]	<1k	-	-	0	2			✓		
ATT&CK	[89]	8k	Techniques	50	4	0	✓	✓	✓	✓	✓
	[3]	8k	Techniques	52	7	0	✓		✓		
	[90]	62k	Techniques	37	4	0			✓		✓
	[91]	25k	Tactics	10	10	4	✓				
	[92]	2k	Techniques	185	0	4			✓	✓	
	[93]	11k	Techniques	-	0	16					
CWE CAPEC ATT&CK	**Ours**	228k	Full Full Subtechniques	680 442 173	0	12	✓	✓	✓	✓	✓

leverage widely used LMs to further automate the feature engineering step. However, many studies omit standard ML practices that are important for reliable evaluation in this domain. These include the use of multiple splits (MS) or stratified splits (SS), multi-label evaluation metrics (MM), strategies to address severe class imbalance (CI), and systematic hyperparameter optimization or exploration (H). This fragmentation, combined with the absence of large-scale benchmark datasets that simultaneously cover multiple taxonomies, results in significant variability across published results.
In contrast, our study integrates all major taxonomies into a unified framework, uses the largest available dataset and class set to date, performs classification at the most granular levels based on current disclosures, and follows best practices for multi-label imbalanced learning and reproducible evaluation.

4.4 Methodology

This section presents the methodology adopted in this study, beginning with the data collection pipeline that transforms historically labeled vulnerabilities and existing knowledge bases into a large-scale dataset suitable for training and evaluating ML models for automated taxonomy mapping. We then introduce a hierarchical classification architecture that decomposes the problem into sequential subproblems aligned with the hierarchical structure of CWE, CAPEC,

[0]By reconstructing live mappings, the dataset dynamically updates class sets as underlying taxonomies evolve and newly labeled CVEs as they become available.

and ATT&CK. By leveraging the DAG structure of these taxonomies instead of performing flat multi-label classification, the approach enables multi-level reasoning while reducing prediction complexity.

4.4.1 Data collection

As discussed in Section 4.3.1, traditional mapping strategies depend heavily on manual CWE annotations before downstream mappings can be derived. This reliance introduces delays and limits the scalability of manual pipelines. To overcome this limitation, we construct a supervised dataset (Zenodo repository) capturing historical mappings between vulnerabilities and the target taxonomies, enabling ML models to learn these relationships directly from prior assignments. The methodology comprises three main stages: data acquisition, label construction and propagation, and dataset normalization.

1. **Data collection and label propagation.** CVE entries are collected from the NVD, from which we extract vulnerability descriptions and available CWE annotations. Only CVEs containing CWE labels are retained for dataset construction. Additional CAPEC and ATT&CK labels are then derived using existing mapping relationships, notably through the `CVE2CAPEC` knowledge base.
   The resulting dataset contains tuples of the form (CVE description, CWE, CAPEC, ATT&CK), where each CVE may contain one or multiple taxonomy mappings.
2. **Hierarchical labeling.** The MITRE taxonomies are represented as DAGs where nodes correspond to categories and edges encode hierarchical relationships, as described in Section 1.4. Labels are propagated along parent-child relations such that if a CVE is associated with a node, all its ancestor nodes are also assigned to the instance. This ensures hierarchical consistency and enables learning across multiple levels of abstraction.
   The hierarchical graph structure is further preserved during training and inference, allowing the model to leverage parent-child dependencies and perform classification at varying levels of granularity.
3. **Cross-matrix adaptation.** As discussed in Section 1.4.3, ATT&CK defines separate matrices for Enterprise, ICS, and Mobile environments, each containing environment-specific techniques and sub-techniques. To support classification across these domains, we extend Enterprise ATT&CK labels to their corresponding ICS and Mobile equivalents whenever mappings are available.
   This translation relies on relationships provided by the `CVE2CAPEC` project, enabling the generation of label sets spanning all three ATT&CK matrices. To the best of our knowledge, this is the first dataset to explicitly support classification across Enterprise, ICS, and Mobile ATT&CK environments.

All stages of the pipeline support continuous data acquisition from the NVD and automatic reconstruction of taxonomy mappings as new information becomes available. Consequently, the dataset can evolve dynamically with the cybersecurity landscape, incorporating newly disclosed vulnerabilities and updates to existing taxonomies. This design ensures that the generated training data remains aligned with the latest versions of the underlying vulnerability and threat intelligence frameworks.

4.4.2 Hierarchical classification

The CWE, CAPEC, and ATT&CK taxonomies define large and continuously evolving class spaces. For instance, ATT&CK contains hundreds of techniques and sub-techniques, and, as mentioned in Section 4.3.1, MITRE updates it every 6 months. Designing models capable of operating over such large and evolving class spaces presents two key challenges: (1) efficiently handling large-scale multi-label classification; (2) adapting efficiently to frequent updates in the taxonomy structure.

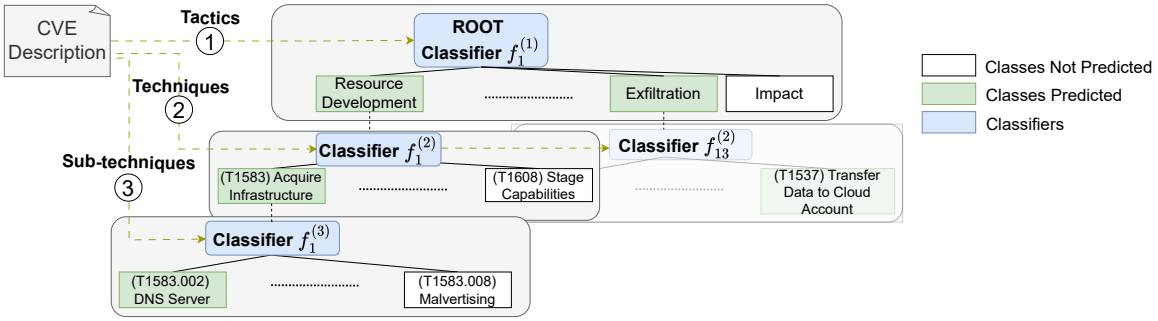


Figure 4.2: Representation of hierarchical classification with ATT&CK as an example.

To address these challenges, we leverage the inherent hierarchical structure of cybersecurity taxonomies and adopt a *divide-and-conquer* strategy. Instead of training a single flat classifier over the entire class space, we decompose the prediction process into a sequence of hierarchical decisions that follow the underlying DAG structure of each taxonomy. Figure 4.2 illustrates this approach using the ATT&CK taxonomy as an example; however, the same hierarchical decomposition strategy is applied to other taxonomies considered in this work, including CWE and CAPEC.

Let the taxonomy be represented as a directed acyclic graph with L levels (for example, $L = 3$ for ATT&CK: Tactics $\rightarrow$ Techniques $\rightarrow$ Sub-techniques). The classification process iteratively involves nodes that predict labels at each level of the hierarchy. Predictions at each level determine which child classifiers are activated at the next level.

More formally, let x denote the CVE description encoding. At level 1, the root classifier performs multi-label prediction: $\hat{y}^{(1)} = f_1^{(1)}(x)$, where $\hat{y}^{(1)}$ represents the predicted labels among the root-level classes. At subsequent levels, each node j at level i is associated with a specialized classifier $f_j^{(i)}$ which predicts labels among the subset of classes reachable from that node: $\mathcal{C}_j^{(i)} \subset \mathcal{C}^{(i)}$. Inference therefore proceeds in a top-down manner along the DAG edges:

$$\begin{aligned} \hat{y}^{(1)} &= f_1^{(1)}(x) \quad \text{(root classifier)} \\ \hat{y}_j^{(2)} &= f_j^{(2)}(x) \quad \text{for all } j \text{ reachable from } \hat{y}^{(1)} \\ &\vdots \\ \hat{y}_k^{(L)} &= f_k^{(L)}(x) \quad \text{for all } k \text{ reachable from } \hat{y}^{(L-1)} \end{aligned}$$

This hierarchical inference mechanism ensures that only classifiers corresponding to relevant branches of the taxonomy are executed. For example, if a vulnerability is predicted to belong to the ATT&CK tactics `Persistence` and `Privilege Escalation`, only the technique-level classifiers associated with those tactics are activated. This avoids evaluating the full set of ATT&CK techniques and instead restricts the prediction process to the relevant subsets.

Advantages. The proposed hierarchical classification strategy offers several benefits:

+ **Inference-time pruning:** Entire branches of the taxonomy can be skipped if parent nodes are not activated. This allows the model to significantly reduce unnecessary computations.
+ **Multi-granular classification:** The architecture naturally produces predictions at multiple levels of abstraction. This enables the system to output high-level or fine-grained labels depending on the inference requirement.
+ **Flexible level prediction:** Predictions may terminate at intermediate levels if no confident child labels are detected. For example, a vulnerability may be associated with a Base CWE without identifying a more specific variant.
+ **Better performance on imbalanced data:** Taxonomy datasets are highly imbalanced, with many classes containing only a small number of samples. By decomposing the classification task into smaller subproblems, each classifier operates on a reduced, more balanced subset of classes, thereby improving learning stability [180].
+ **Modularity and adaptability:** When taxonomies evolve, for example through the introduction of new techniques or sub-techniques, only the classifiers associated with the affected branches must be retrained. This modular structure substantially reduces adaptation costs compared to flat classifiers, as each sub-model operates on a smaller label space and therefore requires fewer training samples, less computation, and shorter retraining times.

Challenges. Despite its advantages, hierarchical classification introduces several challenges:

- **Increased inference overhead:** Multiple classifiers may need to be executed sequentially for each input. This can increase inference time compared to flat classification models.
  Mitigation: To reduce this overhead, we propose leveraging the hierarchy of classifiers by sharing a common encoder-only LM backbone. The encoder produces a shared embedding for each CVE description, which is computed only once. Lightweight classification heads are then applied at each hierarchical level, allowing the system to reuse the same embedding and minimize redundant computations.
- **Error propagation:** Incorrect predictions at higher levels can propagate to lower levels of the hierarchy. For example, if a tactic is misclassified, the subsequent technique predictions will necessarily be constrained to an incorrect branch.
  Mitigation: One strategy to reduce this effect is to prioritize training accuracy at higher levels of the hierarchy, particularly for the root classifiers. These models may be trained with additional data, more epochs, or specialized strategies to ensure reliable high-level predictions.

4.5 Experiments

This section presents the experimental evaluation of the proposed approach. We first describe the experimental setup, including the dataset construction and the evaluation metrics used to assess model performance. We then report the results obtained across the different classification taxonomies using multiple LMs. The experiments continue to evaluate the effectiveness of the proposed hierarchical classification strategy compared to standard flat approaches, while analyzing performance across multiple evaluation settings.

4.5.1 Experimental setup

This section describes the dataset used for training and evaluation as well as the metrics adopted to measure performance across the different classification tasks.

Dataset. The dataset used in the experiments was collected as described in Section 4.4.1 and was last updated on September 19, 2025. It contains a total of 228,479 CVE entries. This number is lower than the total number of CVEs published to date because many historical CVEs lack an associated CWE identifier and were therefore excluded during preprocessing.
Each taxonomy covers a slightly different subset of CVEs because the manual mappings between CVEs and the various taxonomies are incomplete, as discussed in Section 4.3.1. As a result, the number of samples available for each classification task varies with the availability of corresponding labels.
The following statistics summarize the datasets used for the three classification tasks:

- **CWE:** 680 labels; 228,479 samples (avg. 3.33 labels/sample)
- **CAPEC:** 442 labels; 210,001 samples (avg. 32.57 labels/sample)
- **ATT&CK:** 173 labels; 201,562 samples (avg. 12.83 labels/sample)

For the ATT&CK taxonomy, the Enterprise matrix is used as the reference taxonomy. This matrix was selected due to its widespread adoption, comprehensive coverage of adversarial techniques, and the relatively large number of labeled CVEs available for training and evaluation.
Among the three classification tasks, CWE classification constitutes the primary task. Mapping vulnerabilities to CWE categories, in fact, enables optional further linkage to CAPEC attack patterns and ATT&CK techniques through existing cross-taxonomy mappings. However, as discussed in Section 4.3.1, these cross-taxonomy mappings are incomplete, and many vulnerabilities lack explicit links between these frameworks. Consequently, jointly training models across the three taxonomies allows the learning process to capture implicit relationships between them, potentially compensating for missing manual mappings.

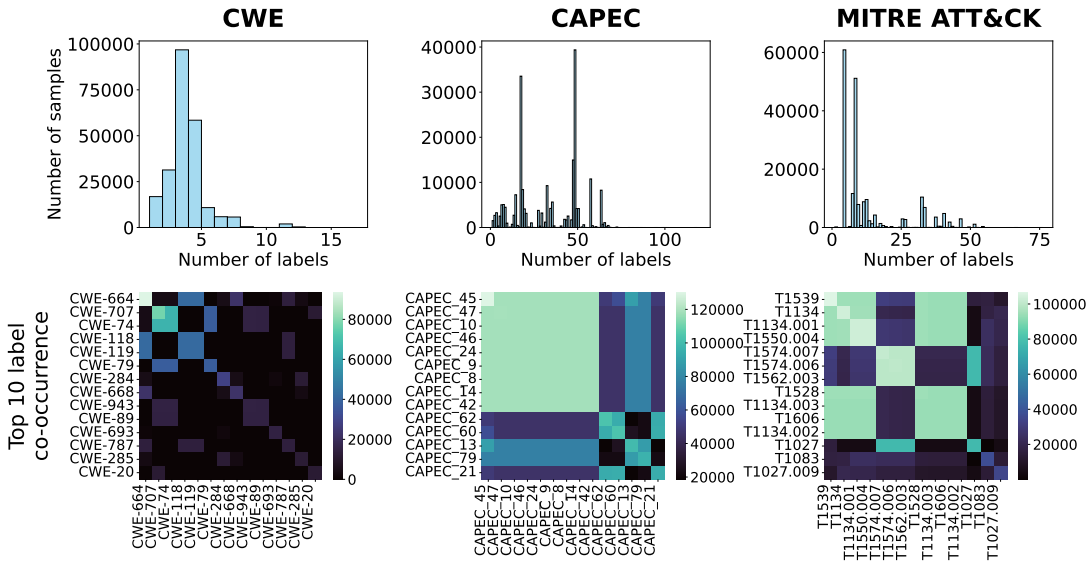


Figure 4.3: Distribution of labels and label co-occurrence in the dataset across the taxonomies.

Figure 4.3 illustrates the distribution of labels across the three taxonomies as well as the most

frequent label co-occurrence patterns. The results highlight important differences between the classification tasks. In particular, the CAPEC and ATT&CK datasets exhibit substantially more labels per sample and more frequent label combinations than the CWE dataset. This reflects the fact that vulnerabilities can often be associated with many more attack patterns or adversarial techniques, making these tasks significantly more complex from a multi-label learning perspective.

Evaluation metrics. To evaluate the performance of the models on the multi-label classification tasks, we adopted a set of evaluation metrics recommended in the multi-label learning literature [184]. The selected metrics were chosen to capture complementary aspects of model performance, including prediction accuracy, ranking quality, and robustness to class imbalance. Additionally, the final subset of metrics was selected based on correlation analyses between performance measures [195], ensuring that the evaluation set remains informative while avoiding redundant indicators:

- *Micro-averaged F-measure* (Micro-F1): Measures overall prediction quality by aggregating true positives, false positives, and false negatives across all labels before computing the F1-score. This metric gives greater weight to frequent labels and provides a robust estimate of overall classification performance.
- *Weighted F1-Score* (Weighted-F1): Computes the F1-score independently for each label and then averages the scores weighted by label frequency. This metric provides a more balanced evaluation when the dataset is class-imbalanced, giving greater weight to labels that appear more frequently in the dataset.
- *Subset Accuracy* (Subset-Acc): Measures the proportion of instances for which the predicted set of labels exactly matches the ground-truth set. This metric is particularly strict in multi-label settings because even a single missing or incorrect label results in an error. As a result, it highlights the model's ability to recover complete label sets rather than partially correct predictions.
- *Mean Reciprocal Rank* (MRR): Evaluates the ranking quality of predicted labels by measuring the position of the first correct label in the ranked list of predictions.

Together, these metrics provide a comprehensive evaluation of multi-label classification performance, capturing both the average and strict correctness of predicted label sets, as well as the quality of label ranking.

4.5.2 Experimental results

This section presents the experimental results. We first compare text-generation and encoder-based language models for multi-label classification across the considered taxonomies. Next, we evaluate the impact of hierarchical decomposition by comparing hierarchical classifiers with their flat counterparts. Finally, we analyze the effect of temporal dataset splits on classification performance and model generalization.

Text-generation large language models

Instruction-tuned Large LMs (LLMs) excel at a wide range of generative and reasoning tasks, raising the question of whether they can be used *zero-shot* [196] for multi-label classification of CVE descriptions. In this setting, *zero-shot* refers to using a model without additional task-specific fine-tuning or training examples, relying only on the instructions provided in the prompt to

infer the appropriate labels. Zero-shot inference is the only practical approach in our setting, as few-shot methods are infeasible: providing correctly labeled examples for a sufficient number of class combinations would require a large number of samples, far exceeding the context window of current models, while also introducing substantial computational overhead and distorting any comparison with encoder-based fine-tuning. In fact, the taxonomies contain hundreds of classes (for instance, the CWE taxonomy in our dataset has more than 680 entries), and a meaningful prompt would require not only the identifiers but also their full human-readable names to avoid ambiguity.

Setup. We benchmark four state-of-the-art text-generation LLMs—GPT-4.1, GPT-5.1, LLaMA-3.1-8b-instant [197], and DeepSeek-V3.1 [198]— prompting them to output CVE labels given the CVE description. We conducted our experiments on a representative subset of the dataset by selecting the 1,000 oldest CVEs, whose disclosure information has been publicly available the longest. This choice represents the most relevant and easiest evaluation scenario, since the models have likely seen these entries during pretraining. The following prompt template is used, substituting the CVE identifier, description, and taxonomy name (CWEs, CAPECs, ATT&CK subtechniques).

Prompt Template

```
You are a classification assistant in the cybersecurity domain.  Only return valid
lists of IDs as requested (e.g., [‘‘LABEL1’’, ‘‘LABEL2’’, ...]).  Do NOT include any
explanations, commentary, or extra text.

CVE ID: ID

Description:  DESCRIPTION

Task:  Perform a multi-label classification to TAXONOMY for this CVE.
```

We also vary the temperature parameter over 0, 0.5, 1.0 for each CVE and taxonomy to study the model's behavior from deterministic to more creative output [199], averaging results across all three settings.

Table 4.2: Zero-shot classification performance of text-generation LLMs across three cybersecurity taxonomy prediction tasks evaluated on the oldest 1,000 CVEs. Reported results represent the average performance across temperature settings of 0, 0.5, and 1.0.

Model	**CWE**				**CAPEC**				**MITRE ATT&CK**			
	Micro-F1	Weighted-F1	Subset-Acc	MRR	Micro-F1	Weighted-F1	Subset-Acc	MRR	Micro-F1	Weighted-F1	Subset-Acc	MRR
GPT-4.1	**0.29**	**0.22**	**0.00**	**0.52**	0.03	0.02	0.00	0.38	**0.01**	**0.01**	**0.02**	**0.11**
GPT-5.1	0.27	0.21	**0.00**	**0.52**	0.04	0.03	0.00	**0.45**	**0.01**	**0.01**	**0.02**	0.10
DeepSeek-V3.1	0.28	0.21	**0.00**	0.49	**0.07**	**0.06**	**0.01**	0.44	**0.01**	**0.01**	0.00	0.09
LLaMA-3.1	0.19	0.13	**0.00**	0.31	0.02	0.01	0.00	0.18	**0.01**	0.00	0.01	0.05

Analysis. Results in Table 4.2 highlight that zero-shot LLM performance remains far from operationally usable across all taxonomies. The comparatively best results are obtained on the CWE taxonomy, likely because CVE-to-CWE mappings are widely available in public sources such as NVD entries, making these associations more likely to appear during pretraining. Nevertheless, even under these favorable conditions, performance remains limited, while predictions for CAPEC and ATT&CK sub-techniques are extremely poor, with near-zero scores.
Another major challenge is the high sensitivity of these models to the temperature hyperparameter. Predictions can vary substantially across different temperature settings, leading to inconsistent outputs for the same input. Consequently, temperature is a critical parameter that must be carefully tuned, as it significantly affects both prediction stability and classification performance. This instability was quantified by measuring agreement across varying temperatures, computed as the average Jaccard overlap between predictions, highlighting the greater consistency of GPT models compared with the others: GPT-4.1 (0.68), GPT-5.1 (0.64), DeepSeek-V3.1 (0.45), and LLAMA-3.1 (0.22).
To summarize, even under favorable conditions, general-purpose text-generation LLMs cannot reliably perform multi-label CVE classification in a zero-shot setting.

Fine-Tuned encoder-only language models

Encoder-only LMs are the standard choice in the literature for multi-label text classification. The typical approach consists of attaching a classification head to a pretrained encoder and fine-tuning the model to learn a direct mapping from the embedding space to the label space. This section investigates whether existing pretrained encoder-only LMs, both general-purpose and cybersecurity-specific, can be effectively fine-tuned in a *flat* setting for the multi-label prediction tasks under consideration.

Setup. We used a 5-fold nested stratified cross-validation (CV) with 80% (train) and 20% (test) splits, conducting an initial study that ignores CVE temporal order and focuses on class distributions. Due to strong label imbalance, we adopt an adaptive label-preserving stratified algorithm [200] to ensure every label occurs at least once in the training data. Results are averaged across the five held-out test folds.
We benchmark the two widely used encoder-only baselines, BERT and ELECTRA [201], as well as variants such as DistilBERT [202] and RoBERTa [203]. We also evaluate four cybersecurity-oriented models: SecureBERT [204], SecRoBERTa [204], SecBERT [205], and CySecBERT [206]. Notably, these cybersecurity-specific models are essentially BERT-family architectures that have already been fine-tuned once on various cybersecurity corpora to produce more domain-adapted embeddings. The fine-tuned layers consist of the classification head and the last encoder layer, which we found empirically to be the minimum required for competitive performance, as fine-tuning additional layers yielded negligible improvement. The models were thoroughly hyperparameter-optimized, including the selection of class-imbalance-aware loss functions. Detailed information on both the optimized and manually configured hyperparameters is provided in Appendix A.1.

Analysis. Results in Table 4.3 show that encoder-only models can be fine-tuned effectively for these large multi-label classification tasks, achieving strong performance across all taxonomies. Although a performance ranking is visible, the gains between models are generally modest. Cybersecurity-specific variants do not consistently outperform general-purpose LMs—for example, RoBERTa surpasses several of these domain-adapted models. This suggests that, given

Table 4.3: Comparison between ELECTRA and BERT models (and variants) using mean test scores obtained from 5-fold CV across the three classification tasks.

Model	CWE				CAPEC				MITRE ATT&CK			
	Micro-F1	Weighted-F1	Subset-Acc	MRR	Micro-F1	Weighted-F1	Subset-Acc	MRR	Micro-F1	Weighted-F1	Subset-Acc	MRR
ELECTRA	0.76	0.71	0.51	0.86	0.83	0.81	0.49	0.89	0.79	0.78	0.65	0.94
BERT	0.76	0.70	0.48	0.86	0.83	0.81	0.45	0.89	0.80	0.78	0.63	0.94
DistilBERT	0.78	0.74	0.53	0.87	0.84	0.82	0.51	0.89	0.80	0.79	0.66	0.94
RoBERTa	0.80	0.76	0.56	0.88	0.84	0.83	0.52	0.91	0.81	0.80	0.67	**0.95**
SecRoBERTa	0.77	0.73	0.52	0.86	0.83	0.81	0.49	0.90	0.79	0.78	0.64	0.94
SecBERT	0.78	0.74	0.54	0.87	0.84	0.82	0.52	0.90	0.80	0.79	0.66	0.94
CySecBERT	0.80	0.77	0.57	0.88	0.85	**0.84**	0.52	**0.92**	**0.82**	**0.81**	**0.68**	**0.95**
SecureBERT	**0.81**	**0.78**	**0.58**	**0.89**	**0.86**	**0.84**	**0.55**	**0.92**	**0.82**	**0.81**	**0.68**	**0.95**

a sufficiently large task-specific dataset, a single fine-tuning largely eliminates any advantage gained from the prior cybersecurity-oriented pretraining. Consequently, the additional computational cost of creating such domain-specific variants appears hard to justify for tasks where large amounts of labeled data are available. To summarize, with a sufficiently large dataset, fine-tuned encoder-only LMs perform similarly on our tasks, regardless of whether they are general-purpose models or have undergone domain-specific pretraining.

Flat vs. hierarchical classification

In this section, we present the empirical results of the proposed hierarchical classification strategy, combined with the strongest baseline identified earlier—fine-tuned encoder-only LMs. The approach exploits the inherent structure of the taxonomies by decomposing the prediction task into smaller sub-problems handled by a hierarchy of aligned models.

Setup. To evaluate hierarchical classification, we compare SecureBERT (the best-performing model found in the previous section) under flat and hierarchical setups, using a maximum hierarchy depth of $L = 3$ across all taxonomies. Deeper levels are flattened, yielding a hybrid flat–hierarchical design. At each level, we fine-tune a separate instance of SecureBERT, all sharing the same pretrained backbone but trained on different label spaces.
The experimental protocol from the previous section is maintained, with additional strategies introduced to handle hierarchy-specific challenges:
(1) *Varying dataset sizes:* Each hierarchical node classifier is trained only on samples containing its corresponding classes, resulting in datasets of different sizes. To ensure balanced training, the number of epochs is scaled inversely with dataset size relative to the root model (which sees all data), allowing smaller datasets proportionally more training epochs;
(2) *Shared children:* When a class appears under multiple parents, we assign it to a single parent to enforce a disjoint partition of the class set. This ensures that each class is handled by exactly one model and that no two classifiers can predict the same class. As a result, the original DAG

structure is transformed into a single-parent hierarchical representation, eliminating prediction conflicts and overlaps.
(3) *Sparse or small datasets:* Nodes with small datasets (fewer than 50 samples) or containing only one child class are merged with their parent to avoid undertrained or irrelevant models.

Table 4.4: Comparison between hierarchical and flat SecureBERT classifiers using mean test scores obtained from 5-fold CV across the three classification tasks.

Model	CWE					CAPEC					MITRE ATT&CK				
	Micro-F1	Weighted-F1	Subset-Acc	MRR	# Models	Micro-F1	Weighted-F1	Subset-Acc	MRR	# Models	Micro-F1	Weighted-F1	Subset-Acc	MRR	# Models
Flat	0.81	0.78	0.58	0.89	1	0.86	0.84	0.55	0.92	1	0.82	0.81	0.68	0.95	1
$L = 2$	0.88	0.87	0.67	0.96	11	0.92	0.91	0.54	0.97	10	0.91	0.90	0.71	1.00	11
$L = 3$	0.90	0.90	0.71	0.97	47	0.94	0.92	0.07	1.00	28	0.93	0.93	0.72	1.00	34

Analysis. The improvements resulting from the hierarchical decomposition reach 12% for CWE, 8% for CAPEC, and 12% for ATT&CK, using weighted-F1 score as a reference metric for $L = 3$ (as shown in Table 4.4). We also report the number of models trained for each taxonomy, illustrating that while increasing L enables finer-grained classification, it also increases computational training/inference costs (i.e., more models). An interesting pattern emerges for CAPEC, the dataset with the most labels per sample: at $L = 3$, subset accuracy drops sharply while weighted-F1 improves, showing that the model performs better on average but struggles with exact matches—a single missing or extra label can cause the mismatch. This also illustrates how hierarchical depth can shift the balance between strict and average performance metrics.

Inference time analysis. The main drawback of the hierarchical solution is the need to invoke multiple LMs for each CVE classification—one for each activated node from root to leaf—which naturally increases inference time. One potential use case for these LMs is as real-time assistants during vulnerability reporting, dynamically suggesting likely classes as analysts draft vulnerability descriptions. In such or similar settings, inference efficiency becomes critical, raising a natural question: *Do the performance gains justify the additional computational cost?*

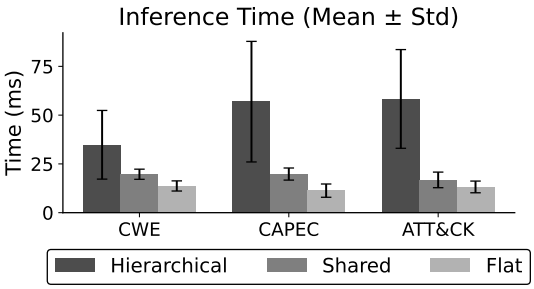


Figure 4.4: Inference time comparison among hierarchical, flat, and shared-embedding hierarchical methods.

The shared-embedding strategy described in Section 4.4.2 can mitigate this issue by computing a single transformer embedding once—since the same LM is used across the hierarchy—and applying only the task-specific components at each level (in our setup, the final encoder layer and classification head). To assess the impact of this design, we benchmark mean inference time

on 10k CVEs using: (i) a flat SecureBERT model, (ii) a hierarchical SecureBERT with $L = 3$, and (iii) its shared-embedding variant, running all experiments on a reference machine[1].
Figure 4.4 shows that the shared-embedding variant significantly mitigates the hierarchical overhead, recovering approximately 72% (CWE), 81% (CAPEC), and 92% (ATT&CK) of the performance gap relative to the flat baseline, thereby eliminating most of the additional inference cost introduced by the hierarchical formulation.

Temporal generalization study

Traditional ML evaluation methods, such as standard K-Fold CV or stratified splits, ensure robustness but ignore the temporal nature of CVE disclosures, thereby limiting their relevance for real-world forecasting. This setup addresses the key question: *can a model trained on vulnerabilities disclosed up to year X generalize to CVEs disclosed after X?* To answer this question, we adopt a temporal evaluation strategy in which the training set is progressively expanded over time. At each timestep X, the model is trained exclusively on CVEs disclosed before and during year X, and its performance is evaluated on future vulnerabilities disclosed after X. For example, a model trained on data from 1999 is evaluated on CVEs from 2000 onward, while a model trained on data from 1999–2000 is evaluated on CVEs disclosed from 2001 onward.
The resulting evaluation provides a measure of temporal generalization, assessing whether models trained on historical vulnerabilities can maintain performance on future CVEs. In Figure 4.5, the x-axis represents the training cutoff year X, while each point reports the performance obtained when training on CVEs disclosed up to X and testing on vulnerabilities disclosed afterwards. Results across multiple evaluation metrics illustrate how model generalization evolves as additional historical data becomes available and reveal potential distribution shifts caused by changes in vulnerability descriptions, terminology, or label sets or distributions.

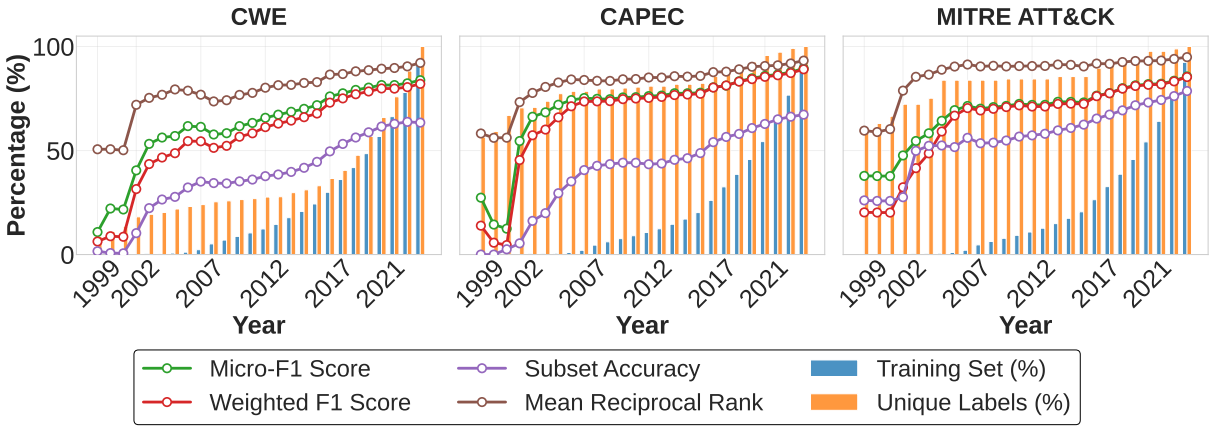


Figure 4.5: Training set coverage, unique label coverage, and performance trends under recursive temporal splitting using the flat SecureBERT classifier.

Setup. We employ a temporal K-Fold CV strategy [207] in which models are iteratively retrained on all CVEs disclosed up to year X and evaluated on CVEs disclosed from $X + 1$ onward. Note that the model may be inherently unable to predict the right classes for all samples, as classes introduced after year X fall outside its class space if no samples existed before that year.

[1]x86_64 AMD Ryzen 9 5900X CPU, 125 GB RAM, NVIDIA GeForce RTX 3090 Ti 24 GB, Ubuntu 22.04.1

To ensure a fair comparison, the number of training epochs is scaled inversely with the current size of the training set, following a procedure similar to the hierarchical experiment, so that earlier years with fewer samples are trained proportionally longer on the smaller dataset.
We also report: (i) the % of CVEs included in the training set relative to the full dataset, indicating how much of the total labeled data available today was available at each year; (ii) the % of classes observed until each year relative to now, reflecting when certain labels first appeared; and (iii) the change in each metric resulting from the addition of new data.

Analysis. Figure 4.5 reports recursive temporal results revealing key insights:

- **Label coverage.** CAPEC and ATT&CK already cover most labels early on (∼60% of those available now by 1999), while CWE labels are expanding mainly in more recent years, reflecting its evolving taxonomy coverage: CVEs with many new weaknesses are appearing more recently. Assuming no new labels emerge, models may not require further adaptation if they continue to perform well on future data. However, as soon as a new label appears, the classifier must be fine-tuned to accommodate it in the output layer. As new labels continue to appear, this is very important, especially for CWE. Hierarchical models offer flexibility, requiring fine-tuning only for the relevant sub-classifier rather than retraining a large, flat classifier over the entire label space.
- **Dataset size.** Although data growth is more concentrated in recent years, the largest performance gains occur in earlier years, suggesting that data quality benefits more than data quantity. Notably, a particularly large performance gain is observed in 2002, with absolute improvements of 23% (CWE), 41% (CAPEC), and 12% (ATT&CK) in the weighted F1 score. These improvements remain significant despite the dataset representing less than 1% of the data available today.
- **Performance.** On average, model performance improves steadily over time across all metrics. Nonetheless, we observe occasional dips in which adding a single year of training data slightly degrades some performance indicators in the future. Taking the weighted-F1 score as a reference, the average yearly improvement is 3.0% (CWE), 3.0% (CAPEC), and 2.6% (ATT&CK).

In summary, for these classification tasks, data availability is no longer the primary bottleneck, as most gains were achieved early. The field needs new methodologies (e.g., our hierarchical approach) to efficiently incorporate emerging classes without full model fine-tuning.

4.6 Environment integration

As discussed in Section 4.1, the proposed classification models can be integrated into the simulation framework to support modeling of newly disclosed vulnerabilities that lack expert-curated taxonomy annotations, allowing the simulator to approximate their outcomes and, as a consequence, enable prioritization strategies under the proposed methodology of this thesis.
During environment initialization, a preliminary check can be performed on each CVE entry to determine whether existing taxonomy mappings are already available. For example, if a CWE label is present and `CVE2CAPEC` provides the corresponding CAPEC and ATT&CK links, these mappings can be directly used by the simulator. Otherwise, a pre-trained LM within the proposed framework can be used to infer missing taxonomy labels. In practice, one of the best-performing models identified in this study—such as the hierarchical SecureBERT classifier with $L = 3$—can be used for this purpose.

As described in Section 3.3.1, the state transition function must determine whether a particular exploitation outcome is feasible for a given vulnerability and model its effects through appropriate state transitions and rewards. The taxonomy labels predicted by the proposed fine-tuned LMs can be leveraged for this purpose before expert-curated mappings become available, enabling preliminary analyses of newly disclosed vulnerabilities. For instance, predicted ATT&CK techniques can be used to infer likely exploitation outcomes within the simulation. Figure 4.6 showcases the simulation pipeline executed at each agent timestep when a selected vulnerability does not have expert-provided labels and an outcome classifier is integrated into the simulator. In this case, the classifier estimates the possible attack outcomes associated with the vulnerability. The figure illustrates this process using MITRE ATT&CK Tactics as the outcome space, where the predicted outcomes determine the subsequent observation and reward received by the agent. This mechanism enables the simulation of scenarios involving newly disclosed or previously unlabeled vulnerabilities, for which manual security annotations are unavailable. To further improve the granularity and realism of the modeled behavior, predicted CWE and CAPEC labels can also be incorporated. These provide complementary information about the underlying software weaknesses and attack patterns that may be leveraged against them, enabling a richer representation of vulnerability behavior. The *C-CyberBattleSim* simulator implemented in this work (see Section 3.4) currently supports ATT&CK labels at the tactic level as outcome representations, covering 10 out of the 14 tactics defined by the framework. However, the simulator was designed as an extensible framework, allowing the future integration of additional tactics, techniques, and other taxonomy-based representations to support more fine-grained vulnerability behavior modeling.

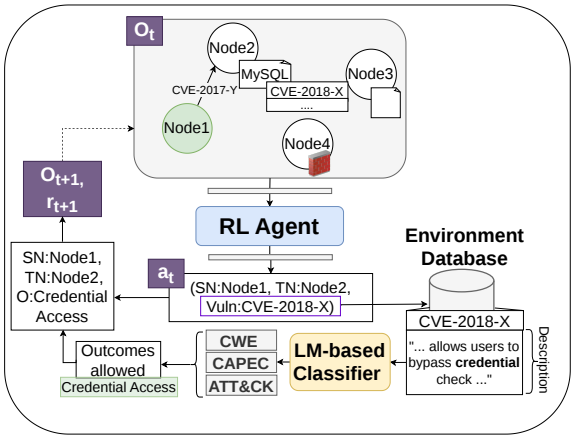


Figure 4.6: Integration of the LM-based classifier as outcome approximator in the simulation environment.

Finally, the attacker profile employed by the reward function can be adapted to account for these inferred characteristics, introducing rewards and penalties that better reflect the objectives, capabilities, and preferences of different attacker types.

4.7 Summary

This chapter presented a unified solution to the problem of multi-label vulnerability classification across the three most widely used taxonomies (CWE, CAPEC, and ATT&CK) at their most granular levels with an extensive set of classes (680, 442, and 173, respectively). The approach leverages the largest up-to-date dataset currently available for this task (228k samples), constructed through the automated data collection pipeline introduced in this work.

Unlike prior studies, which typically focus on a single taxonomy, the proposed framework standardizes classification across all three taxonomies and introduces a consistent experimental methodology. This includes the use of multiple stratified dataset splits, evaluation metrics tailored to multi-label and class-imbalanced settings, and systematic hyperparameter optimization

across several encoder-only LMs.
The results demonstrate that exploiting the hierarchical structure of these taxonomies significantly improves classification performance. In particular, the hierarchical approach yields weighted-F1 improvements of approximately 12% for CWE, 8% for CAPEC, and 12% for ATT&CK classification. While hierarchical classification introduces additional inference overhead, this drawback is mitigated through the proposed shared-embedding strategy, which allows multiple hierarchy levels to reuse common model representations, recovering up to 92% of the speed of the corresponding flat model.
However, these results should be interpreted in the context of the labels used for training and evaluation. In particular, the high classification scores primarily demonstrate the models' ability to reproduce the reconstructed mappings present in the dataset, rather than necessarily establishing the semantic correctness of those mappings. This distinction is particularly relevant for mappings such as CVE2CAPEC, where existing mappings may contain false positives or other labeling errors, as highlighted by previous studies [208]. Nevertheless, the results show that the models are able to learn consistent mapping patterns from the available data, suggesting that improvements in label quality could provide further performance gains.
Additional experiments with temporal dataset splits show that the models generalize to newly disclosed vulnerabilities, highlighting their potential applicability in real-world scenarios.

Link to the thesis objectives

The proposed LM-based multi-label classifiers can automatically map CVEs, based on their textual descriptions, to the CWE, CAPEC, and ATT&CK taxonomies. Since these annotations are often delayed or incomplete when performed manually, the proposed approach provides approximate mappings with meaningful performance that can be leveraged by the POMDP framework developed in Chapter 3, which relies on this metadata to realistically model vulnerability behavior and attack outcomes.

Part III

Reinforcement learning for graph combinatorial optimization

Chapter 5

Scalable and generalizable RL agents for graph combinatorial optimization

Summary: *This chapter presents the third and primary contribution of this thesis: a general methodology for improving the scalability and generalization capabilities of RL solutions in discovering heuristics for GCO problems. The focus of this chapter is on the learning contribution itself, independently of any particular application domain or task. Consequently, the various benchmarks considered throughout the chapter—including the attack path prioritization benchmark introduced in this thesis—serve primarily as evaluation frameworks to assess the effectiveness of the proposed methodology and to demonstrate its ability to improve performance, scalability, and generalization across diverse problem settings.*

Contents

5.1 Introduction . . . 89
5.2 Graph representation learning . . . 91
5.2.1 Unsupervised auto-encoders . . . 92
5.2.2 Graph neural networks . . . 92
5.3 Problem formulation . . . 94
5.4 Methodology . . . 94
5.4.1 Node representation learning . . . 94
5.4.2 Continuous spaces . . . 95
5.4.3 Projection agent . . . 96
5.4.4 RL-GCO automated framework . . . 97
5.5 Existing RL-GCO approaches . . . 98
5.6 Experiments . . . 99
5.6.1 Generalization study . . . 102
5.6.2 Inference time scalability . . . 103
5.7 Summary . . . 105

5.1 Introduction

The problem of identifying critical vulnerability paths in graph-structured network environments, introduced in the previous chapters, can be formulated as a GCO problem, as discussed in Section 2.4. More generally, the efficient solution of GCO problems has attracted significant attention in recent years, as a wide range of NP-hard optimization problems across diverse

application domains can be naturally represented as graphs and formulated within the GCO framework [209]. Consequently, advances in GCO methodologies have the potential to benefit not only the cybersecurity use case considered in this thesis but also numerous optimization problems across fields.

In the seminal work of [210], which introduced 21 NP-complete problems, nearly half correspond to versions of graph optimization problems, including the Traveling Salesman Problem (TSP) [211]. Most of these problems can also be formulated on graphs [135]. However, due to the combinatorial nature of these problems, where the number of possible solutions grows rapidly with graph size, exact methods become computationally intractable [130].

As a result, heuristic methods are widely employed, trading optimality for computational efficiency [19]. However, traditional heuristics typically rely on domain expertise, handcrafted rules, and iterative trial and error [131], often requiring substantial human effort to redesign and adapt them to each problem variation. More recently, ML approaches have emerged as a promising alternative for data-driven heuristic discovery [132, 135, 212]. In particular, as explained in Section 2.7, RL methods [213] have attracted increasing attention, as they do not rely on labeled samples—which are costly to obtain for large instances [214, 136]—and construct solutions autoregressively through sequential decision-making, mirroring traditional GCO solvers [209]. RL has been used both for direct heuristic approximation and to accelerate exact solvers, e.g., by guiding branch-and-bound in constraint programming [213]. Therefore, improving RL for GCO (RL-GCO) is important to advance both directions.

A key advancement in RL-GCO has been the adoption of GNNs [215] to learn transferable, fixed-dimensional representations of graph-structured data. GNNs have been widely employed to efficiently encode graph-based observation spaces [216], enabling RL agents to operate on graphs of varying sizes while maintaining a fixed-dimensional state representation. Moreover, GNNs naturally produce representations that are invariant to node ordering, a desirable property for graph-based learning and generalization.

However, standard RL-based approaches still rely on *discrete* action spaces, where actions are tightly coupled to specific graph instances in both semantics and dimensionality, effectively reducing them to non-transferable identifiers [217, 218]. Using discrete actions also reduces scalability as the output dimensionality of function approximators grows with the size of the action space [219]. To address these limitations, more recent approaches extend the use of GNNs to the action space by introducing action embeddings, typically constructed from the embeddings of graph components. For instance, in the critical attack path prediction problem formulated as a POMDP in Section 3.3.1, actions correspond to host nodes and include internal dimensions representing their associated vulnerabilities. These methods then evaluate each embedded action using a Q-function $Q(o, a)$, iterating over the full set of candidate actions together with the observation vector o, as exemplified by approaches such as `S2V-DQN` [220]. However, this comes at a high computational cost, as the number of evaluations increases linearly with the number of actions, making runtime dependent on graph size and leading to scalability challenges [20].

In particular, this computational cost is exacerbated in real-world GCO tasks with structured, high-dimensional decision variables, where the number of actions can grow super-linearly with graph size. A prominent example is our benchmark application, where decisions involving combinations of nodes and their associated vulnerabilities lead to a quadratic explosion of the action space. Similar challenges arise in other applications, such as traffic engineering, where actions correspond to routing paths, resulting in even more severe combinatorial growth [221]. As a result, evaluating a DNN multiple times, proportional to the action space size, quickly becomes computationally prohibitive. This issue is amplified when the application

context involves dynamic settings, such as GCO applications in communication networks or data center infrastructures, where graph representations evolve continuously, and decisions must be recomputed repeatedly under strict latency constraints [19].

Contributions

To improve the applicability of RL-based heuristics to GCO problems, particularly with respect to scalability, efficiency, and generalization across large and diverse instances, we propose a unified methodology that enhances the RL-GCO framework in an application-independent manner. The proposed approach is designed to be broadly applicable across a wide range of tasks, rather than being tailored to a specific domain or benchmark.
Our main contributions are summarized as follows:

- **Projection agents for scalable and generalizable RL-GCO.** We introduce *projection agents*, which operate directly within a continuous GNN-based action-embedding space. Inspired by the Wolpertinger architecture [219], the DNN predicts a desired action embedding in a single forward pass, which is mapped to a valid discrete action via Nearest Neighbor (NN) search. This yields up to 16.2× faster inference than iterative methods and improves relative generalization by up to 40
- **Unlocking RL-GCO for problems with complex and super-linear decision variables.** We introduce a framework for constructing structured action embeddings via (i) semantic representations of individual decision components and (ii) composite embeddings capturing their interactions. This supports realistic super-linear decision spaces involving multiple interdependent variables, as found in modern real-world GCO tasks (e.g., traffic engineering, where selections involve paths and edges).
- **Unsupervised representation learning framework to better compare RL-GCO.** We use unsupervised GNN-based representations [222] to learn node embeddings, which are then used to build structured, joint observation–action spaces with semantic alignment. By decoupling representation learning from policy learning, we enable fair comparison across RL methods within a shared embedding space.
- **Unified Python library**. We provide `LaGCO-RL`, a modular library for automated **L**atent **a**ction-space construction in **GCO** with **RL**. The library enables rapid adaptation to new tasks by requiring only environment-specific logic, while promoting reproducibility of the proposed methodology [223]. The implementation natively supports the proposed projection agent alongside existing RL baselines and is built on standardized Stable-Baselines3 (SB3) [224] implementations.

This chapter presents a general contribution that is independent of both the simulation environment introduced in Chapter 3 and the critical attack path discovery problem. The latter is considered only as a benchmark task for the experimental evaluation, while the proposed methodology is applicable to a broader class of GCO problems.

5.2 Graph representation learning

Representation learning aims to automatically learn meaningful and compact representations of data that can be efficiently exploited by ML models [225]. Instead of relying on manually engineered features, representation learning methods extract latent features directly from raw

data while preserving the most relevant structural and semantic information [226].
More formally, representation learning seeks to learn a parameterized mapping function h_θ that projects raw input data $x \in \mathcal{X}$ into a fixed-dimensional latent embedding space $\mathbb{R}^d$.

$$\begin{aligned} h_\theta : \mathcal{X} \to \mathbb{R}^d, \qquad z_i = h_\theta(x_i), \\ \|z_i - z_j\| \ll \|z_i - z_k\|, \quad \forall (x_i, x_j) \sim \mathcal{S}, \quad (x_i, x_k) \not\sim \mathcal{S} \end{aligned} \tag{5.1}$$

The representation is built such that the learned embeddings $z_i \in \mathbb{R}^d$: (i) preserves the semantic and structural properties of the original sample x_i, (ii) maintains similarity relationships between related samples, and (iii) provides a compact and fixed-dimensional encoding suitable for downstream learning tasks. This objective can be generally expressed as in Equation 5.1, where $\mathcal{S}$ denotes a semantic similarity relation between samples. Therefore, semantically similar entities are mapped to nearby points in the embedding space, while dissimilar entities remain farther apart.
One major advantage of fixed-dimensional embeddings arises in domains such as graph-structured data [227], where the size and complexity of the input may vary significantly across samples. Learning representations independent of the original graph size enables downstream ML tasks to operate in a consistent space. This property is particularly important for many ML algorithms based on feed-forward DNNs, which generally require inputs and outputs to be of fixed dimensionality.
Furthermore, the semantic organization of the latent space facilitates downstream learning by encoding structural and relational patterns directly within the learned representations. As a result, the complexity associated with extracting meaningful information from raw data is partially addressed during the representation learning phase, allowing downstream models to focus more efficiently on task-specific objectives [228].

5.2.1 Unsupervised auto-encoders

Autoencoders are among the most widely used approaches for representation learning [229]. An autoencoder is composed of two functions: an encoder h_θ that projects the input data into a latent representation, and a decoder g_ϕ that reconstructs the original input from this latent space. Given an input sample x, the latent embedding z and the reconstructed sample $\hat{x}$ are defined as:

$$z = h_\theta(x), \qquad \hat{x} = g_\phi(z)$$

The objective is to learn compact latent representations that preserve the most informative characteristics of the original data by minimizing a reconstruction loss $\mathcal{L}(x, \hat{x})$. The decoder primarily serves as a training mechanism that guides the encoder to learn informative latent embeddings via the reconstruction objective. Once training is completed, the decoder is typically discarded, while the encoder and its learned representations are retained for downstream tasks. The encoder and decoder architectures can vary depending on the data modality. For instance, convolutional neural network autoencoders are commonly used for images [230], recurrent architectures for sequential data [231], and GNN-based autoencoders for graph-structured data. Consequently, the representation learning framework remains generic and flexible, where the choice of encoder primarily depends on the structural properties of the input data.

5.2.2 Graph neural networks

GNNs are deep learning architectures specifically designed to process graph-structured data. Unlike traditional DNNs, which typically assume Euclidean data structures such as vectors or

grids, GNNs operate directly on graphs composed of nodes and edges, allowing them to capture both feature information and relational dependencies between entities.
A graph is generally represented as $G = (V, E)$, where V denotes the set of nodes and E denotes the set of edges connecting them. Nodes and edges may both contain associated features describing their properties or interactions. The objective of a GNN is to learn meaningful representations at different levels of abstraction, including node-level, edge-level, or graph-level embeddings.
Most GNN architectures rely on a message-passing framework [232], in which node representations are iteratively refined by aggregating information from neighboring nodes and propagating information along edges. At each layer, a node updates its embedding by combining its current representation with information received from its local neighborhood. Formally, the representation of a node v at layer $k+1$ can be expressed as in Equation 5.2.

$$h_v^{(k+1)} = \phi^{(k)} \left(h_v^{(k)}, \bigoplus_{u \in \mathcal{N}(v)} \psi^{(k)} \left(h_v^{(k)}, h_u^{(k)}, e_{uv} \right) \right) \tag{5.2}$$

In this formulation, $h_v^{(k)}$ denotes the representation of node v at layer k, $\mathcal{N}(v)$ represents the neighborhood of node v, and e_{uv} denotes the features associated with the edge connecting nodes u and v. The message function $\psi^{(k)}$ computes the information exchanged between neighboring nodes, potentially incorporating both node and edge features. The aggregation operator $\bigoplus$ combines incoming messages from neighboring nodes using permutation-invariant operations such as summation, averaging, or max-pooling. Finally, the update function $\phi^{(k)}$ integrates the aggregated neighborhood information with the current node representation to produce the updated embedding.
Through successive message-passing iterations, node embeddings progressively capture larger receptive fields and richer structural information from the graph. After several layers, the learned representations encode both local neighborhood patterns and higher-order graph dependencies, eventually converging toward stable latent representations suitable for downstream tasks.
Once node representations are obtained, embeddings for higher-level graph components can also be derived through proper pooling operations [233]. For example, edge embeddings h_{uv} can be constructed from the representations of their incident nodes and edge attributes (Equation 5.3), while a graph-level embedding h_G can be obtained by aggregating all node embeddings (Equation 5.4). $\mathrm{READOUT}(\cdot)$ denotes a permutation-invariant pooling operation over the set of embeddings, such as sum, mean, max-pooling, or attention-based aggregation.

$$h_{uv} = \mathrm{READOUT}(h_u, h_v), \tag{5.3}$$

$$h_G = \mathrm{READOUT}\left(\{ h_v \mid v \in V \} \right), \tag{5.4}$$

Several GNN variants have been proposed in the literature, primarily differing in the message, aggregation, and update functions used to combine neighborhood information and update node embeddings at each layer. Graph Convolutional Networks [234] extend convolution operations to graph domains by aggregating normalized neighborhood information. Graph Attention Networks [235] introduce attention mechanisms to dynamically weigh the importance of neighboring nodes during aggregation. GraphSAGE [236] focuses on inductive learning by sampling and aggregating neighborhood features, enabling improved scalability to large graphs.

5.3 Problem formulation

We formulate GCO as a sequential learning problem aimed at constructing a solution on a graph $G = (V, E, V_f, E_f)$, where V and E denote the sets of nodes and edges, $V_f \in \mathbb{R}^{|V| \times d_v}$ encodes node features, and $E_f \in \mathbb{R}^{|E| \times d_e}$ encodes edge features (with $|V|$ and $|E|$ representing the cardinality of the node and edge sets, respectively, and d_v and d_e their corresponding feature dimensions). Let $\mathcal{C}(G)$ denote the decision space of admissible graph substructures (e.g., node subsets, edge subsets, paths, or induced subgraphs). This formulation generalizes beyond single-node decisions, enabling the selection of arbitrary substructures and thus yielding a richer and more challenging action space. In the autoregressive setting, a solution is constructed sequentially as $\mathbf{S} = (S_1, \ldots, S_T)$, where each decision $S_t \in \mathcal{C}(G)$ may depend on previous selections. The full sequence $\mathbf{S}$ defines a feasible solution to the GCO problem with the objective function in Equation 5.5.

$$\min_{\mathbf{S} \in \mathcal{C}(G)^T} F(G, \mathbf{S}) \quad \text{s.t.} \quad H(G, \mathbf{S}) \leq 0, \tag{5.5}$$

F measures the global cost and H encodes optional feasibility constraints. This formulation captures realistic GCO settings in which both F and H may depend on node and edge features, as well as higher-order structural interactions. This autoregressive formulation naturally induces an MDP with a discrete action space as below:

- **Observation.** Each observation $o_t \in \mathcal{O}$ represents the intermediate solution at step t, defined as $o_t = (G, S_1, \ldots, S_{t-1})$.
- **Action.** An action $a_t \in \mathcal{A}(o_t) \subseteq \mathcal{C}(G)$ selects a valid graph substructure that extends the current partial solution.
- **Transition.** The transition deterministically appends the selected substructure to the solution while enforcing constraints H; the episode terminates when a solution is formed, or a cutoff is reached.
- **Reward.** The reward is derived from $F(G, \mathbf{S})$, so maximizing return is equivalent to minimizing the combinatorial objective.

For example, in TSP, the observation is a city graph with binary indicators of the current partial tour. The action selects an unvisited city, tour updates are triggered, and the reward favors shorter tours.

5.4 Methodology

This section introduces our core contributions: the framework for unsupervised embedding learning, the construction of latent spaces, and the design of projection agents. We conclude with an overview of the supporting `LaGCO-RL` library created to support the contributions.

5.4.1 Node representation learning

We pre-train a GNN encoder for representation learning to produce node-level embeddings that serve as shared blocks for building both observations and actions of a given RL task. This design (i) decouples representation learning from action selection, enabling fair comparison across RL-GCO methods through shared embeddings; (ii) improves sample efficiency by restricting the policy to observation–action mapping via a lightweight controller [237]; and (iii) stabilizes learning by ensuring that action points remain stationary during training so the agent can reliably preserve learned mappings without representational drift.

We learn node-level representations using a Graph Auto-Encoder (GAE) [238] strategy. A GNN encoder maps each node based on its features and neighborhood to a p-dimensional embedding, forming a matrix $\mathbf{Z} = h_\theta(G) \in \mathbb{R}^{|V|\times p}$. A feed-forward decoder g_ϕ reconstructs graph information from $\mathbf{Z}$. The decoder is multi-headed, with separate heads for node features $x_i \in \mathbb{R}^{d_v}$, edge features $e_{ij} \in \mathbb{R}^{d_e}$ when present, and the adjacency matrix $A \in \mathbb{R}^{|V|\times|V|}$. The encoder–decoder pair is trained jointly (Figure 5.1) by minimizing a weighted reconstruction loss over components (Equation 5.6).

$$\mathcal{L}(\theta,\phi) = \alpha \frac{1}{|V|}\sum_{i\in V}\mathcal{L}_{\text{node}}(\hat{\mathbf{x}}_i,\mathbf{x}_i) + \beta \frac{1}{|E|}\sum_{(i,j)\in E}\mathcal{L}_{\text{edge}}(\hat{\mathbf{e}}_{ij},\mathbf{e}_{ij}) + \gamma\,\mathcal{L}_{\text{adj}}(\hat{A},A), \tag{5.6}$$

In this formulation $\hat{\mathbf{x}}_i$, $\hat{\mathbf{e}}_{ij}$, and $\hat{A}$ are reconstructions produced by $g_\phi(\mathbf{Z})$, and α,β,γ control the contribution of each component.
All terms are normalized by the number of reconstructed elements in order to balance gradient magnitudes across heterogeneous components. Additionally, node and edge features are decomposed by type (binary, categorical, continuous) to enable appropriate normalization and dedicated decoder heads with suitable loss functions (e.g., mean squared error for continuous features).
We pretrain the GAE on trajectories obtained by sampling random valid actions, allowing the graph structure to evolve during learning (e.g., tour trajectories in TSP). This corresponds to maximal valid exploration while remaining independent of any learned policy. After convergence, the decoder is discarded, and the encoder is retained to produce meaningful embeddings.

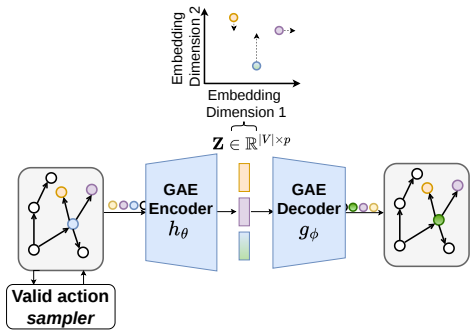


Figure 5.1: Joint GNN encoder-decoder framework leveraging valid action-sampled snapshots for node representation learning.

5.4.2 Continuous spaces

This subsection describes the solution used to derive proper observation and action spaces from the node embedding set $\mathbf{Z}$ derived in Section 5.4.1.

Observation space. We represent each observation as a graph G_t describing the current solution state. A fixed-dimensional observation vector o_t is obtained by applying K permutation-invariant graph aggregation operators $\mathcal{P}_k$ (e.g., sum, mean, or handcrafted graph statistics) over the node embeddings $\mathbf{Z}_t \in \mathbb{R}^{|V|\times p}$, and concatenating the resulting representations:

$$o_t = \Phi_{\text{obs}}(G_t) = \bigoplus_{k=1}^{K}\mathcal{P}_k(\mathbf{Z}_t),$$

where Φ_{obs} denotes the observation mapping function and $\bigoplus$ the concatenation operator. This yields a fixed-dimensional, order-invariant graph embedding that summarizes the agent's observation.

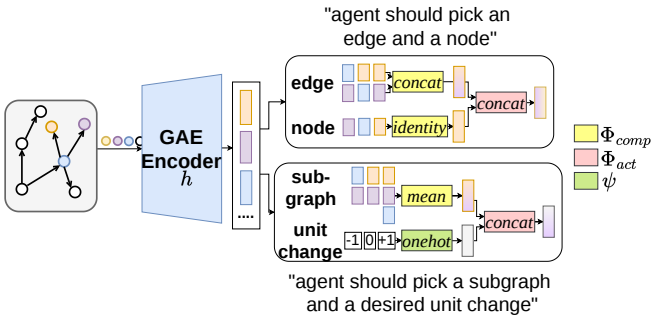


Figure 5.2: Overview of action space construction from node embeddings, illustrating two application examples and possible representation operator choices.

Action space. Actions may target graph components at varying levels of granularity, including nodes, edges, paths, subgraphs, or combinations thereof (e.g., edge pairs). An action a is represented as a set of *components* $\mathcal{C}(a) = c_1, \ldots, c_m$, where each component corresponds to a subset of nodes $\mathcal{V}(c_i) \subseteq V$. A component embedding is obtained by aggregating the embeddings of its nodes with a pooling Φ_{comp}:

$$h(c_i) = \Phi_{\text{comp}}(z_j \mid j \in \mathcal{V}(c_i)).$$

Depending on the application, this operator may be permutation-invariant (e.g., for subgraphs) or permutation-variant (e.g., edges where $(i, j) \neq (j, i)$). The action embedding is then obtained by combining embeddings of its components through an aggregation operator Φ_{act} (Figure 5.2):

$$\mathbf{u}(a) = \Phi_{\text{act}}(h(c_1), \ldots, h(c_m)), \quad c_i \in \mathcal{C}(a),$$

which again can be permutation-invariant depending on whether component ordering is relevant. In many applications, actions involve not only graph components but also additional non-graph attributes α_i, such as discrete choices or features associated with graph elements. For example, in the attack-path prioritization benchmark, vulnerabilities are associated with network nodes and described through textual information. Depending on the problem formulation, these attributes may be selected independently or may be conditioned on previously selected graph components, resulting in hierarchical decision dependencies.
These are encoded via a task-specific function $\psi(\cdot)$ and combined as:

$$\mathbf{u}(a) = \Phi_{\text{act}}\big(h(c_1), \ldots, h(c_m), \psi(\alpha_1 \mid c_1, \ldots, c_m), \ldots, \psi(\alpha_k \mid c_1, \ldots, \alpha_{k-1})\big).$$

This unified formulation enables heterogeneous actions typical of real-world applications by embedding both graph components and custom attributes in a shared space.

5.4.3 Projection agent

The learned action embeddings enable the proposed projection agent by defining a bounded continuous latent action space over all possible action embeddings, onto which the agent projects its decisions. The latent action space $\mathbf{U} = \{\mathbf{u}(a_1), \ldots, \mathbf{u}(a_n)\}$ is integrated into the agent through a three-step pipeline: (1) optional preprocessing, such as normalization or dimensionality reduction, while preserving the relative geometric relationships between action embeddings; (2) empirical estimation of the latent space bounds across graph instances (minimum and maximum values per dimension); and (3) alignment of the resulting dimensionality and bounds with the agent's `gym.Box` continuous action space (see RL agent in Figure 5.3).
Given an observation o_t, the policy π outputs a continuous proto-action in this space $\tilde{a}_t = \pi(o_t) \in \mathbf{U}$, where $\tilde{a}_t$ represents a "desired" target point in latent space rather than a valid discrete action. The term *proto-action* is used because this output is not itself executable: it is an intermediate representation of the agent's decision before being converted into a valid action available in the environment. This proto-action is then decoded by retrieving its k nearest embeddings, forming

a candidate set $\mathcal{A}_k(o_t)$. A final single action is selected from this set using a proper decoding strategy: e.g., simple NN (k=1), stochastic or rule-based selection, or a learned scoring function over the k-NN candidates. This separation between decision generation and action execution allows the RL agent to operate in a smooth continuous space while still interacting with the discrete action space required by the environment.
This formulation results in a region-based learning signal in $\mathbf{U}$, where the decoding induces a space partition, and all points within a region map to the same discrete action. Consequently, the projection function is piecewise constant, and learning updates optimize the geometry of these regions rather than isolated discrete actions. This enables generalization across similar actions and reduces the difficulty of exploring large combinatorial action spaces.

5.4.4 RL-GCO automated framework

This subsection presents our `LaGCO-RL` library, which implements the proposed automated environment construction framework and unifies representation learning and latent action construction (Sections 5.4.1-5.4.3). Users define a standard Gym environment encoding the task logic and extend it through a class inheriting from both `ContinuousEnv` (Fig. 5.3) and a task-specific base class. The framework minimizes task-specific engineering by requiring users to provide only a small set of high-level definitions through the inherited interface. These definitions are then automatically processed by the `gym.Wrapper` module, which integrates the representation learning and latent continuous-space construction methodology described in the previous sections.

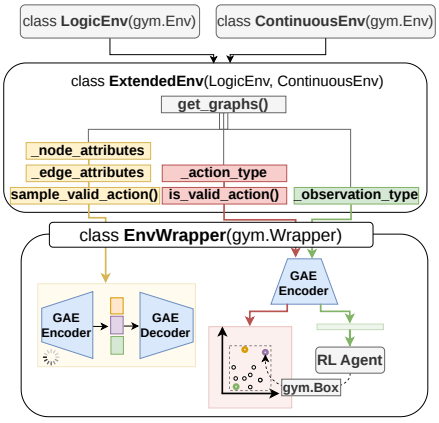


Figure 5.3: User-defined logic, attributes, and methods provided for GAE pre-training and RL integration.

The environment description is provided through abstract interfaces defining the graph structure, attributes, observations, and actions. The graph representation is specified through **get_graphs()**, while node and edge information is provided through **_node_attributes** and **_edge_attributes**. These definitions allow the framework to automatically construct the components required for graph representation learning, including decoder heads and unsupervised learning objectives, supported by the generation of valid training samples for training the GAE through **sample_valid_action()**.
Observation construction is handled through **_observation_type**, which enables the use of generic invariant aggregation functions or task-specific observation mappings. Similarly, the action space is defined through **_action_type**, describing the components that compose an action. The framework then automatically generates the complete action space through Cartesian products and can optionally remove infeasible combinations using **is_valid_action(action)**, supporting both static and time-varying action spaces.
Finally, the generated action representation is exposed through standard Gym interfaces. For the proposed projection agent, the continuous latent action space is mapped to a `gym.Box` representation, while other RL strategies can be supported through compatible wrappers. This abstraction enables the reuse of the same environment construction methodology across different graph-based RL problems with minimal task-specific modifications.
In practice, the attack path benchmark POMDP described in Chapter 3 represents one possible

underlying task environment. The proposed framework can be applied on top of this environment by providing the required methods and attributes, which enable the automated construction of the representations and continuous action spaces described in the previous sections. Similarly, other environments with different task logic can adopt the same methodology by implementing the corresponding interface definitions, without requiring changes to the underlying framework.

5.5 Existing RL-GCO approaches

This section discusses the related work of RL-GCO, particularly the two main axes illustrated in Figure 5.4: (i) the action space complexity of existing benchmark GCO problems, and (ii) the ability of current RL-based solutions to effectively handle the typical action space challenges of GCO.

Action space complexity gap. Prior work on RL-GCO has largely focused on simplified benchmarks with limited decision complexity. Seminal studies [220, 239, 131] consider classical problems such as the *TSP* and Minimum Vertex Cover (*MinVertex*) [240] (see Figure 5.4's action space instances coming from our experimental study), where graph representations remain basic: edge features are often reduced to scalar weights or ignored, and node features are limited to simple indicators (e.g., visited flags). Moreover, these approaches predominantly treat nodes as the sole decision variables, resulting in action spaces that scale linearly with graph size and exhibit relatively simple structure. In such settings, GNN-based node embeddings are typically sufficient to produce well-separated action manifolds, facilitating efficient RL learning. In contrast, real-world applications such as *OSPF* engineering [241] or *Traffic* routing [221] involve more complex decision variables (e.g., paths), resulting in action spaces that grow super-linearly in both size and structural complexity. This discrepancy highlights the gap in the resulting action space complexity between commonly used and real-world benchmarks.

RL-GCO solutions. Existing RL approaches fall short of key operational requirements for generalization and scalability in real-world GCO. Figure 5.4 presents an overview of related work across: (1) action-space complexity, where real-world problems require richer decisions beyond node embeddings h_i; and (2) comparison of our projection method and others based on: *graph-size invariance*, support for *dynamic action sets*, *action semantics* (vs. flat indices), *permutation invariance*, *one-shot inference* (one forward pass for a decision), and support for *action interpolation*. ✓, $\sim$, and $\times$ denote full, partial, and no support, respectively.

- *Discrete* methods rely on padding-based encodings to transform variable-sized graph action spaces into fixed-dimensional representations by padding them to a predefined maximum size, typically assigning one DNN output neuron to each possible action [27]. This enables efficient single-shot inference using standard feed-forward DNNs, but constrains the approach by a maximum graph size and introduces many invalid actions, necessitating invalid-action masking to ignore infeasible choices. Furthermore, actions are represented as positional indices rather than semantic entities, leading to poor generalization across unseen graph topologies. These methods also lack permutation invariance and interpolation capabilities, although they remain efficient due to their one-pass prediction.
- *Iterative* latent action-value methods address the lack of semantic representations by embedding both states and actions into a latent space, often using GNNs, before evaluating them with a learned Q-function [220, 20], which iterates over all state-action pairs. This provides

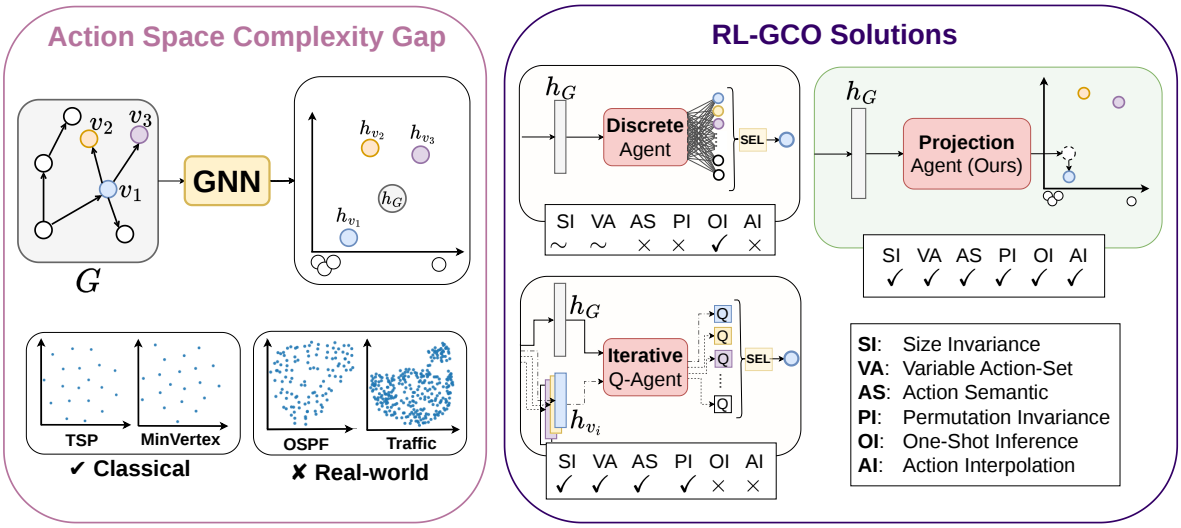


Figure 5.4: Overview of related work on action-space complexity and the operational capabilities of RL methods across several key features.

semantic awareness, permutation invariance, and size invariance through GNN-based representations, while naturally supporting varying action spaces since only feasible actions are evaluated. However, each candidate action requires a separate forward pass through the Q-network, causing inference complexity to scale linearly with the number of valid actions, which becomes expensive for large combinatorial spaces. Moreover, despite using latent embeddings, these approaches still rely on explicit scoring over discrete candidates and therefore do not support interpolation between actions.

- Our proposed *projection* approach instead predicts a target embedding directly within the latent action space and delegates the final action selection to a decoding mechanism operating in that space. Similar to iterative methods, GNN-based embeddings provide semantic awareness, permutation invariance, and size invariance. Moreover, varying action spaces are naturally supported, since the decoding mechanism can be restricted to feasible actions. Additionally, only a single forward pass is required to predict the target embedding, making policy inference independent of the number of candidate actions, except during the decoding stage. This decoding process can nevertheless be implemented efficiently using simple techniques such as 1-NN retrieval, which can be further improved with indexed sublinear search methods [242]. Furthermore, because actions are represented as continuous embeddings, the latent space supports interpolation between semantically related actions, potentially improving generalization, optimization smoothness, and exploration.

5.6 Experiments

This section presents the benchmarks used to evaluate existing RL-GCO solutions compared to our approach, together with the experimental setup and obtained results, with a particular focus on generalization and scalability.

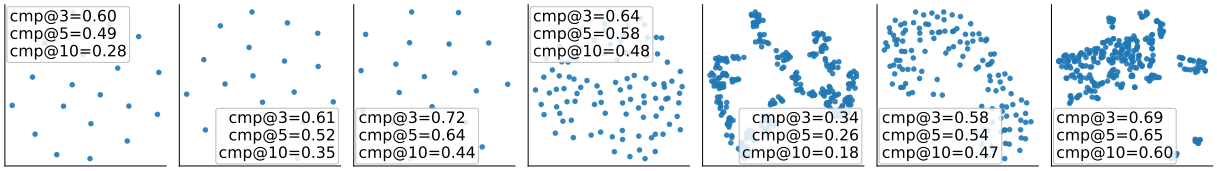
Benchmark environments. We evaluate our approach on seven GCO benchmarks. This includes three standard problems with node-level decisions and linearly growing action spaces—TSP, MinVertex, and Maximum Cut (*MaxCut*) [243]; the latter is of particular importance, as more than half of Karp's 21 Karp's canonical CO problems are reducible to MaxCut [144]. In addition, we consider four application-driven tasks originating from delay-sensitive domains, networking and cybersecurity.
Unlike classical CO benchmarks, these tasks involve structured actions that combine graph entities, leading to super-linear growth in the action space as problem complexity increases.
A brief overview of the logic underlying each benchmark is provided below:

- **TSP:** Finds the shortest tour visiting each city exactly once before returning to the origin. Cities are represented as graph nodes and edges encode pairwise distances. Actions consist of selecting a non-visited node.
- **MinVertex:** This problem seeks the smallest subset of nodes such that every edge is incident to at least one selected node. Actions consist of selecting a non-selected node to include in the cover.
- **MaxCut:** Partitions graph nodes into two sets to maximize the total weight of edges crossing between partitions. Nodes represent graph entities and edges carry weights. Actions consist of selecting and assigning or reassigning a node to a partition.
- **VMP:** Assigns VMs to Physical Machines (PMs) while optimizing utilization, energy efficiency, load balancing, and security constraints (multi-objective function). Nodes represent VMs and PMs, and actions consist of selecting a VM–PM pair to modify the allocation.
- **Attack-Path**: We use the critical attack path discovery benchmark introduced in this thesis, whose underlying logic is described in Section 3.3.1. In the present study, the benchmark is instantiated with small-scale scenarios and simplified dynamics (e.g., source-target-vulnerability action selection) to enable controlled evaluation. In subsequent chapters, it is extended through the more realistic *C-CyberBattleSim* simulator introduced in Section 3.4, allowing a more detailed analysis under complex attack scenarios.
- **OSPF:** Optimizes OSPF routing weights to minimize congestion and maximum link utilization. Nodes represent routers or hosts, while edges correspond to weighted communication links. Actions consist of selecting an edge and a weight variation $\Delta w \in \{-1, 0, +1\}$ for OSPF to reschedule its allocations.
- **Traffic:** Routes communication demands through feasible paths while respecting link capacities and minimizing congestion. Two graph representations share the same host nodes: a communication-demand graph and a topology graph. Actions consist of selecting a communication edge together with a feasible routing path.

Table 5.1 summarizes their action components, the expected growth of the action space, and the structured action representations used for each benchmark (h denotes the node embedding function). It also highlights that *TSP* and *MinVertex* impose hard validity constraints, requiring the sequential selection of previously unvisited nodes. Any violation results in an invalid solution with a score of 0, increasing the likelihood of zero-score outcomes, particularly during early exploration or when the learned policy has not yet captured the required constraints.
TSP, *Cyber-Path*, and *MinVertex* are evaluated in a non-revisiting setting, where decisions cannot be revisited, whereas *MaxCut*, *VMP*, *OSPF*, and *Traffic* allow actions to be revisited within the same episode, such as reallocating VMs from one PM to another.
Additional details on the precise dynamics of each environment, the episode structure, the score

Table 5.1: Action-space structure across benchmarks, including worst-case growth and the chosen representation function.

Benchmark	**Action Components**	**Worst-Case Growth**	**Representation** $(h : \mathcal{V} \to \mathbb{R}^{16})$
TSP	node v (not visited)	$O(\|V\|)$	$h(v)$
MinVertex	node v (not visited)	$O(\|V\|)$	$h(v)$
MaxCut	node v	$O(\|V\|)$	$h(v)$
VMP	source node u (type=VM) target node v (type=PM)	$O(\|V\|^2)$	concat$(h(u), h(v))$
Attack-Path	source node u (status=owned) target node v (status=discovered) vuln $\alpha \in v$	$O(k\|V\|^2)$	concat$(h(u), h(v), \texttt{BERT}(\alpha_{\text{text}}))$
OSPF	edge (u, v) weight change $\Delta w \in \{-1, 0, 1\}$	$O(k\|V\|^2)$	concat$(h(u), h(v), \texttt{onehot}(\Delta w))$
Traffic	edge (u, v) (type=traffic) path $(u_1, \ldots, u_n)$ (type=link)	$O(\|V\|^2 2^{\|V\|})$	pad(concat$(h(u), h(v), ..., h(u_n))$)



(a) TSP (b) MinVertex (c) MaxCut (d) VMP (e) Attack-Path (f) OSPF (g) Traffic

Figure 5.5: UMAP projection of benchmark action spaces for a 20-node graph instance, illustrating local geometry and neighborhood proximity with the compactness metric.

function, and the graph features are provided in Appendix B.1.

Figure 5.5 shows 2D UMAP [244] projections of the action spaces of the benchmarks for 20-node instances. We quantify complexity using compactness, $\text{cmp@}K$, defined as the ratio of the mean pairwise distance to the k-NN distance $(k \in 3, 5, 10)$, averaged over 20 instances of 20 nodes. Real-world benchmarks typically exhibit higher cmp, indicating denser neighborhoods and greater overlap. During scenario generation (details in Appendix B.2), we perform an extended empirical sweep inspired by [220] (e.g., 10k sweeps for *TSP*) to approximate worst- and best-case GCO solutions via standard heuristic-guided exploration for each instance (see Appendix B.3 for details). This produces a normalized reference score in $[0, 1]$, which mitigates scale effects (e.g., distance-dependent costs in *TSP*), enables per-step relative rewards to alleviate the challenges of sparse episodic rewards [144] (with reward measuring how good we are at moving between worst and best known solutions), and ensures comparability and aggregation across instance scores.

Models & baselines. We compare the three RL-GCO approaches from Section 5.5. Discrete baselines are evaluated with two observation encodings—padding (P-, permutation-variant) and GNN-based (G-, permutation-invariant pooling)—with (-M) and without action masking, using

`PPO` and `MaskablePPO` from SB3, respectively. Our projection method uses `PPO` for fair comparison, operating in a z-score–normalized action space with $(K{=}1)$-NN decoding, corresponding to the most challenging high-precision setting in which exact actions must be recovered through accurate latent-space projections alone. We also include the *iterative* solution, implemented via the extension of SB3's fitted Q-iteration `DQN` (re-implemented due to the lack of availability of this solution in existing Python RL libraries). Both latent methods are evaluated only on valid actions (the equivalent of a masking strategy).
We restrict comparisons to RL-based baselines to evaluate performance in a purely autoregressive, data-driven setting. Supervised and relaxation-based approaches are excluded because they either rely on existing labels or struggle to enforce the feasibility constraints characteristic of the benchmarks under consideration. Nevertheless, to provide additional context on solution quality, we report scores normalized to strong heuristic solutions obtained via exhaustive iterative search, with thousands of iterations per scenario. This normalization offers a clearer perspective on the performance of RL methods relative to near-optimal solutions.

Hyperparameter tuning. Solution-specific hyperparameters, along with those governing the GAE and its architecture, are optimized for 25 trials each. The objective is task-specific performance evaluated on a held-out validation set, considering only the *TSP* benchmark for simplicity. The complete search spaces, details of the lightweight neural controller used for the agent, random seeds, and all hyperparameter configurations are provided in Appendix A.2.

5.6.1 Generalization study

This section compares the out-of-distribution generalization capabilities of existing RL-GCO approaches with those of the proposed projection-based agent.

Setup. For each benchmark, we generate 101 diverse scenario instances by sampling key parameters from the typical distributions of each benchmark, in order to evaluate the out-of-distribution generalization capabilities [245] of the methods across varying instance sizes and training configurations. Scenarios are then ordered by size and evaluated under four transfer-oriented training strategies. In the first three, agents train on a single scenario based on their size—*smallest* (S), *medium* (M), or *largest* (L)—and are tested on all remaining instances, resulting in a generalization score; each strategy is repeated $K = 5$ times with different seeds and scenarios (removing those already selected from the selection pool). The fourth, *varied* (V), adopts a K-fold-inspired approach: agents are trained on five non-overlapping subsets of scenarios (20%), with training instances rotated every 100 episodes. For each run, performance is assessed over 5 episodes per unseen instance (80% for V, 100 instances for S, M, and L), with the best result retained; aggregated outcomes across 20 runs per solution–benchmark pair are reported in Table 5.2. We report the Interquartile Mean (IQM) across the 100 best scores, following the guidelines on suitable RL indicators in [246]. Additionally, the Δ columns report the test-to-training gap (averaged across strategies) for each method.

Analysis. Table 5.2 shows that, on hard-constrained classical tasks, *discrete-action* methods consistently underperform relative to both *projection* (ours) and *iterative* approaches, highlighting the advantage of GNN-based semantic embeddings for action selection. The *iterative* framework achieves the best performance on *TSP*, reaching empirical maxima comparable to those obtained with 10k heuristic sweeps, despite using only five trials per instance. This highlights the ability

Table 5.2: IQM of the normalized generalization score across test benchmarks and training strategies.

	TSP					MinVertex					MaxCut				
Method	S	M	L	V	Δ	S	M	L	V	Δ	S	M	L	V	Δ
P-Discrete	0.00	0.00	0.00	0.00	+0.00	0.00	0.00	0.00	0.21	-0.63	0.90	0.89	0.85	0.34	+0.07
P-Discrete-M	0.50	0.47	0.47	0.48	-0.05	0.00	0.05	0.16	0.36	-0.54	0.91	0.90	0.39	0.27	+0.09
G-Discrete	0.00	0.00	0.00	0.00	-0.05	0.00	0.00	0.06	0.02	-0.29	0.00	0.89	0.90	0.89	-0.26
G-Discrete-M	0.51	0.50	0.50	0.50	-0.19	0.00	0.07	0.16	0.11	-0.24	0.00	0.90	0.91	0.90	-0.25
Iterative	0.90	0.79	0.97	**0.99**	-0.04	0.39	0.07	0.00	0.15	-0.19	0.92	0.85	0.80	0.82	+0.00
Projection (ours)	0.78	0.76	0.60	0.71	-0.13	**0.63**	0.00	0.03	0.08	-0.22	**0.95**	0.95	0.92	0.94	-0.01

	VMP					Attack-Path					OSPF					Traffic				
Method	S	M	L	V	Δ	S	M	L	V	Δ	S	M	L	V	Δ	S	M	L	V	Δ
P-Discrete	0.55	0.52	0.38	0.55	-0.36	0.19	0.18	0.19	0.18	-0.05	0.20	0.04	0.00	0.07	-0.31	0.26	0.49	0.10	0.20	-0.30
P-Discrete-M	0.64	0.65	0.66	0.65	-0.29	0.61	0.63	0.61	**0.67**	-0.20	0.07	0.39	0.25	0.01	-0.16	0.41	0.73	0.74	0.76	-0.10
G-Discrete	0.11	0.40	0.49	0.34	-0.49	0.18	0.19	0.18	0.17	-0.05	0.16	0.13	0.14	0.52	-0.63	0.24	0.57	0.49	0.62	-0.39
G-Discrete-M	0.07	0.36	0.60	0.51	-0.53	0.45	0.45	0.51	0.50	-0.28	0.26	0.38	0.64	0.67	-0.41	0.73	0.73	0.78	0.78	-0.15
Iterative	0.39	0.12	0.03	0.15	-0.10	0.34	0.24	0.19	0.24	-0.22	0.00	0.03	0.00	0.00	-0.20	0.00	0.00	0.00	0.00	-0.20
Projection (ours)	0.85	0.88	0.86	**0.91**	-0.10	0.60	0.64	0.60	**0.67**	-0.17	0.68	0.78	0.84	**0.88**	-0.12	0.80	0.80	0.81	**0.83**	-0.05

of RL agents to match strong heuristic baselines under limited evaluation budgets (5 trials). On *MinVertex* and *MaxCut*, *projection* outperforms *iterative*, and the limitations of the latter become dramatically more pronounced in realistic benchmarks. As action spaces grow superlinearly in applied benchmarks, the method struggles to produce reliable Q-value estimates, leading to degraded performance that falls even below *discrete* baselines. In contrast, the *projection* agent remains stable across settings, achieving the strongest performance on applied benchmarks while remaining competitive on classical ones.

The *projection* approach also shows the best overall trade-off between test performance and Δ values, which capture transfer capability (via low magnitudes), particularly on realistic benchmarks. The *Attack-Path* prediction benchmark shows a shared top ranking between the projection approach and discrete padding with masking. However, this result will be further investigated in the following chapters using larger-scale scenarios and more complex, realistic environmental dynamics.

Finally, the impact of training scale reveals a consistent pattern. On classical benchmarks, the optimal regime is typically S when using the *projection* agent, with no clear benefit from denser or more varied action spaces. In contrast, V emerges as the most effective regime on real-world benchmarks, suggesting that *projection* can support improvements from exposure to diverse and heterogeneous action spaces (not always observed with V in other approaches).

5.6.2 Inference time scalability

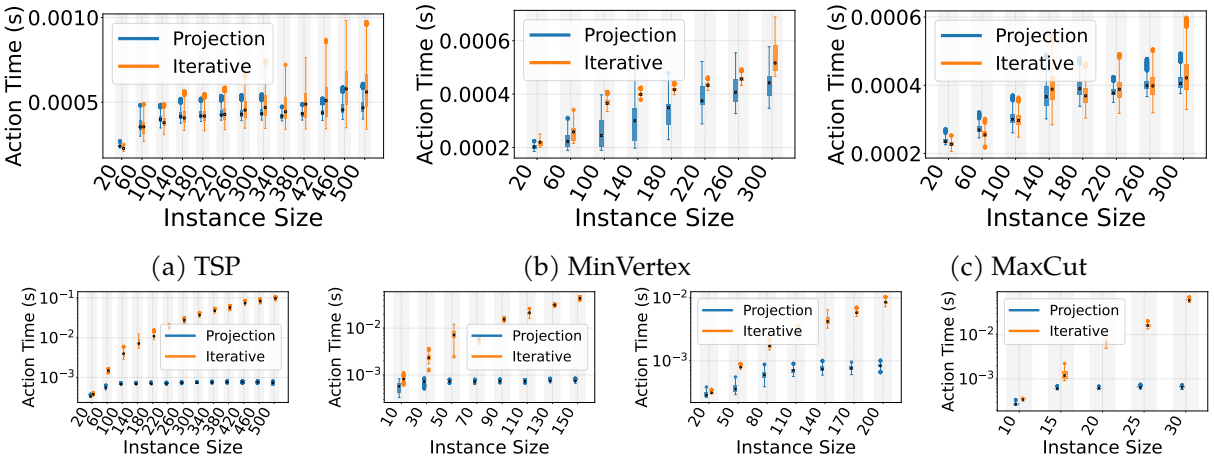
This section evaluates the inference-time efficiency and scalability of the proposed projection-based agent in comparison with the iterative latent-action approach.

Setup. We study computational scalability by analyzing how action-selection time evolves with graph size for both the proposed GNN-based *projection* method and *iterative* methods. The projection approach relies on a FAISS Flat Index for NN retrieval with linear retrieval complexity [247], while iterative approaches explicitly evaluate candidate actions. Experiments

are conducted on a benchmark machine.[2]
To characterize scalability, we fit a power-law model:

$$T(n) = c \cdot n^{\alpha},$$

where $T(n)$ denotes the median action-selection time for a graph of size n, and α is the empirical scaling exponent. The *discrete* baselines are omitted from this analysis, as they require only a single DNN inference and no additional search operations. Consequently, their inference time remains approximately constant with respect to graph size when the function approximator architecture is fixed.



(a) TSP (b) MinVertex (c) MaxCut

(d) VMP (e) Attack-Path (f) OSPF (g) Traffic

Figure 5.6: Evolution of action-selection time across graph sizes and benchmarks.

Analysis. Table 5.3 reports the estimated exponent α together with the corresponding R^2 values across benchmarks, while the full runtime curves used to derive these metrics are shown in Figure 5.6. The results highlight the computational advantage of the *projection* approach, which selects actions via NN retrieval in a latent space rather than repeatedly evaluating a DNN on all candidate actions.
This distinction becomes particularly important in the proposed delay-sensitive benchmarks, where the action space grows superlinearly with graph size due to structured combinations of entities and parameters. In such settings, iterative methods rapidly become computationally impractical as problem size increases, whereas the projection-based agent exhibits substantially better scalability and remains tractable on large instances. This property is particularly important in networking applications, where real-world scenarios often involve large and dynamically evolving graphs.

	Projection (ours)		Iterative	
Benchmark	α	R^2	α	R^2
TSP	**0.17**	0.85	*0.25*	0.94
MinVertex	**0.30**	0.88	*0.32*	0.95
MaxCut	**0.22**	0.94	*0.25*	0.92
VMP	**0.22**	0.85	*1.81*	0.99
Attack-Path	**0.09**	0.76	*1.46*	0.95
OSPF	**0.49**	0.94	*1.44*	0.99
Traffic	**0.73**	0.71	*4.68*	0.99

Table 5.3: Scaling of action-selection time with fitted power-law exponent α.

[2] *Benchmark Machine:* AMD Ryzen 7 PRO 7840U w/ Radeon 780M Graphics, 30Gi RAM, Ubuntu 22.04.4

5.7 Summary

We introduce an end-to-end framework for learning latent action spaces for RL-GCO. Building on this formulation, we propose a projection-based action-selection mechanism that navigates the latent action space by first generating a *desired* action representation (*what the agent needs*) and then projecting it onto the closest valid action available in the current problem instance. This design avoids explicit evaluation over the full action space and enables scalable decision-making in large, structured graph scenarios. In addition, we design a suite of benchmarks that capture more realistic action-space structures inspired by networking and cybersecurity applications, in which actions involve combinations of graph and non-graph entities and the action space grows superlinearly with problem size. Experimental results demonstrate consistent generalization improvements of the proposed approach on these realistic benchmarks, achieving an average relative performance gain of (19%), together with an average inference-time speedup of $\times 8.45$. Importantly, these improvements are consistent across all real-world application benchmarks considered. To support reproducibility and future research on other benchmarks, the `LaGCO-RL` library facilitates the integration and evaluation of new GCO environments and RL-GCO solutions.

Link to the thesis objectives

The proposed projection agent has been designed to improve scalability and generalization in RL-GCO compared with both widely used iterative latent solutions and standard discrete approaches. Chapter 6 further investigates this framework in the context of critical attack path discovery, extending it with adaptations tailored to the challenges of this benchmark, which is the focus of the thesis, thereby highlighting its advantages in scalability and generalization. The same framework is subsequently reused in Chapter 7, where these capabilities are transferred to the VMP benchmark studied as a use case from a security-oriented perspective.

Chapter 6

Scalable and generalizable RL agents for critical attack path discovery

Summary: *This chapter presents the adaptation and extension of the projection agent introduced in Chapter 5 for the task of critical attack path discovery. In addition to this adaptation, the chapter examines the key challenges associated with the problem, including scalability, generalization, and dynamicity in network environments, and proposes a practical solution for real-world deployment.*

Contents

6.1 Introduction . . . 107
6.2 Action space dynamics . . . 109
6.3 Existing approaches . . . 110
6.3.1 Agent formulation . . . 110
6.3.2 Benchmark scenario complexity . . . 111
6.4 Methodology . . . 112
6.4.1 Graph representation . . . 112
6.4.2 Discrete local spaces . . . 114
6.4.3 Continuous latent spaces . . . 115
6.5 Experiments . . . 117
6.5.1 Experimental setup . . . 117
6.5.2 Scalability study . . . 119
6.5.3 Generalization study . . . 120
6.5.4 Dynamicity study . . . 122
6.6 Deployment-oriented study . . . 123
6.7 Summary . . . 125

6.1 Introduction

The problem of critical attack path discovery, which is the primary focus of this thesis, can be formulated as a GCO problem (see Section 2.4). The objective is to identify critical sequences of vulnerability exploitations within a networked environment that are consistent with a specified threat model. In this context, an agent must reason over a partially observable network and its progressively evolving representation, while sequentially selecting attack actions that maximize

the attacker's objective. To achieve this, the agent must continuously adapt its decision-making process in response to newly discovered information and changes in the system state.
The use of RL for this problem is especially attractive for several additional reasons extending the motivations discussed previously in Section 5.5. First, this application domain suffers from a severe lack of large-scale supervised datasets [248]. This limitation stems from the sensitivity and confidentiality of historical attack data, as well as the scarcity of comprehensive scans of real-world networks that could be used to apply attack graph generation and symbolic execution techniques to derive optimal attack paths. Consequently, obtaining high-quality labeled data at scale is extremely challenging, making it difficult to train, generalize, and deploy supervised learning approaches effectively in practice. Second, the sequential decision-making formulation of RL naturally aligns with the structure of real-world attack campaigns, in which attackers progressively exploit vulnerabilities to unlock new opportunities, closely reflecting realistic multi-stage intrusion processes. As a result, RL policies can learn attacker behaviors representative of plausible and critical attack campaigns while also producing interpretable attack traces through the sequence of selected actions with respect to the evolution of the system states [110].
The problem becomes particularly challenging under the action-space formulation of this task, as defined in the POMDP of Section 3.3.1, where the agent must jointly select a source node, a target node, a vulnerability in the target, and an outcome dimension for exploitation. In this formulation, the action space grows more than quadratically with the graph size, since the vulnerability-outcome dimensions also depend on the number and diversity of vulnerabilities present at each target node. Moreover, these dimensions are inherently dynamic, as vulnerability, validity, and exploitability may vary with the specific attacker state. Consequently, policies must learn representations that generalize simultaneously across graph structures, vulnerability semantics, and evolving attacker knowledge. Compared to classical GCO tasks, this introduces significantly stronger scalability and generalization challenges.
Scalability, generalization, and adaptation to dynamic conditions [249] are critical requirements in this application domain. Retraining RL policies for every network configuration or environmental change is impractical, particularly because the critical attack path discovery problem rarely presents identical operating conditions across different scenarios [250]. Moreover, even a single network evolves continuously over time due to real-world events such as node additions and removals, changes in connectivity, software updates, configuration modifications, and the emergence of new vulnerabilities. Consequently, effective solutions must generalize across diverse network topologies and adapt to evolving environments without requiring retraining for each new configuration. This highlights the need for representations and learning mechanisms that are decoupled from the specific structural characteristics of individual graph instances while preserving the information necessary for decision-making.
To this end, this chapter extends the solution introduced in Chapter 5 by proposing embedding-based representations for observations and actions specifically designed for the critical attack path prediction task.
Furthermore, existing works on RL-based critical attack path prediction often lack extensive evaluations of scalability and generalization, and rarely provide comparisons with manually designed heuristics to justify the practical benefits of RL for this application. This chapter aims to address these limitations by systematically studying these crucial axes and analyzing how different RL formulations behave as network complexity and vulnerability diversity increase.

Contributions

This chapter provides the following main contributions to address scalability and generalization challenges of RL-GCO to the critical attack path prediction benchmark:

- An **adaptation** of the **projection-based RL agent** for the task of critical attack path prediction, including a methodology for constructing latent observation and action spaces.
- A comparative **analysis** of the **scalability** and **generalization** capabilities of projection-based approaches against widely used discrete RL-GCO solutions.
- An extensive **comparison of DRL algorithms** operating on latent spaces and their effectiveness in learning strong policies for this task.
- A **comparison** between RL-based policies and **manual heuristics**, highlighting the practical advantages of learned heuristics for critical attack path prediction.

6.2 Action space dynamics

The most expressive action-space formulation of the POMDP introduced in Section 3.3.1 for the critical attack path discovery task represents actions as the tuple $(source, target, vulnerability, outcome)$, where the *source* denotes a node already discovered and controlled by the attacker, the *target* denotes a discovered node that can be attacked, the *vulnerability* corresponds to a vulnerability present on the target node, and the *outcome* represents the desired effect for the exploit.
Compared to many of the benchmarks studied in Chapter 5, this formulation introduces substantially higher complexity due to the dynamic nature of both the attack process and the vulnerability landscape. In fact, the set of valid actions is not fixed throughout the episode but evolves continuously as the attacker interacts with the environment and progressively discovers new attack opportunities. Several sources of action-space dynamics emerge in this context, particularly regarding both the expansion of the action space and the evolution of action validity:

- **Target discovery.** Initially, only a limited portion of the network may be visible to the attacker. Reconnaissance actions and successful exploits can reveal previously unknown hosts and communication paths. Each newly discovered node introduces additional candidate *target* nodes, dynamically expanding the action space along this dimension.
- **Source expansion.** At the beginning of an episode, the attacker typically controls only a single node with limited capabilities. As unknown hosts are discovered, exploits succeed, and lateral movement is achieved, additional hosts become compromised. These newly compromised nodes can subsequently be used as new *source* nodes, progressively increasing the number of feasible attack actions available to the agent.
- **Vulnerability dimension growth.** Vulnerabilities are associated with specific services and assets deployed on target nodes. As new nodes are discovered, or as attacker privileges evolve (*privilege escalation* outcomes), the set of vulnerabilities accessible to the attacker also changes. Consequently, the vulnerability dimension of the action space expands as the network discovery process progresses.
- **Outcome variability.** A single vulnerability exploitation may lead to multiple possible outcomes, such as credential access, privilege escalation, persistence, or discovery. Therefore, the same source-target-vulnerability triplet may correspond to several distinct actions,

depending on the outcome dimension considered.

- **Network and software-level dynamics.** Beyond changes in the attacker's progress, the software environment within nodes may also evolve over time. When considering the underlying scenario as dynamic, service installation or shutdown, software updates, configuration changes, or defense mechanisms may alter the set of exploitable vulnerabilities associated with a node. As a result, actions that were previously valid may later become invalid in dynamic settings.

These characteristics create important scalability, robustness, and generalization challenges for RL agents in this task. Policies must not only generalize across varying graph topologies and vulnerability distributions but also adapt to continuously evolving action spaces whose validity changes dynamically during the attack process.

6.3 Existing approaches

This section reviews related work from two complementary perspectives. First, it examines the discrete-agent formulation, which is predominantly adopted in the literature, along with its associated limitations. Second, it analyzes the characteristics of benchmark scenarios used in previous studies that address the attack path prioritization problem under POMDP formulations similar to those considered in this work.

6.3.1 Agent formulation

All existing studies that address the critical attack path prediction task using RL solutions rely on discrete representations for both the observation and action spaces. In particular, approaches such as [251], [252], and [253] adopt a *global discrete* representation of the environment, following formulations similar to Equation 6.1 (representing a task-specific adaptation of the *discrete* framework discussed in Section 5.5). These methods constitute the primary body of related work for this task and therefore serve as the main baseline against which our approach is compared.

$$\pi_{\text{global}} : \mathbb{R}^{N \times f} \rightarrow \mathbb{R}^{N \times N \times |V| \times |O|} \tag{6.1}$$

In approaches relying on this global discrete view of the environment, the observation vector represents the discovered network graph as a flattened array obtained by concatenating the feature vectors of all discovered nodes, each of dimension f. Since the number of discovered nodes may vary over time within the POMDP framework, the observation is typically padded to a predefined maximum graph size N. Consequently, the input layer of the policy network is instantiated with a fixed dimensionality of $N \times f$, requiring all possible scenarios to be represented within this predefined observation size.
Similarly, the discrete action space is typically defined as the set of all possible tuples composed of a source node, target node, vulnerability, and outcome, with each dimension padded to a predefined maximum size. The output layer of the policy network is therefore instantiated with one output neuron for each possible source–target–vulnerability–outcome combination, resulting in an output dimensionality of $N \times N \times |V| \times |O|$. The agent must consequently assign a score to every possible action combination, including those that are invalid in the current environment state. As a result, the size of the action space (and output layer) grows rapidly with key environment characteristics that can vary substantially across scenarios, most notably the graph size and vulnerability set.
These representations introduce several important limitations:

- **Scalability:** The observation and action spaces grow rapidly with the number of nodes, vulnerabilities, and outcomes. Even when the outcome dimension is fixed (e.g., the 14 MITRE ATT&CK tactics as of March 2026), it still increases the number of possible source–target–vulnerability combinations by a constant factor. Furthermore, a large proportion of these combinations correspond to invalid actions at a given timestep, yet the agent must still dedicate output neurons and learning capacity to represent them. This leads to inefficient use of model capacity and limits scalability as scenario complexity increases.
- **Generalization:** Flattened representations encourage the agent to learn patterns that depend on the specific ordering of nodes and vulnerabilities within the input representation rather than on their underlying structural and semantic properties. The same limitation applies to the action space, where discrete actions are commonly represented through a fixed output layer with one output neuron per possible action combination. Consequently, the learned policy is tied not only to positional encodings but also to the specific action set observed during training. Any previously unseen action combination, newly introduced vulnerability, or additional graph element that was not included when defining the output layer cannot be directly selected by the agent, even if it would be a valid and potentially optimal action. Therefore, these approaches limit transferability across scenarios and hinder adaptation to unseen vulnerabilities, network topologies, and configurations.
- **Dynamicity:** Because observations and actions are padded to predefined maximum dimensions, these approaches are inherently constrained to fixed graph and action-space sizes. However, real-world networks continuously evolve through changes in topology, service deployments, configurations, and vulnerability inventories. Accommodating larger or structurally different scenarios therefore often requires modifying the policy parameterization and subsequently fine-tuning or retraining the policy, limiting adaptability to dynamic operating conditions.

6.3.2 Benchmark scenario complexity

As mentioned before, previous studies addressing the attack path discovery problem under POMDP formulations have all relied on the discrete global formulation presented in Section 6.3. Due to the inherent scalability limitations of their action representations, whose dimensionality increases rapidly with the number of nodes and vulnerabilities in the scenarios of interest, existing approaches have generally been evaluated on benchmark scenarios with relatively limited complexity, both in terms of network size and vulnerability-space dimensionality.
Table 6.1 summarizes the characteristics of the benchmark scenarios and evaluation objectives considered in related work employing POMDP formulations similar to those adopted in this study, particularly approaches based on *CyberBattleSim*-like environments.
The table highlights that existing evaluations have generally been conducted on relatively small-scale environments and rarely capture the complexity characteristics of even medium-sized networked systems. An additional challenge is the lack of standardized reporting across studies. Many works do not explicitly specify the number of nodes, the size of the vulnerability set, or both, making direct comparisons between approaches difficult.
To provide a broader perspective on the state of the literature, we also indicate whether each study explicitly evaluates key challenges, including scalability, generalization, and adaptation to dynamic environments. While the dynamic nature of cyber environments is frequently considered, it is often addressed as an isolated research objective, for example through attacker interactions with dynamic defenders or jointly trained attacker–defender settings. In contrast,

Table 6.1: Comparison of benchmark scenario complexity and evaluation dimensions in related work.

Study	# Nodes	# Vulnerabilities	Generalization	Scalability	Dynamicity
[254]	10	11	No	No	No
[255]	Not Reported	Not Reported	No	No	Yes
[256]	10	11	No	No	Yes
[257]	10	12	No	No	No
[258]	15	Not Reported	No	No	Yes
[259]	10	11	No	No	Yes
[260]	Not Reported	Not Reported	No	No	No
[251]	17	Not Reported	Yes	No	Yes
[253]	30	7	Yes	No	No
[252]	10	11	No	No	No
[261]	15	15	No	No	No
Ours	250	250	Yes	Yes	Yes

other critical properties required for practical RL-based GCO solutions, such as scalability to larger topologies and vulnerability spaces, as well as the transferability of learned policies across previously unseen environments, remain comparatively underexplored.
To the best of our knowledge, the present study considers the largest benchmark scenarios evaluated to date in terms of both the number of unique nodes and vulnerabilities, while simultaneously investigating scalability, generalization, and adaptation to dynamic environments.

6.4 Methodology

In this section, we present the graph representation proposed for this task, as well as alternative agent formulations that improve on the *discrete global* approach, with the objective of enhancing the robustness of learned policies to the challenges discussed in Section 6.2.

6.4.1 Graph representation

The graph representation used in this study to model the environment at timestep t builds upon the host-based attack graph introduced in Section 3.3.1. The graph consists of nodes representing network hosts, each associated with a set of host-specific features, vulnerabilities, and attack-history information. To remain consistent with the partial observability assumptions of the POMDP, the graph is restricted to hosts that have been discovered by the attacker at timestep t. Furthermore, when only partial information about a discovered host is available, the corresponding components of its feature vector are masked according to the visibility conditions defined by the attack state.
A key design objective is to represent these characteristics without relying on the number or order of vulnerabilities associated with a host. In addition, the representation is designed to explicitly capture attack history, enabling the agent to distinguish between previously exploited and unexplored attack opportunities.

To achieve this, vulnerabilities are encoded into a fixed-dimensional latent space according to their semantic meaning using LMs combined with appropriate pooling operations. The resulting pooled embedding provides a compact representation of the vulnerabilities present on a host while maintaining a fixed dimensionality and permutation invariance. Consequently, the node feature vector remains independent of both the size and ordering of the vulnerability set associated with each host (Figure 6.1).

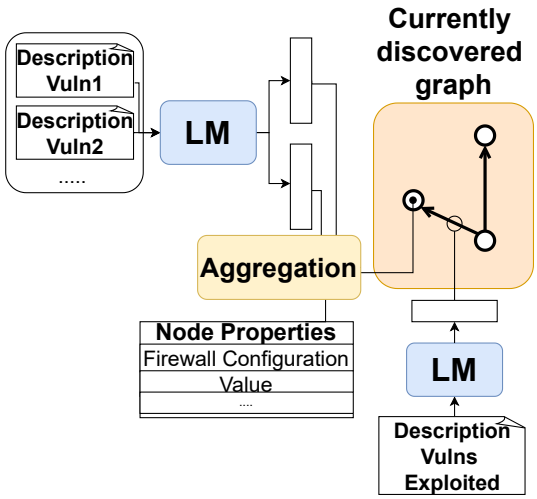


Figure 6.1: Graph representation of the environment scenario, illustrating the design of node and edge feature vectors.

Together with the vulnerability representation, each node feature vector incorporates several additional attributes that capture the assets, obstacles, and other security-relevant properties defined in the POMDP formulation presented in Section 3.3.1. These attributes are aggregated into a unified node representation, preserving the host-based attack graph abstraction while providing the agent with a compact description of the environment observation.

- **firewall_config_array**: An array detailing the node's firewall configuration. Each element corresponds to a port and indicates whether an outgoing or incoming BLOCK rule is present.
- **listening_services_running_array**: An array with binary values indicating whether each service on the node is currently running.
- **visible**: A binary value representing the level of visibility of the node, representing whether it is fully visible or partially visible.
- **persistence**: A binary value indicating whether the agent has achieved persistence on the node.
- **has_data**: A flag indicating whether data is present on the node, treated as a binary value.
- **data_collected**: A binary value indicating whether data has been collected from the node.
- **data_exfiltrated**: A binary value indicating if data has been exfiltrated from the node.
- **defense_evasion**: A binary value reflecting whether the agent has evaded the defense systems on the node (firewall and potential defenders).
- **privilege_level**: An integer denoting the agent's privilege level on the node (*No Access*,

Basic User, *ROOT*).

- **status**: An integer value representing the current status of the node, such as running, re-imaging, or stopping.
- **value**: Importance value assigned to the node.
- **sla_weight**: The weight associated with the node's service level agreement, reflecting its criticality.
- **listening_services_embeddings**: A vector with the pooled embeddings of services, derived with LMs applied to their names.

Edges of the graph represent the vulnerabilities exploited between nodes up to the current timestep, with their feature vectors again pooling the embeddings of the exploited vulnerabilities (derived with LMs over their descriptions). This structure enables the agent to encapsulate the current state of discovery, the historical context of vulnerabilities exploited up to the current timestep, and the semantics of vulnerabilities within the network.
This graph representation can be leveraged by a variety of RL-GCO formulations, each encoding the underlying information into observation and action spaces according to its specific design. The representation therefore provides a common, topology- and vulnerability-aware abstraction of the environment upon which different learning formulations can be built.

6.4.2 Discrete local spaces

As a first comparison point over agent formulations, while still maintaining a discrete formulation, we propose a *discrete local* view (Equation 6.2). Unlike the global approach, this formulation restricts the agent to operating on a localized subset of the graph at each timestep, aiming to alleviate several of the scalability and generalization issues identified in Section 6.3.

$$\pi_{\text{local}} : \mathbb{R}^{2\times f+g} \rightarrow \mathbb{R}^{(|V|\times|O|)+|S|} \tag{6.2}$$

This formulation leads to a significantly smaller observation space by focusing only on a single source-target node pair. The resulting observation vector therefore has a size of $2 \times f$, corresponding to the feature representations of the current source and target nodes. Optionally, this representation can be augmented with a size- and order-invariant graph feature vector g that contains condensed structural information about the overall graph.
The discrete action space requires the agent to assign a score to every possible vulnerability–outcome pair associated with the currently selected source–target pair, in addition to a small set of graph-size-invariant switching actions (S) (e.g., *switch source* and *switch target*). Consequently, the action space size becomes $(|V| \times |O|) + |S|$.
Compared to the discrete global formulation, this approach introduces several improvements and remaining limitations:

- **Scalability:** The formulation removes the direct dependency on the number of nodes N, significantly improving scalability with respect to network size. However, the approach still scales with the vulnerability set size $|V|$, which may remain large in realistic environments.
- **Generalization:** The localized representation reduces sensitivity to node ordering, improving transferability across different graph topologies. Nevertheless, ordering dependence persists across the vulnerability and outcome dimensions, limiting full semantic generalization.
- **Dynamicity:** By operating on local source-target pairs, the approach naturally adapts to varying graph sizes without requiring a globally padded node representation. The number

of invalid actions is also reduced, since invalid source-target combinations are no longer explicitly represented in the action space. However, adaptation along the vulnerability dimension remains challenging, as changes in the vulnerability set introduce limitations similar to those encountered in the global discrete formulation.

Despite these advantages over the global discrete formulation, restricting the agent to a local view may lead to suboptimal decisions, since identifying critical attack opportunities can require multiple switching actions before an effective critical exploitation action can be performed.

6.4.3 Continuous latent spaces

The latent observation and action spaces introduced in Chapter 5 are extended and adapted in this chapter to address the critical attack path prediction problem. These representations are designed to be invariant to the underlying graph structure, as well as to variations in vulnerability set size and ordering. This enables a single agent to better address the scalability, generalization, and adaptation challenges associated with dynamic environments, which are particularly relevant for this benchmark.

First, the graph representation introduced in Section 6.4.1 is adopted as the underlying structure for unsupervised representation learning. Node embeddings are then generated following the strategy presented in Section 5.4.1, where the host-based attack graph is combined with node-level information to learn a latent representation of each node. Using this approach, embeddings capture both the semantic characteristics of individual nodes, including their associated vulnerabilities through LM-based representations, and their structural context within the graph. Consequently, the resulting representations encode information about the current attack state and the historical progression of the attack process.

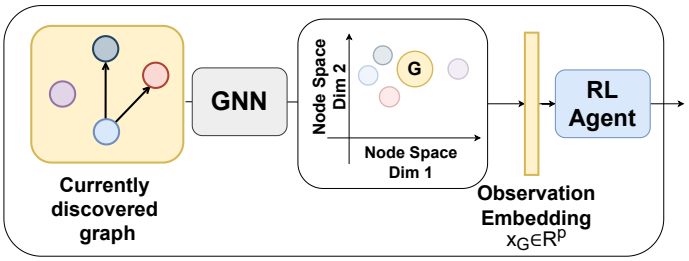


Figure 6.2: The discovered graph is processed by a GNN to generate node embeddings, which are pooled into a fixed-dimensional graph embedding used as the agent's observation.

Building upon these node embeddings, the observation space can then be constructed following the principles introduced in Section 5.4.2 and commonly adopted in the graph representation learning literature, where graph-level embeddings are obtained through appropriate pooling operations over node representations (Figure 6.2).

Regarding the action space, as the agent is tasked with selecting a tuple (*source*, *target*, *vulnerability*, *outcome*) at each timestep, different models should be used to generate embeddings for each tuple element and form the action space:

- For the **source** and **target nodes**, p-dimensional embeddings can be generated using the pre-trained GNN for representation learning.
- For **vulnerability** choices, embeddings can be derived from the textual descriptions using pre-trained LMs. These models are used to generate continuous vector embeddings of size q, capturing the semantics of the vulnerability description.

- **Outcome** embeddings can be represented as one-hot vectors of size $|O|$, where O denotes the outcome set. Depending on this dimensionality is not critical, since this set remains consistent across scenarios when relying on standard taxonomies such as MITRE ATT&CK tactics to represent outcomes.

Once embeddings for each element of the tuple have been constructed individually (Figure 6.3), a unified latent action space can be formed, where each point represents a semantic tuple of the form (*source, target, vulnerability, outcome*). To construct the action space, we perform Cartesian products of all possible action elements by concatenating all possible combinations of source node embeddings with those of the target nodes, followed by the embeddings of the possible vulnerabilities and outcomes. This results in each action point being represented as a vector of dimensionality $p + p + q + |O|$, with each position semantically meaningful, as the vector is derived from the concatenation of meaningful embeddings within their respective spaces.

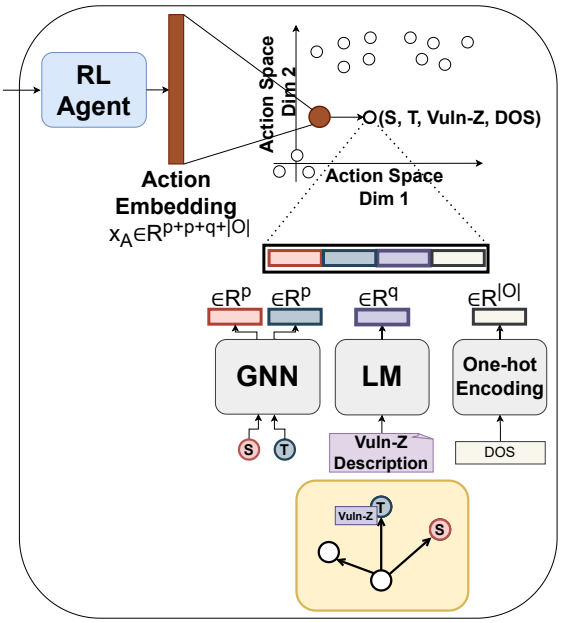


Figure 6.3: The action space is defined as the set of all possible concatenations of source, target, vulnerability, and outcome embeddings.

As depicted in Figure 6.3, the *projection agent* is trained to generate a latent action vector representing a desired position within the learned action embedding space. Based on this prediction, a proximity-based decoding strategy (e.g., nearest-neighbor retrieval), as described in Section 5.4.3, selects the closest valid action embedding while preserving the same methodology and adapting it to the current benchmark. The selected embedding is then mapped back to the corresponding discrete action tuple in the original action space, which is executed by the environment simulator.

As a result of the fixed-dimensional vectors, the projection agent defines input and output spaces whose dimensions are independent of the graph structure and vulnerability set, and depend solely on the embedding dimensionality. As a result, the agent parameterization can support varying graph structures and vulnerability sets (Equation 6.3).

$$\pi_{continuous}: \quad \mathbb{R}^p \rightarrow \mathbb{R}^{p+p+q+|O|} \tag{6.3}$$

This approach enables the agent to operate on the latent spaces produced by two invariant representation models: one capturing graph structure and the other capturing vulnerability semantics. Rather than treating nodes, vulnerabilities, and actions as isolated discrete entities, the agent learns to navigate structured embedding spaces that encode their underlying relationships and properties.

The advantages of this formulation with respect to the key challenges of the critical attack path prediction problem are summarized below:

- **Generalization:** Latent representations promote generalization by mapping semantically similar graphs, nodes, and vulnerabilities to nearby regions of the embedding space. As a result, knowledge acquired in one scenario can be transferred to scenarios exhibiting

different topologies or vulnerability distributions but similar underlying characteristics. Unlike discrete representations, embeddings preserve structural and semantic information, allowing the agent to exploit similarities across environments. When combined with exposure to diverse training scenarios, for example, through domain randomization, this representation can support learning policies that generalize beyond the scenarios encountered during training, as investigated in Section 6.5.3.

- **Scalability:** In discrete formulations, the dimensionality of the observation and action spaces typically grows with the number of nodes, vulnerabilities, and other scenario-specific elements. In contrast, latent representations maintain a fixed dimensionality regardless of environment size. Consequently, scalability challenges shift from managing increasingly large observation and action vectors to learning effective representations and navigation strategies in the latent space.
- **Dynamicity:** Latent representations naturally accommodate evolving graph structures and vulnerability sets. New nodes, vulnerabilities, or topology changes can be encoded into the same fixed-dimensional embedding observation spaces without modifying the underlying DNN input layer. Similarly, actions are represented in a fixed-dimensional latent space, while the decoding mechanism maps latent actions to the currently available entities. As a result, the same policy architecture can operate across dynamically changing environments without requiring structural redesign.

6.5 Experiments

This section first presents the experimental setup used to generate the POMDP instances considered in this study. It then provides an empirical comparison of the *scalability* of agents based on discrete and continuous latent-space formulations. Given the stronger invariance properties of continuous latent representations, the study further investigates their impact on *generalization* to previously unseen scenarios and on *adaptation to dynamic environments*.

6.5.1 Experimental setup

The experimental study relies on the POMDP formulation and the *C-CyberBattleSim* implementation introduced in Chapter 3. The following experimental setup choices are adopted:

- The **environment database**, which is used to populate synthetic network scenarios represented as graphs (as outlined in Section 3.3.2), is constructed using Shodan data. It identifies 172 service versions and 829 unique vulnerabilities by querying the "50 most frequently used services globally, and their top 5 versions."
  The following are the services identified from the Shodan query (at the timestep of 18 November 2024) and the node types simulated:

  Services: Nginx, Apache httpd, Squid http proxy, Microsoft IIS httpd, Exim smtpd, Jetty, MongoDB, Remote Desktop Protocol, Jenkins, Wildix Collaboration, Apache Tomcat, NET-DK, Metabase, Outlook Web App, Tengine, VMware ESXi, OpenSSH, OpenResty, MySQL, Control Web Panel, Apache Tomcat/Coyote JSP engine, Nextcloud, Elastic, lighttpd, GoAhead Embedded Web Server, Hikvision IP Camera, RabbitMQ, Boa Web Server, Grafana, micro_httpd, Boa HTTPd, DrayTek Vigor Series (2925, 2862, 2860, 2762, 2912, 2926, 2133, 2927, 2865), ZTE H268A, ZTE ZXHN H168N, Cisco Systems, ZTE F680, Bbox.

Types: Windows Host, Unix Host, IoT node, Industrial Control System (ICS) node, Router.

- The environment database is combined with a set of ranges for graph-based and cyber-terrain parameters (detailed in Appendix C.1) to generate a total of 800 synthetic scenarios through **domain randomization**. Each generated scenario produces a distinct graph topology and vulnerability distribution, enabling extensive robustness studies across diverse environments.
- The **outcome approximator** for CVEs without known MITRE ATT&CK mappings, integrated into the C-CyberBattleSim simulator to model potential vulnerability outcomes, is based on SecureBERT. The model is fine-tuned for flat multi-label classification of MITRE ATT&CK Tactics from vulnerability descriptions, following the approach presented in Chapter 4, and achieves a reported test-set F1 score of 76%.

The choices for RL episodes, scoring metrics, and RL libraries are as follows:

- Considering the **episode evolution**, agents undergo each training and testing episode by varying the starter node and scenario from the corresponding set (e.g., training set). The episode length scales with the graph size as $K \times \#nodes$, with K defined as the *episode-length coefficient*, ensuring unbiased scores across varying graph sizes. This finite action budget encourages the agent to prioritize the most critical vulnerability paths and maximize impact within the given action frame.
- We use *control*, *discovery*, and *disruption* goals as experimental **threat models**, varying the goal in each run. The reward function is structured into several components organized into bonuses and penalties. These terms have been carefully selected and adjusted based on the specific goals (*control*, *discovery*, *disruption*) and their semantics (details on the specific values in the simulator). The list of tuned bonuses and penalties is as follows:

  Reward bonuses: Node Owned, Node Disrupted, Node Discovered, Data Collected, Privilege Escalation, Acquired Visibility, Data Exfiltrated, Defense Evaded, Episode Won

  Reward penalties: Vulnerability Cost, No Enough Privileges, Success Rate Failed, No Data to Collect, No Data to Discover, No Data to Exfiltrate, Already Persistent, Machine Already Stopped, Node Already Owned, Node Already Visible, Already Defense Evasion, Scanning Unopen Port, Privilege Escalation in Node Not Owned, Privilege Escalation to Level Already Had, Blocked by Local/Remote Firewall, Episode Lost

  The **goal scores** used for the evaluation of the paths are defined here and used throughout the experiments: percentage of nodes controlled and with maximum privilege (ROOT privilege) for *control*, the amount of information gathered and exfiltrated (including node IDs and their data) for *discovery*, and the percentage of nodes disrupted for *disruption*. These scores are normalized relative to the maximum achievable score starting from the episode's initial node, representing the maximum number of other nodes it can control, discover, or disrupt.
- Across the studies, we follow **best practices** for RL evaluation, including hyperparameter optimization, static seeds, and reporting BCIs with a confidence level $\alpha = 0.95$ for statistical reliability from a limited number of runs [246].

The embedding spaces for the experiments are constructed as follows:

- The dimensionality of the **node embeddings** (GNN module) was set to $p = 64$, with both this dimension and the GAE architecture and its hyperparameters tuned through hyperparameter optimization. For constructing graph-level embeddings, the node embeddings were aggregated using a combination of *mean*, *min*, and *max* pooling operations.

- For the generation of **vulnerability embeddings**, pre-trained LMs are used and varied between experiments. These include four general-purpose LMs (BERT, DistilBERT, GPT-2, RoBERTa) and four cybersecurity-specific LMs (CySecBERT, SecBERT, SecRoBERTa, SecureBERT). All of these models have the same output layer size, producing 768-dimensional embeddings, fixing the vector dimensionality to $q = 768$. In each RL agent training run, a different pre-trained and frozen LM is selected to generate embeddings, ensuring that we compare agents across multiple vulnerability mapping functions.
- The *nearest-neighbor* selection method is used as the proximity-based **decoding strategy**, employing *cosine distance* to address the curse of dimensionality in the high-dimensional action space.

6.5.2 Scalability study

This study assesses the ability of discrete and continuous spaces to create agents that can learn from scenarios with increasing node sizes and vulnerabilities. Increasing the number of nodes and vulnerabilities expands the agent's available options while complicating the mappings between spaces. In discrete methods, this leads to larger observation/action vectors and necessitates an NN with larger input and output layers. In contrast, continuous spaces maintain fixed input and output vectors and NN layers, with the complexity manifesting instead as a higher density of points within the spaces, as explained in Section 6.4.3.

Setup. A total of 300 synthetic scenarios are generated. These comprise three scenarios, one for each threat model, for each of the 100 unique combinations of node and vulnerability counts, both varying between 5 and 250.
The proposed *continuous latent* spaces used by the projection-based agent and the *discrete local* spaces are compared with the *discrete global* formulation, which has been predominantly adopted in the literature. Rather than reproducing the specific agent architectures used in previous studies, the comparison focuses on the representational spaces themselves. To this end, each space is paired with its best-performing DRL algorithm and corresponding hyperparameter configuration, ensuring that the evaluation reflects the maximum performance attainable within each formulation. For the discrete approaches, action masking is applied over the vulnerability–outcome dimensions, such that output neurons correspond only to vulnerabilities present on the selected nodes and their associated outcomes. This prevents the agent from allocating probability mass to invalid actions while preserving the underlying action-space formulation.

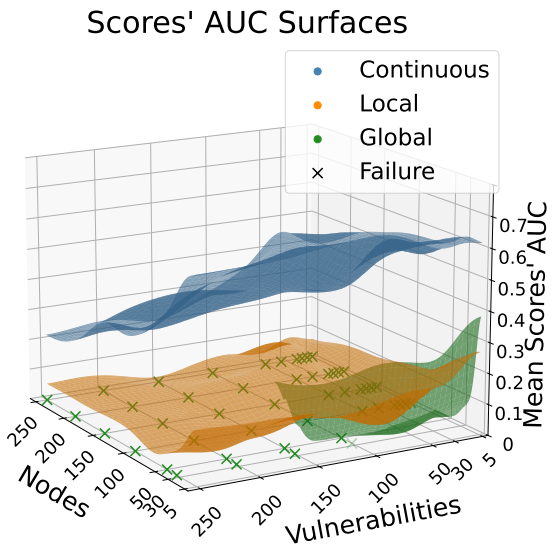


Figure 6.4: The normalized average AUC score for agents learning from different spaces, with the curve depicting score progress.

For each representation space, 300 agents are trained for 250,000 iterations, corresponding to all combinations of graph size, vulnerability set size, and threat model, with the episode-length coefficient fixed to (K = 30).

Performance is evaluated using the Area Under the Curve (AUC) of the goal score over the course of training. The AUC is normalized with respect to that of an ideal agent that immediately converges to the optimal policy. This metric jointly captures learning speed and final performance, providing a single measure of learning efficiency for each representation space.

Analysis. Figure 6.4 aggregates the scores across goals per scenario size and shows the trend of the training AUC scores when the scenario sizes increase. The results show that discrete agents struggle as the number of nodes and vulnerabilities increases, since the observation and action vector sizes—and consequently the DNN input and output layers—scale with these factors. While discrete global agents initially achieve higher scores, they plateau sooner than discrete local agents, and their training experiences failures due to the excessively large resulting DNN model needed to process the corresponding vectors, which exceeds the capacity of the benchmark machine used for training[3]. In contrast, the discrete local agents support all experiments but achieve lower scores due to the limited visibility on a subset of nodes. On the other hand, projection agents over latent continuous spaces demonstrate faster, more efficient learning, with improved scalability and a smaller performance decline, achieving an average relative improvement of 9.3x over discrete methods, 7.4x over the discrete local approach, and 11.2x over the discrete global approach.
Nevertheless, discrete local and global approaches remain effective solutions in constrained settings where inference efficiency is prioritized over scenario scalability. As they avoid the additional cost of GNN representation learning, LM-based embeddings, and latent action-space search, they are suitable when training and evaluation follow the same scenario distribution, environment sizes are limited, and the action space remains manageable. In particular, local approaches provide a practical trade-off for medium-sized scenarios under these assumptions. However, for larger, more diverse, or unseen scenarios, projection-based approaches become increasingly advantageous by trading additional computational cost for improved scalability and generalization.

6.5.3 Generalization study

We conducted generalization experiments to evaluate the ability of DRL algorithms to train projection-based agents that can transfer knowledge across scenarios when operating in latent spaces. In particular, we investigate the extent to which policies trained on simpler scenarios can generalize to more complex ones. This analysis is restricted to the proposed latent-space formulation, as the scalability study presented in Section 7.6.6 showed it to be the only approach capable of effectively handling the larger-scale scenarios required for effective use of heuristic learning.

Setup. 500 synthetic scenarios are generated using *domain randomization* with varying graph parameters (e.g., number of nodes in $[10, 100]$), cyber-terrain features (e.g., firewall rule probabilities), and vulnerability allocations. These scenarios are divided into training (60%), validation (20%), and testing (20%) sets, based on an aggregated complexity metric derived from the randomized features of each scenario (see Appendix C.2 for details). In this way, scenarios are split by increasing difficulty: training uses simpler cases, validation uses moderate ones, and testing uses the most complex ones. Table 6.2 summarizes the distributional shifts in key scenario features observed in the validation and test sets relative to the training set.

[3] x86_64 AMD Ryzen 9 5900X 12-Core Processor, 125 GB RAM, Ubuntu 22.04.1

Table 6.2: Relative increase or decrease of selected scenario characteristics in the validation and test sets with respect to the training set.

Feature	Validation	Test
Average number of nodes per scenario	1.21×	1.29×
Unique service combinations across hosts	1.04×	1.17×
Average firewall rule probability	1.11×	1.18×
Probability of initial user-level access on first node access	1.09×	1.16×
Probability of partial visibility on first node access	1.15×	1.23×
Probability of data presence on a node	0.87×	0.78×
Probability of service startup/shutdown	1.29×	1.42×
Average percentage of nodes discoverable across all starter nodes	0.92×	0.88×
Average percentage of nodes reachable via lateral movement across all starter nodes	0.88×	0.82×
Average percentage of nodes vulnerable to disruption across all starter nodes	0.86×	0.80×

All RL algorithms from Stable-Baselines3 [224] supporting continuous action spaces are evaluated, namely PPO, TRPO [262], TD3 [263], TQC [264], SAC [265], Recurrent PPO [266], and A2C [265], together with a heuristic that selects *random valid* points from the action space at each timestep. The evaluated methods include both *on-policy* algorithms (PPO, TRPO, Recurrent PPO, and A2C) and *off-policy* algorithms (TD3, TQC, and SAC). Since these two families rely on fundamentally different learning paradigms, evaluating both is important for identifying the most suitable RL heuristic strategy for strong performance on the critical attack-path discovery problem. On-policy methods update their policies using trajectories generated by the current policy, whereas off-policy methods can additionally leverage previously collected experiences stored in replay buffers [267, 268].
Each algorithm is evaluated using its optimal hyperparameters, determined through hyperparameter optimization (see Appendix A.3 for details), and is used to train 8 agents per goal for 300k iterations with $K = 10$ (limited action budget per episode).
A *random switching strategy* is employed to ensure that the simulator switches scenarios and starting nodes at the beginning of each episode, throughout training, validation, and testing. This approach guarantees that the agent encounters a wide range of configurations during training and is evaluated on multiple configurations during The best checkpoint of each training run is determined doing *early stopping* on the validation set (scenarios of moderate complexity) and it is used on the test set (most complex scenarios) to generate 100 episodes, generating overall 24 agents $\times\, 8$ algorithms $\times\, 100$ episodes = 19,200 trajectories used for comparison, with the goal score used as trajectory metric (e.g., % of owned nodes with ROOT privilege for *control*).

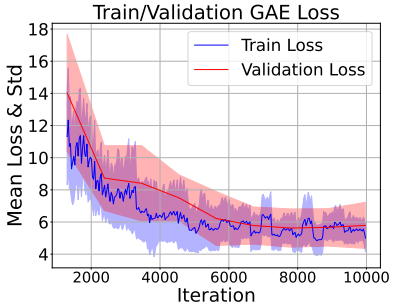


Figure 6.5: GAE reconstruction training and validation loss.

GAE analysis. The training and validation curves of the GAE on the generalization scenario set are reported in Figure 6.5. The close agreement between the training and valida-

tion losses indicates that the embedding module generalizes
well to unseen graph instances with respect to the reconstruction objective. The model checkpoint achieving the lowest validation loss is selected and subsequently used to generate node embeddings. These embeddings encode the structural information of the input graphs into a lower-dimensional representation, providing a more compact and continuous description of the original graph structure. The resulting node embeddings form the basis of the latent continuous observation and action spaces employed by the projection-based agent.

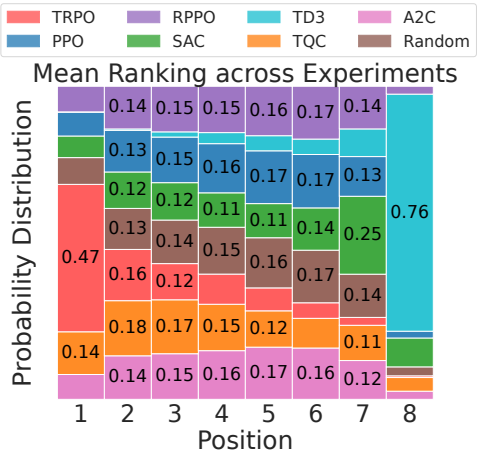


Figure 6.6: Rank probability matrix highlighting the likelihood of each DRL algorithm to achieve a certain rank across goals.

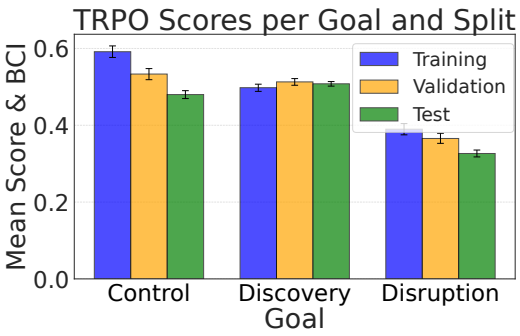


Figure 6.7: TRPO agent performance across different attack goals and scenario sets.

Agent analysis. Figure 6.6 moves beyond traditional binary comparisons, presenting a summary of the test set goal scores with a probability rank matrix indicating the likelihood of each RL algorithm achieving a specific rank, averaged across all goal experiments. The results indicate that TRPO outperforms all other algorithms, while TD3 decisively ranks the lowest. It is noteworthy that several other algorithms perform at a level comparable to or marginally better than the *random valid* heuristic, demonstrating that not all algorithms can learn meaningful policies in continuous spaces.
The TRPO algorithm, recognized as the top performer, is used to analyze scores by retesting its agents across different split sets, given that they were only exposed to the training set consisting of simpler scenarios. The results in Figure 6.7 show that scenario complexity impacts the *control* and *disruption* goals, while *discovery* maintains more consistent performance across different scenario sets.
summarize generalization performance, we compute the mean ratio of test-set scores (representing the most complex scenarios) to training-set scores (corresponding to the simplest scenarios encountered during training). On average, TRPO agents achieve an 89% test-to-training score ratio, demonstrating effective generalization to more complex scenarios.

6.5.4 Dynamicity study

TRPO has been used to analyze generalization under dynamic conditions, a critical requirement for deployment in online environments [249]. To this end, we introduce a *random-events probabilistic defender* into the test-set scenarios, while maintaining the same experimental study as in

Section 6.5.3.
The defender agent intervenes on each node with probability p_n after each attacker action, and we visualize the resulting effect on the episode scores for the attacker agent. When intervening, the defender randomly selects one of four actions with equal probability, applied to a randomly chosen service or firewall rule on the node:

- **Starting a service** (potentially beneficial)
- **Stopping a service** (potentially detrimental)
- **Allowing a firewall rule** (potentially beneficial)
- **Blocking a firewall rule** (potentially detrimental)

These defender actions alter the network services and firewalls, making certain vulnerabilities exploitable or not.
Consequently, the observation vector changes (as node embeddings vary), and the set of valid points in the action space shifts accordingly. Whether the defender's action benefits or harms the attacker depends on which action is randomly selected. The results (Figure 6.8) indicate that agents trained for *discovery* and *disruption* goals experience a slight performance decline (i.e., a reduction in the average amount of information discovered and nodes disrupted), but still generalize effectively, even with high intervention probabilities (up to 50% of nodes that undergo at least a change after each attacker agent action). In contrast, agents trained for the *control* goal show improved scores by effectively leveraging beneficial changes to gain control over more nodes, while also exhibiting resilience to detrimental changes.

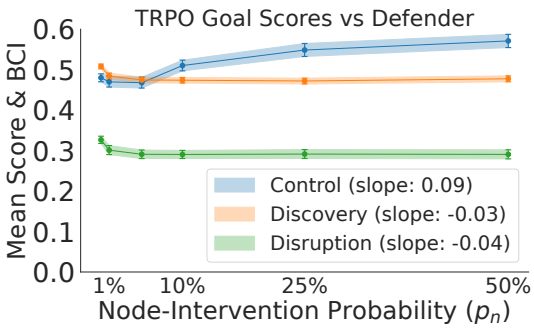


Figure 6.8: TRPO agent performance on the test set under varying defender intervention probabilities.

6.6 Deployment-oriented study

Although the proposed methodology has been evaluated only in synthetic scenarios, the same pipeline can be extended to real-world digital twins by incorporating vulnerability scan results and constructing a virtual replica of the target network. While the transition dynamics and security objectives remain simulated, they are grounded in a realistic representation of the vulnerabilities and characteristics of the network nodes.
The deployment of agents within these replicas follows the following steps:

1. A virtual replica with the same structure overlays the application scenario and is optionally updated with periodic vulnerability *scan snapshots*.
2. *Missing* features and *information* (e.g., data presence) are either provided by the administrator or probabilistically varied to represent various scenarios.
3. *Continuous latent spaces* are constructed using a set of mapping functions (GNN and LM of choice) to generate embeddings based on the graph and vulnerability descriptions.
4. Agents can either be *retrained* on the virtual replicas and then deployed in them for the prediction of paths, or trained on synthetic scenarios and then only *deployed* to the virtual replicas. The deployment involves running the agents for a configurable number of episodes

N, with a budget of actions (episode-length coefficient K) from selected starter nodes, producing critical paths.

5. The generated critical paths—logs detailing sequences of source, target, vulnerability, and outcome—can help administrators *allocate defenses* based on research questions (e.g., identifying frequently targeted nodes), with the efficacy of these defenses measured by the associated episode scores.
6. (*Optional*) The *pipeline* can be *re-iterated* to evaluate defense strategies by re-deploying the agents after the modifications and tracking any score deterioration.

We validate this pipeline by using virtual replicas derived from real-world and emulated network scans of organizational scenarios. Additionally, we use this experiment to evaluate the research question: *Can agents trained exclusively on synthetic scenarios generalize to realistic scenarios, and is their performance comparable to that of agents trained directly in those real-world inspired networks?*

Setup. Two scenarios of 100 nodes: one derived from a scan of an emulated network (350 vulnerabilities, adapted from [126]) and one from a real-world network (784 vulnerabilities), with distinct vulnerability sets and distributions (as represented in Figure 6.9). We compare the following solutions:

1. Projection-based TRPO agents *retrained* specifically on each application scenario for 300k iterations and then re-tested in the corresponding scenario seen during training;
2. Projection-based TRPO agents trained solely on *synthetic* scenarios using domain randomization (as done in Section 6.5.3) and then tested zero-shot on the unseen application scenarios;
3. A heuristic strategy prioritizing vulnerabilities with the highest *impact scores* (maximize damages);
4. A heuristic strategy prioritizing *exploitability scores* (maximize ease of exploitation).

Each trained agent and heuristic is used to generate $N = 240$ trajectories (episodes) for each of the application scenarios using $K = 30$, allowing longer episodes and monitoring near-to-maximum impact. Average goal scores from testing episodes are used to assess the criticality of attack paths and to determine whether an agent or heuristic is more effective at generating these paths for a given goal.

Analysis. Results in Figure 6.9 show that both projection agents (retrained and trained solely on synthetic scenarios) outperform heuristics on both the real-world and emulated scans for the *control* and *discovery* goals. Heuristics and agents show no clear winner for the *disruption* goal, underscoring the challenge for agents of learning disruption strategies. This is further complicated by the use of heuristics that are more effective in this case, as vulnerabilities with high exploitability and impact scores typically result in disruption.
As expected, projection agents retrained on the specific application scenarios outperform those trained solely on synthetic data. However, computing the average ratio of scores between synthetic-only and retrained agents shows that synthetic-only agents achieve 75% of the scores of retrained agents (ratio of scores)—84% in the emulated scenario and 65% in the real-world scenario. This validates the generalization of projection agents over continuous latent spaces, showing that those trained on synthetic scenarios can achieve competitive results, with retrained agents serving as the upper bound. Future research may bridge the sim-to-real gap when synthetic-trained agents match the performance of those trained on specific application scenarios. While retraining on application-specific data ensures high-performance agents, it may not be

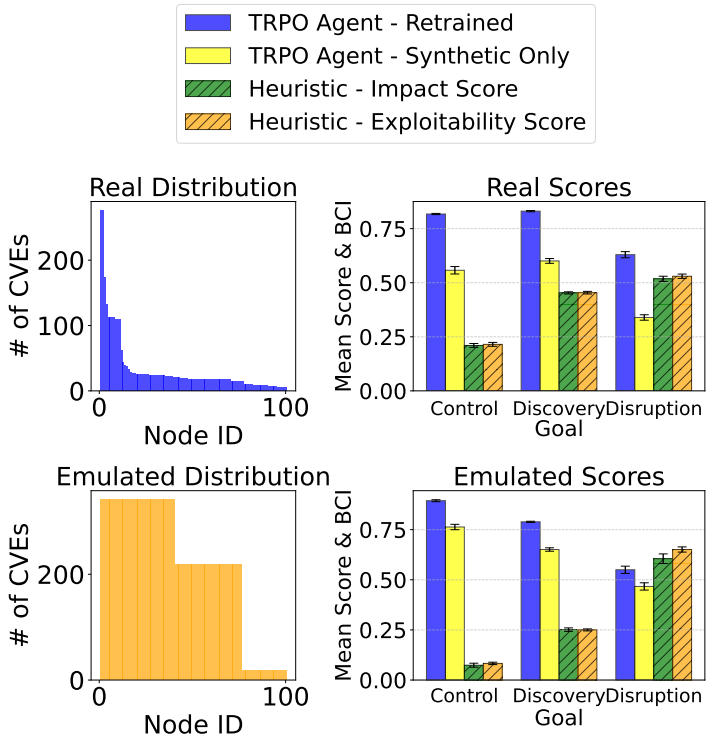


Figure 6.9: TRPO agent, retrained or not, tested against heuristics on both emulated and real-world replica environments.

feasible in privacy-sensitive contexts where vulnerability scans are restricted. In such cases, agents trained on synthetic data can, in fact, provide a viable alternative.
The total execution time of the agent on the replica environments depends on several factors, including the episode-length coefficient K and the number of episodes N. For instance, on a benchmark personal computer[4], each iteration (action) of the projection agent on the replica of the real-world network with 100 nodes and 784 vulnerabilities takes approximately 0.0215 $\pm$ 0.0016 seconds. Using $K = 10$ and $N = 10$ episodes, the agent would take approximately 3 minutes and 35 seconds on average for the entire run.

6.7 Summary

This chapter presented an extensive evaluation of the proposed RL-GCO framework on the critical attack path discovery problem. Building upon the POMDP formulation introduced in this thesis, this work extended the concepts of continuous latent observation and action spaces, along with projection-based agents operating within these spaces, to the domain of RL-driven critical attack path discovery. By decoupling the agent's observation and action spaces' dimensionality from specific network topologies and predefined vulnerability sets, the proposed formulation enables policies to operate across heterogeneous scenarios without retraining for each new one.
The proposed approach was evaluated on 800 scenarios spanning a wide range of network sizes and vulnerability distributions, including 829 real-world vulnerabilities. The results demonstrated a 9.3x improvement in scalability compared to agents operating in discrete spaces, while

[4]x86_64 AMD Ryzen 7 PRO 7840U with Radeon 780M Graphics, 30 GB RAM, Ubuntu 22.04.1

achieving an average generalization score of 89% when transferring from simpler training environments to more complex scenarios. A final deployment-oriented study further investigated the transferability of policies trained in simulation to environments derived from real-world and emulated vulnerability scans. In these settings, the agents achieved, on average, 75% of the performance obtained by policies trained directly on the target scans, providing evidence of effective knowledge transfer across environments.
Beyond scalability and transferability, the learned policies consistently outperformed widely used manually designed heuristics under two commonly adopted threat models. Overall, the results demonstrate that combining topology- and vulnerability-invariant latent representations with projection-based action selection enables the learning of effective and scalable attack-path prioritization strategies while simultaneously addressing the challenges of generalization and adaptation that have traditionally limited RL approaches in this domain.

Link to the thesis objectives

This chapter connected several core contributions introduced throughout the thesis by reusing the RL-GCO solution proposed in Chapter 5 within the simulator, leveraging the POMDP defined and implemented in Chapter 3, itself further enabled by the LM-based multi-label classifier introduced in Chapter 4.
The chapter demonstrated the scalability and generalization capabilities of the proposed approach for the critical attack path discovery task. Finally, Chapter 7 concludes this line of work by transferring the attacker-agent framework developed here to the VMP problem, a networking application where security considerations are becoming increasingly important.

Chapter 7

Multi-agent RL framework for secure virtual machine placement

Summary: *This chapter presents an application of the critical attack path discovery framework to the VMP problem, a networking optimization task of major interest in the literature, in which security is increasingly incorporated as an optimization objective. To address this problem, we propose a MARL framework in which an attacker agent identifies critical attack paths, while a defender agent leverages this information to dynamically reconfigure VM placement within a virtualized infrastructure.*

Contents

7.1 Introduction . . . 128
7.2 Problem formulation . . . 129
7.3 Reinforcement learning extensions . . . 130
7.3.1 Solving multi-objective optimization with reinforcement learning . 130
7.3.2 Multi-agent reinforcement learning . . . 131
7.4 Existing solutions . . . 131
7.4.1 Modeling approaches . . . 131
7.4.2 Solutions for placement . . . 132
7.5 Methodology . . . 133
7.5.1 Graph representation . . . 133
7.5.2 Defender agent . . . 134
7.5.3 Latent spaces for the defender agent . . . 136
7.5.4 Simulation environment . . . 137
7.6 Experiments . . . 138
7.6.1 Experimental setup . . . 138
7.6.2 Threat model study . . . 140
7.6.3 Security–performance trade-off study . . . 141
7.6.4 Impact of cloud vulnerability landscapes . . . 141
7.6.5 Impact of initial allocation strategies . . . 142
7.6.6 Scalability study . . . 143
7.6.7 Replica pool study . . . 144
7.7 Deployment pipeline . . . 145
7.8 Summary . . . 146

7.1 Introduction

This chapter extends the study of attack-surface estimation through critical attack paths to another important GCO problem: secure VMP. By leveraging the RL-based attacker modeling and POMDP formulations developed in previous chapters, it introduces a more realistic and threat-aware approach to this optimization problem. More broadly, the proposed framework illustrates how learned attacker strategies can be used to guide security-oriented optimization policies, enabling defender decisions that account for adaptive adversarial behavior and ultimately reduce the attack surface of the network configuration.

Virtualization has become a core component of modern computing infrastructures, including enterprise data centers and cloud platforms, due to its ability to improve resource utilization while reducing hardware, maintenance, and energy costs [269]. In such environments, multiple VMs share the same PM through a hypervisor layer responsible for resource allocation and isolation [270]. The problem of determining how VMs should be assigned to PMs is referred to as VMP [271], a CO problem well-known to be NP-hard [272].

As virtualized infrastructures continue to increase in scale and heterogeneity, VMP becomes increasingly challenging. Existing approaches typically aim to optimize infrastructure efficiency through aggressive VM *colocation* strategies [273], where as many VMs as possible are consolidated onto a reduced number of PMs in order to minimize resource consumption and energy usage [274]. However, while such strategies improve operational efficiency, they also increase the attack surface by amplifying co-residency risks and the potential exploitation of vulnerabilities affecting the virtualization layer.

Consequently, incorporating security considerations into VMP naturally transforms the problem into a Multi-Objective Optimization Problem (MOOP), where objectives such as security, performance, resource utilization, and energy efficiency often conflict with one another [275]. In practice, stronger isolation policies may improve security while simultaneously degrading consolidation efficiency and increasing infrastructure costs.

Although recent studies increasingly consider security-aware VMP formulations, most existing approaches rely on simplified representations of the attack surface, typically based on static vulnerability metrics such as CVSS scores [276, 277]. While computationally convenient, these metrics fail to capture vulnerability interdependencies and do not explicitly model attacker behavior under realistic threat models. As a result, they provide only a limited approximation of the actual security risks associated with a placement configuration.

To address these limitations, this chapter extends the POMDP attack modeling framework of this thesis to the VMP setting. Instead of relying solely on static vulnerability scores, the security objective is estimated through RL-based attacker agents capable of identifying critical multi-stage attack paths under different threat models. Within this framework, attacker agents evaluate the attack surface induced by a placement configuration, while defender agents iteratively optimize VM placements to reduce attack exposure while preserving infrastructure performance objectives.

Unlike dynamic VM *consolidation* approaches, which continuously migrate VMs during runtime in response to workload or threat variations [278, 279], the proposed framework focuses on computing stable placement configurations in advance, reducing attack opportunities while avoiding the operational overhead and instability associated with frequent live migrations [280]. Building upon the contributions presented throughout this thesis, this chapter therefore investigates how MARL can be used to jointly model attacker and defender CO strategies for a secure and efficient VMP in large-scale virtualized infrastructures.

Contributions

The main contributions of this chapter, aimed at enabling the learning of effective heuristics that jointly consider security and operational efficiency in the VMP problem, are the following:

- A **MOOP formulation** combining traditional infrastructure performance metrics with a realistic attack-surface estimation based on sequential critical attack path modeling.
- A **MARL framework** enabling the training of scalable and generalizable attacker–defender agents for secure VMP under adaptable threat models.
- An extension of *C-CyberBattleSim*, redesigned to model virtualization environments and support sequential turn-based attacker–defender interactions for VMP optimization.
- A comprehensive **experimental evaluation**, including comparisons between RL algorithms, analyses across multiple threat models, exploration of security–performance trade-offs, evaluation of vulnerability landscape effects, and assessment of scalability and deployment feasibility.

7.2 Problem formulation

We consider a general Infrastructure-as-a-Service model, in which cloud providers offer on-demand computational resources as virtualized instances to multiple tenants. Tenants deploy applications by requesting VMs, which must be efficiently mapped onto PMs. This mapping is addressed through VMP, a CO task often formulated as a variant of the bin-packing problem [281]. The objective is to determine a (sub-)optimal allocation of VMs to PMs while satisfying system constraints in a virtualized environment.

Let $\{v_1, v_2, \ldots, v_p\}$ represent the set of VMs, $\{t_1, ..., t_q\}$ represent a set of tenants owning the VMs, and $\{p_1, p_2, \ldots, p_r\}$ represent the set of PMs in the system.

The problem is typically formalized by constructing an abstract model of VMs and PMs, often represented as a graph $G = (V, E)$, where nodes V correspond to VMs and PMs—each described by a feature vector capturing relevant characteristics (e.g., resource requirements for VMs)—and edges E represent allocations or other relevant relationships, such as communication between PMs.

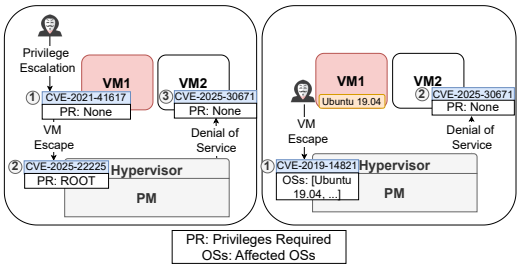


Figure 7.1: Overview of the VMP attack surface, illustrating VM escape through hypervisor vulnerabilities and OS-specific configurations.

This graph-based model should be sufficiently detailed to accurately represent the system's structure and all the aspects to be considered for the optimization, reflecting real-world dynamics and relationships. In particular, in MOOPs, the goal is not to find a single optimal solution but rather a set of trade-off solutions, known as Pareto optimal solutions [282], in which no objective can be improved without worsening another. When multiple VMs are co-located on the same PM and share underlying resources, compromising one VM or exploiting VM escape vulnerabilities can expose other

co-located VMs, making security a critical consideration in VMP strategies. A malicious user may deploy or access a VM and exploit vulnerabilities at both the VM and hypervisor levels to attack one or more target VMs [283, 284, 285].
Figure 7.1 presents two examples of critical vulnerability chains in a sample virtualized infrastructure, highlighting their interconnections and potential impact on the attack surface. In the first example, an attacker exploits a vulnerability on a VM to escalate privileges and subsequently leverages a hypervisor-level vulnerability—requiring ROOT access on the compromised VM—to target a co-located VM. Patching VM privilege-escalation vulnerabilities could have blocked this lateral movement. The second example illustrates that only certain Guest OSs may be susceptible to specific hypervisor vulnerabilities, potentially enabling lateral attacks to co-located VMs. This shows that selecting a different OS/hypervisor combination during the placement decision could have prevented this attack. These examples emphasize the importance of understanding vulnerability chains and the impact on the attack surface.

7.3 Reinforcement learning extensions

The secure VMP problem considered in this chapter introduces additional challenges beyond the standard single-agent RL setting studied previously in this thesis. In particular, the problem simultaneously involves multiple potentially conflicting optimization objectives and the interaction among multiple decision-making entities with adversarial goals. To address these aspects, this section presents two important extensions of traditional RL: the adaptation of RL to MOOPs, and MARL.

7.3.1 Solving multi-objective optimization with reinforcement learning

Applying RL to MOOPs requires extending the traditional RL formulation to account for multiple objectives in the optimization. In the context of secure VMP, for example, objectives such as minimizing energy consumption, maximizing resource utilization, reducing operational cost, and minimizing the attack surface cannot generally be optimized simultaneously without trade-offs. Standard RL algorithms, however, are designed to optimize a single scalar reward signal. Consequently, when RL is applied to MOOPs, the multiple objectives must first be transformed into a single scalar optimization target.
Among the various *scalarization* strategies proposed in the literature, one of the most widely adopted approaches is the *weighted-sum* method [286]. In this formulation, each objective contributes to the final reward according to a predefined importance coefficient. The resulting scalar reward is expressed as in Equation 7.1.

$$\mathbf{f} = \big(f_1, \ldots, f_k\big) \;\Rightarrow\; r(s, a, s') = \sum_{i=1}^{k} w_i f_i(s, a, s') \tag{7.1}$$

$f_i(s, a, s')$ denotes the value of the i-th objective evaluated over the transition from state s to state s' after taking action a, and w_i represents the corresponding weighting coefficient defining its relative importance.
The weighted-sum scalarization is particularly attractive due to its simplicity, computational efficiency, and compatibility with existing RL algorithms. Furthermore, by varying the weighting coefficients w_i, different trade-offs between objectives can be explored, enabling the approximation of different regions of the Pareto front. This property is especially relevant in security-aware

optimization settings such as VMP, where infrastructure administrators may prioritize different trade-offs between security and operational efficiency depending on deployment requirements. However, weighted scalarization introduces challenges, as the choice of weights can bias the learned policy and objectives, with different scales may dominate the reward signal. Consequently, careful reward design and normalization are essential.

7.3.2 Multi-agent reinforcement learning

MARL extends the traditional RL framework to environments involving multiple interacting agents. Instead of learning a single policy, MARL aims to learn a set of policies $(\pi_1, \pi_2, \ldots, \pi_N)$, one for each agent, while accounting for the fact that the environment dynamics are simultaneously influenced by the actions of multiple decision-makers.
Depending on the problem formulation, interactions between agents may be *cooperative*, *competitive*, or *mixed*. In cooperative settings, agents share a common objective and collaborate to maximize a joint reward. In competitive settings, agents pursue opposing objectives, creating adversarial dynamics similar to game-theoretic formulations. Mixed settings combine both collaborative and adversarial interactions.
The secure VMP problem naturally fits within a competitive MARL formulation. In this setting, defender agents aim to optimize VMP decisions to reduce the attack surface while preserving infrastructure performance, whereas attacker agents attempt to identify and exploit critical attack paths within the resulting placement configuration. Consequently, the optimization process becomes inherently adversarial, as placement decisions directly influence attackers' opportunities, and attackers' behavior influences the defender's optimization strategy.
More specifically, the framework considered in this chapter follows a *turn-based* interaction model. Rather than acting simultaneously, agents operate sequentially: the defender first modifies the VMP configuration, after which the attacker reassesses the environment and attempts to exploit the resulting attack surface. This sequential interaction closely reflects realistic preventive security dynamics, where defenders periodically deploy infrastructure changes to prevent critical attacks on the target network.

7.4 Existing solutions

This section outlines two key axes for comparing our approach with related work: the way the problem is modeled and the solution algorithms used.

7.4.1 Modeling approaches

Existing research on VMP primarily varies in how the optimization problem is formulated, particularly in the choice of objective functions f_i, which typically drive the definition of other modeling components (e.g., variables and constraints).
Recent studies have incorporated the attack surface as an objective, but often rely on oversimplified representations, limiting realism and effectiveness. Table 7.1 qualitatively compares selected secure VMP approaches with respect to attack surface modeling and performance objectives. Common strategies to derive this surface include manual heuristic (MH) methods [287, 288], which estimate sub-optimal solutions based on manual design (e.g., approximating side-channel risks); user-requirement (UR) approaches [279, 278], which prioritize the security priority of VMs based on user-defined importance; and CVSS-based methods [277], which assign static scores based on exploitability and impact but ignore interdependencies of vulnerabilities.

Study	Attack Surface	VM Traffic / Latency	Resources Utilization	Energy Consumption	VM Importance	Outages	Migration Cost	VM Isolation	Replicas
[287]	MH		✓	✓					
[288]	MH	✓	✓	✓			✓		
[289]	MH		✓	✓					
[279]	UR		✓	✓		✓			
[278]	UR			✓		✓			
[290]	CVSS			✓					
[277]	CVSS		✓		✓				
[275]	CVSS	✓	✓						
[291]	CVSS								
Ours	**Attack Paths**	✓	✓	✓	✓	✓	✓	✓	✓

Table 7.1: Comparison of security-related and performance-related objectives in existing studies for multi-objective VMP.

However, real-world attacks are sequential and depend on multiple factors, including vulnerability types and locations, threat models, VM co-location, and network policies (e.g., firewall rules). Assigning static security values to individual VMs or vulnerabilities is therefore insufficient, and manually designing accurate, context-aware heuristics is highly challenging. To address this limitation, we propose leveraging critical vulnerability propagation paths inherent in the VMP configuration, which naturally incorporate key aspects such as network topology, VM co-location, and interactions between vulnerabilities. These paths provide a more realistic estimate of the attack surface by modeling how an attacker can propagate through the system by chaining vulnerabilities.
As shown in the last row of Table 7.1, our approach proposes a critical attack-path method to model the attack surface while incorporating all key performance metrics from prior work. Additionally, we introduce two novel (optional) extensions:

- *Placement with isolation:* Our model supports the application of isolation techniques, enabling mitigation of certain hypervisor-level vulnerabilities and providing control over which isolation measures to apply.
- *Replica deployment:* The defender agent can opt to deploy VM replicas to enhance availability and user satisfaction, at the cost of increased resource usage and potentially a larger attack surface.

7.4.2 Solutions for placement

Existing approaches for solving the VMP problem predominantly rely on classical one-shot heuristics. In particular, many security-aware VMP formulations employ non-autoregressive meta-heuristic strategies, such as Genetic Algorithms [288], Firefly Algorithms [287], or other population-based methods. These approaches typically operate by directly transforming an initial placement configuration into a candidate final solution via global optimization procedures, without explicitly modeling the sequential nature of the placement process.
While such approaches can effectively explore large combinatorial search spaces, they generally treat placement decisions as static optimization variables rather than as a sequence of dependent actions. In practice, however, VMP is inherently sequential. Placement configurations emerge from a sequence of allocation, migration, consolidation, or isolation decisions, in which each

intermediate action modifies the infrastructure's future state and influences the set of subsequent feasible decisions. RL naturally aligns with this sequential perspective by modeling optimization as an autoregressive decision-making process [209]. Instead of directly generating a complete solution in a single optimization step, RL agents progressively construct placement configurations through a sequence of actions, learning policies that account for the long-term consequences of placement decisions.
Moreover, as discussed in previous chapters, RL-based attacker modeling enables fast inference of critical attack paths compared to attack-graph approaches that require costly graph generation and symbolic analysis, even if at the expense of exact optimality guarantees. This advantage is particularly important during defender training, where security metrics must be evaluated repeatedly across many placement configurations to support the millions of interactions typically required by the RL paradigm.
Building on this sequential formulation, this chapter proposes a unified MARL framework in which RL is employed simultaneously for attacker modeling and defender optimization. Within this framework, attacker agents evaluate placement configurations by exploring multi-stage attack paths under specific threat models, while defender agents iteratively modify VM placements to minimize attack exposure while preserving infrastructure efficiency objectives.

7.5 Methodology

This section presents the methodology of this study, covering the modeling of the VMP problem, the design of a multi-agent solution that captures the dynamics of the defender agent, and the procedure for generating synthetic scenarios.

7.5.1 Graph representation

The proposed VMP formulation extends the abstraction introduced in Section 7.2 by enriching the graph representation with the performance-related aspects summarized in Table 7.1, as well as features required to estimate the attack surface. We model the infrastructure as an overlay heterogeneous graph $G = (V, E)$, where V contains both PM and VM nodes. The node feature vectors include the following elements:

- **VM node:**
  - **Performance:** resources required to operate (e.g., memory, CPU, I/O, network, file-system capacity)
  - **Security:** incorporate the detected guest OS and service vulnerabilities using their CVEs
  - **Other:** tenant IDs, firewall rules, system status indicators, data presence, importance-level metrics
- **PM node:**
  - **Performance:** physical resources available (memory, CPU, I/O, network, file-system capacity)
  - **Security:** incorporate the detected hypervisor vulnerabilities using their corresponding CVEs
  - **Other:** firewall rules, system status indicators, importance-level metrics

CVEs are incorporated by converting their description to numerical representations using LM

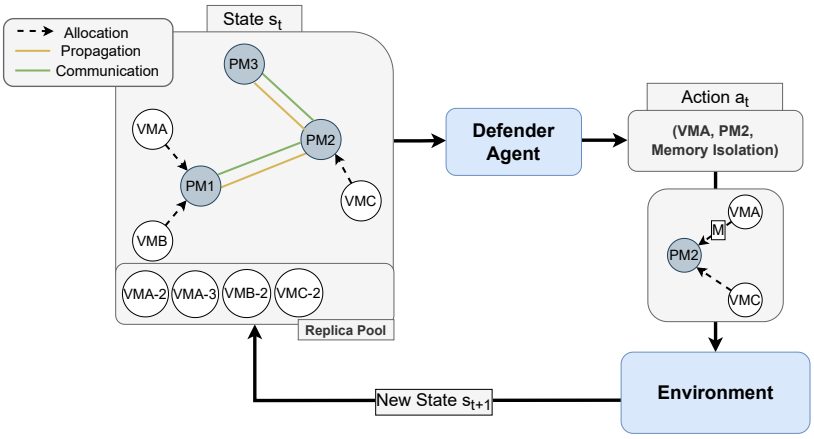


Figure 7.2: The MDP for the defender agent trained to produce reconfiguration actions.

techniques to capture their semantics, as explained in Section 6.4.1.
The edges E in the graph are instead defined to encode three distinct relationships between VMs and PMs:

- **Allocation edges** ($E_{\text{allocation}}$): Undirected edges between VMs and PMs, indicating existing allocations, including an attribute representing the isolation level applied.
- **Communication edges** ($E_{\text{communication}}$): Undirected edges between PMs representing inter-host communication, annotated with an attribute specifying the bandwidth.
- **Propagation edges** ($E_{\text{propagation}}$): Directed edges between PM nodes that model the propagation of potential attacks. Specifically, these edges represent the potential for an attacker who compromises a VM on one PM to exploit remote vulnerabilities in VMs on other PMs.

7.5.2 Defender agent

We model the defender agent as an RL agent trained to leverage an initial VMP configuration and perform a sequential reconfiguration to balance multiple optimization objectives. Each episode (Figure 7.2) begins with an initial allocation of VMs to PMs and terminates after a fixed duration. Unlike the attacker, the defender operates within the scenario's fully observable network graph G, which is assumed to be managed by the cloud resource manager and provides complete network visibility.

Observation space. The defender's observation consists of the complete scenario graph G, including $E_{\text{allocation}}$, $E_{\text{communication}}$, and $E_{\text{propagation}}$, together with a set of optionally deployable VM replicas. These replicas are represented as nodes in a separate initially disconnected graph and can be placed by the defender within the network.

Action space. The action space consists of all (*source VM*, *destination*, *isolation*) tuples, where *source* can be any VM or a non-allocated replica, *destination* can be any available PM or the *replica pool* if a source VM is to be de-allocated, and *isolation* is chosen from *None*, *Memory*, *CPU*, *I/O*, *Network*, *File-System*.
Choosing an isolation level sets the constraints on an attacker targeting the VM. For example, *memory* isolation prevents exploits of hypervisor memory vulnerabilities, such as those caused by

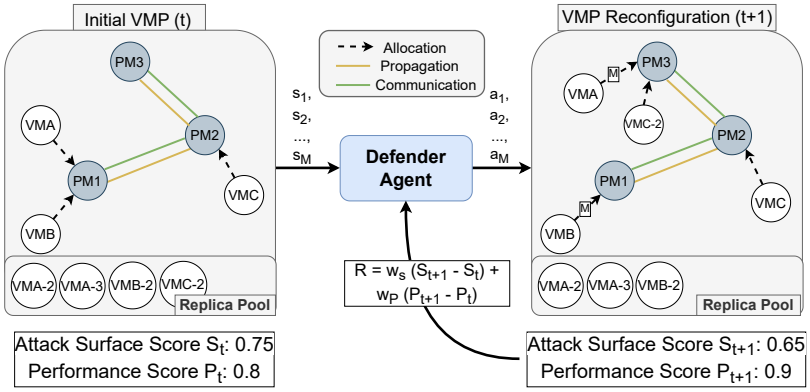


Figure 7.3: The defender receives an episodic reward based on the weighted change in attack surface and performance between the original and new VMP.

improper memory allocations. This can be represented in the model using mechanisms provided by modern hypervisors; for instance, advanced memory isolation can be achieved via AMD's Secure Encrypted Virtualization [292], which prevents access even from a hypervisor.

State transition function. The state transition function of the defender's MDP verifies whether a proposed migration satisfies the PM's resource constraints and the VM's resource requirements. If these constraints are met, the allocation state is updated accordingly, including the set of VM co-locations, the remaining PM capacities, and the associated performance metrics. Additionally, when selected, the isolation mechanism is simulated during the transition. This mechanism blocks the exploitation of the corresponding vulnerabilities from that point onward, while introducing additional resource overhead to simulate the isolation mechanism.

Reward function. The reward signal for the defender is *episodic*, provided only at the end of each episode. It quantifies changes in both performance metrics and the attack surface between the initial and finalized VMP configurations, reflecting variations in the VMP strategy. Since these aspects can only be evaluated after the placement is completed, the reward captures the cumulative effect of the entire sequence of actions taken during the episode (Figure 7.3).
The MOOP is encoded in the reward function as a weighted combination of the objectives (Equation 7.2), linking the interaction between the two agents. The weights w_s and w_p control the relative importance of security and performance, respectively, while ΔS and ΔP denote the changes in attack surface and performance metrics.

$$R = w_s \cdot \Delta S + w_p \cdot \Delta P, \tag{7.2}$$

Performance objectives

This study considers the following performance components as objective functions, each integrated into the environment simulator using reference mathematical models:

- **Resource utilization:** Measures the efficiency of CPU, RAM, and storage usage across PMs.
- **Tenant latency:** Estimates the communication delay experienced by tenants due to the VMP and PM bandwidths.

- **Energy consumption:** Quantifies the power usage of active PMs, reflecting the energy efficiency of the VMP.
- **Isolation costs:** Evaluates the overhead of maintaining isolation between VMs.
- **Migration costs:** Accounts for the resource impact of moving VMs between hosts during reconfiguration.
- **Replica allocation gains/costs:** Captures costs and benefits of replicating existing VMs.

A weighted-sum approach is again used here to balance the contributions of each component, with the signs of the weights indicating whether each component is to be minimized or maximized (e.g., maximize resource utilization, minimize latency). Components are min-max normalized per scenario to prevent scale differences from distorting the reward, ensuring that the weights alone determine each component's influence.

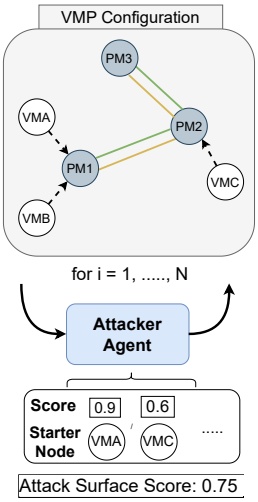


Figure 7.4: Approximation of the attack surface objective for the defender agent.

Attack surface objective

The defender's attack-surface objective component is computed by running the attacker agent for N episodes from several starter nodes on both the initial and modified VMPs (Figure 7.4). Each episode corresponds to a distinct critical attack path identified in the network, and the parameter N can be adjusted to trade off the number of critical attack paths considered against computational efficiency. A scalar security score is then computed for each critical path based on relevant statistics, such as the percentage of successful exploits. An aggregator function, such as the mean score, summarizes the average security gap of the VMP configuration. The difference in this aggregated score before and after the defender's actions constitutes the reward component, guiding the defender to iteratively reconfigure the network so as to minimize exposure to attacks consistent with the threat model.

7.5.3 Latent spaces for the defender agent

Building upon the RL-GCO formulations introduced in Chapter 5, latent observation and action spaces were designed for the defender agent in the secure VMP setting, extending the simplified benchmark abstraction presented in that chapter. This methodology, combined with the projection-based agent, was selected because it outperformed other RL-GCO approaches on this task. Furthermore, using latent continuous representations, rather than discrete ones, decouples the defender policy from specific infrastructure sizes and configurations, thereby improving scalability and facilitating transferability across heterogeneous virtualized environments.

- The defender **observation space** is constructed using a graph-level embedding representation derived from the underlying infrastructure graph. More specifically, node embeddings are first computed for the entities that compose the environment, including PMs, VMs, and their associated attributes. These node-level representations are then aggregated via pooling operations to produce a fixed-dimensional graph embedding that summarizes the current infrastructure state and placement configuration.
- Similarly, the defender's **action space** is represented in a latent embedding space. Placement actions, defined by the VM to be migrated and the destination PM selected by the defender,

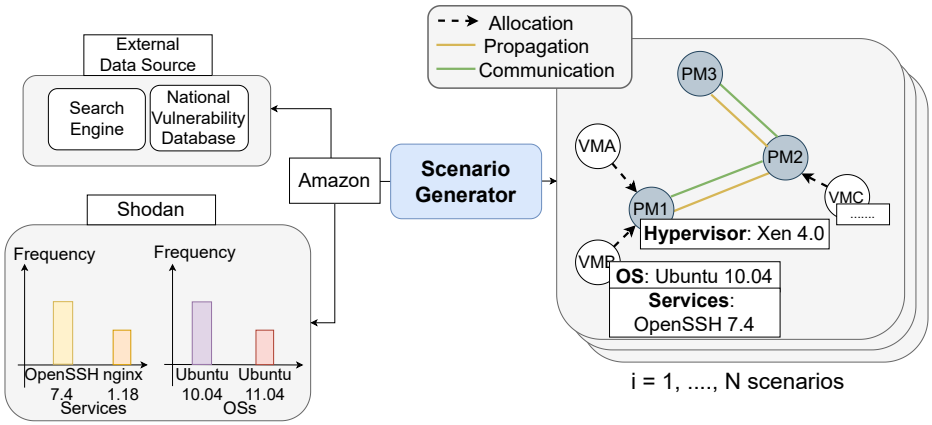


Figure 7.5: The scenario generator selects a cloud provider and uses CTI tools to gather statistics for generating realistic simulation scenarios.

are encoded by concatenating the embeddings of the source and target nodes involved in the placement decision. In contrast, isolation actions are represented using a standard one-hot encoding, since the set of possible isolation operations is fixed and small. The resulting one-hot vector is concatenated with each valid placement action representation, allowing both placement and isolation decisions to be jointly encoded within the action space.

7.5.4 Simulation environment

The simulation framework used throughout this chapter builds upon *C-CyberBattleSim* (Section 3.4), extending its capabilities to support virtualization-aware features, attacker–defender interactions, and the sequential decision-making dynamics required for secure VMP. More specifically, the framework introduces several extensions designed to capture both infrastructure-level placement characteristics and realistic attack-surface dynamics:

- **Scenario generator**: To emulate realistic cloud and virtualized infrastructures, the scenario generation process was extended to support virtualization-specific components, including PMs, hypervisors, and VMs. In addition to the CTI-based generation mechanisms introduced in previous chapters, metadata on hypervisors and their versions is collected from public documentation sources and integrated into the environment generation pipeline (Figure 7.5). Since empirical deployment frequencies for hypervisors are generally unavailable, these configurations are assigned to PMs via randomized sampling.
- **Vulnerability metadata retrieval**: The vulnerability retrieval pipeline was further extended to include vulnerabilities affecting OSs, hypervisors, and virtualization-specific software components. Vulnerability information is retrieved from the NVD, while additional metadata relevant to virtualization environments is also extracted when available. In particular, the framework collects additional contextual information such as affected guest OSs.
- **Overlay representation**: Additional graph-based features were integrated into the simulator to support the graph representation described in Section 7.4.1. In particular, the generated cyber scenarios are overlaid with virtualization-aware infrastructure attributes representing VM and PM properties. The framework supports configurable probability distributions for sampling infrastructure characteristics such as computational capacities,

storage resources, memory constraints, VM demands, and placement constraints. Furthermore, objective functions for the VMP optimization problem are directly integrated into the environment and dynamically computed from the graph structure and the current placement configuration at each timestep.

Overall, these extensions transform *C-CyberBattleSim* into a unified simulator for studying secure VMP under realistic attacker–defender dynamics, while supporting scalable experimentation on heterogeneous and synthetic virtualized infrastructures.

7.6 Experiments

This section presents the results of the proposed VMP solution and investigates the influence of several key factors, including threat models, security–performance trade-offs, heuristic strategies, and scalability considerations.

7.6.1 Experimental setup

The experimental pipeline was constructed based on the design choices highlighted in this section.

Table 7.2: Scenario statistics per cloud provider, including hypervisor type and associated average counts.

Cloud Provider	Hypervisor	# OS Vulns	# Service Vulns	# Hyp Vulns	% Hyp Vulns	# OS Versions	# Hyp Versions	# Service Versions
Linode	KVM	18906	1720	65	0.4%	13	4	24
Google	KVM	32203	1262	61	0.2%	24	4	25
AlibabaCloud	Xen	41944	1666	467	1.1%	15	80	24
DigitalOcean	KVM	10557	1813	62	0.6%	19	4	25
Amazon	Xen	31925	1340	429	1.5%	16	78	25

Scenario sets. Scenarios were generated to replicate the vulnerability distributions of the five major cloud providers identified via Shodan as having the highest numbers of vulnerable devices: *Amazon*, *Alibaba Cloud*, *Linode*, *Google*, and *DigitalOcean*. For each provider, as discussed in Section 7.5.4, Shodan was used to query the distributions of the most-used *services* and *OSs* across their Internet-connected devices. *Hypervisor types and versions* were approximated using publicly available information from each provider.
For each cloud provider, Table 7.2 shows several statistics, including the hypervisor used, the average number of vulnerabilities per scenario (across OSs, services, and hypervisors), and the proportion of hypervisor-level vulnerabilities. It also reports the average number of unique OS, service, and hypervisor versions per scenario, highlighting the diversity of components.
Each generated simulation scenario includes the vulnerabilities reported in the NVD for each component, assuming they are initially unpatched.

Domain randomization. For each cloud provider's data distribution, *60 synthetic scenarios* were created using domain randomization, altering virtualization parameters while preserving

statistical alignment with observed frequencies, yielding *300 total scenarios* for training and testing.
In the domain randomization process of each scenario, the following characteristics have been randomized:

- **Topology types:** The communication and propagation topologies across PMs were varied as follows: *Chain* (50%) and *Mesh* with varying levels of connectivity (50%). The generation process assumed that the sets of communication and propagation edges overlap in the scenario topologies ($E_{\text{communication}} = E_{\text{propagation}}$).
- **Initial VM allocation:** Different well-established VM allocation strategies—prioritizing performance (P), security (S), or offering a neutral (N) behavior—are used as initial placement policies in our scenarios: *smallest_fit* (20%) allocates each VM to the PM with the least available resources (P); *worst_fit* (20%) to the PM with the most resources (S); *round_robin* (20%) assigns each VM to the first PM with sufficient resources in circular order (S); *greedy_packing* (20%) allocates each VM to the largest available PM first (P); and *random* (20%) assigns VMs randomly to PMs (N).
- **Other features:** Several other features have been varied randomly in different custom-defined ranges (details in Appendix D.1): number of VMs/PMs, capacities of VMs/PMs, number of tenants, number of services per VM, PM bandwidths, firewall rules, data presence, visibility level, need for privilege escalation, service shutdown probabilities (representing *outages* in scenarios), VM importance (representing *node importance*).

The scenarios are divided into a training set (75%) and a held-out test set (25%) used to evaluate the agent's generalization capabilities. The split is guided by a complexity-based metric, ensuring that the agent is trained on simpler scenarios and evaluated on previously unseen, more complex ones. This setup allows us to assess whether the agent can transfer knowledge acquired during training to more complex scenarios. Scenarios are interchanged during both training and testing using a switching strategy, where a new scenario is selected at random every $T_s = 5$ episodes. This exposes the agent to multiple scenarios throughout training, enabling it to learn patterns across diverse scenario samples. During testing, the same mechanism allows us to assess the agent's generalization capabilities across several scenarios.

Training & evaluation hyperparameters. Each episode is set to last K$= 2 \times num_nodes$ iterations, ensuring fairness across different scenario sizes. Agents are trained for 100,000 iterations, after which their checkpoints are evaluated for 100 test episodes. For each test scenario, the episode that achieves the best trade-off is selected, and the results are averaged. To evaluate the attack surface, the attacker runs for $N_A = 5$ episodes before and after reconfiguration in order to generate the defender attack surface reward component.

Learning algorithm and embedding spaces. TRPO is used to train both the attacker and defender agents, motivated by its superior performance in the empirical comparison with other RL algorithms presented in Section 6.5.3. The same hyperparameters are retained in this chapter. Likewise, the GNN feature extractors used by both agents are trained using the unsupervised GAE-based approach introduced in Section 5.4.1, with the corresponding hyperparameters kept unchanged.
Graph embeddings are employed as observation-space representations and are constructed by aggregating node embeddings through a combination of *mean*, *max*, and *min* pooling operations in order to obtain fixed-dimensional representations independent of graph size. For the attacker's

vulnerability dimension, vulnerability descriptions are encoded with the pre-trained BERT, which is employed as a frozen feature extractor to generate semantic vulnerability embeddings.

Evaluation metrics. These metrics have been used for the evaluation of VMP reconfigurations:

- **Performance score difference** (ΔP): A weighted sum capturing variations in inter-tenant VM latency, resource efficiency, and energy consumption. Additional performance metrics (e.g., migration costs) are shown separately as they represent additional costs introduced.
- **Attack surface score difference** (ΔS): The relative change in the percentage of successful exploits before and after reconfiguration is used as the measure of the attack surface. Attackers are assumed to exploit the most critical vulnerabilities, and a negative ΔS with a large magnitude indicates their mitigation.
- **Weighted sum score difference**: ΔP and ΔS are combined into a single aggregate score with proper weights, which serves as an indicator of trade-off improvement.

The mathematical model underlying each of the above metrics is presented in Appendix D.2. Mean metrics and their 95% Bootstrapped Confidence Intervals (BCIs) are shown in plots with green shading indicating areas where beneficial changes are present for each metric (i.e., the metric improved when decreased or increased).

7.6.2 Threat model study

To understand how different attacker objectives influence the defender's learned strategies, we analyze the performance–security trade-offs obtained under the three attacker threat models considered.
Defender agents were retrained to jointly optimize system performance and resilience against an attacker agent (each retrained for a different threat model). By assigning equal weight to performance and attack-surface objectives ($w_S = w_P = 1$), this experiment reveals the average performance–security balance that the defender learns when both goals are treated as equally important.

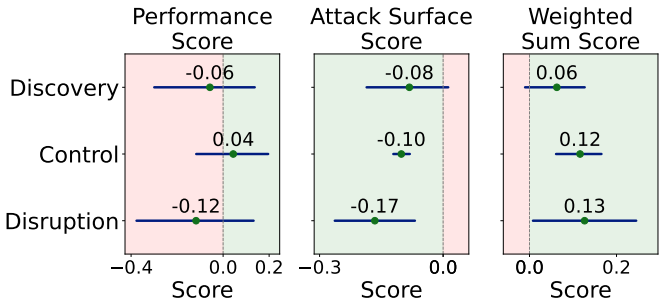


Figure 7.6: Average variation of metrics across different attacker threat models.

Analysis. Figure 7.6 summarizes the resulting trade-offs. Overall, the defender achieves more favorable balances against *control* and *disruption* attackers than against *discovery* attackers. On average, the learned reconfiguration strategies prevent approximately 8% of *discovery* exploits, 10% of *control* exploits, and 17% of *disruption* exploits.
Among the considered threat models, the *control* objective produces the most controllable attack-surface behavior for the defender. In this setting, the defender simultaneously reduces the

attack surface and achieves a mean improvement in infrastructure performance of approximately 4%. Although the weighted-sum scores obtained for *control* and *disruption* are relatively close, the *control* setting exhibits substantially narrower BCIs, indicating more stable and predictable optimization behavior.
These observations suggest that attacker objectives strongly influence the defender's learned policies and the resulting security–performance trade-offs. In particular, *discovery*-oriented attackers appear more difficult to mitigate efficiently, likely because information gathering objectives can often be achieved without requiring aggressive exploitation chains. Based on these results, the *control* threat model serves as the reference setting for subsequent analyses.

7.6.3 Security–performance trade-off study

To analyze how the defender adapts its strategy as the relative importance of security and infrastructure performance varies, we study the impact of the reward-weighting coefficient on the learned trade-offs. We introduce the security–performance coefficient ratio β, which governs how the defender prioritizes attack-surface reduction relative to system performance. By varying β, we observe how the defender adjusts its strategy across a continuum of possible trade-offs through its reward function, used for the training process.

Analysis. Figure 7.7 illustrates how the defender's behavior evolves as this weighting changes, reporting the value of different metrics. When $\beta = 0$, the reward function ignores the security objective entirely, leading to the highest performance improvements but also producing the largest attack surface. Conversely, when $\beta \to +\infty$, the defender prioritizes security almost exclusively, achieving the strongest reductions in attack success at the expense of substantial degradation in infrastructure performance.
The results therefore highlight the expected antagonistic relationship between security and operational efficiency in secure VMP. Increasing the emphasis on security progressively shifts the defender toward more aggressive isolation and placement strategies that successfully reduce attacker opportunities but simultaneously reduce consolidation efficiency and increase operational costs.

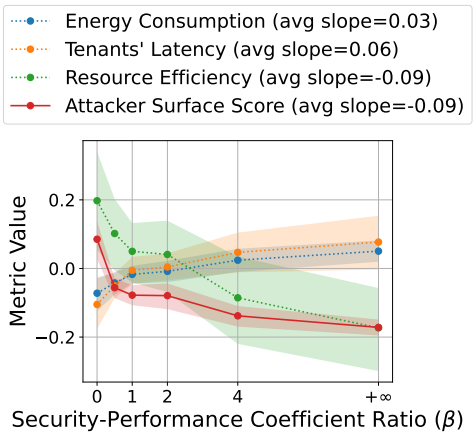


Figure 7.7: Average variation of metrics as a function of β, smoothed with a Savitzky-Golay filter.

The slope values reported in the figure legend summarize the average sensitivity of each metric to variations in β. Among the performance-related objectives, resource efficiency exhibits the largest BCIs, indicating that it is the least predictable component with respect to changes in reward weighting. For the remaining experiments, the coefficient is fixed to $\beta = 1$, corresponding to an equal weighting between security and performance objectives.

7.6.4 Impact of cloud vulnerability landscapes

To evaluate how provider-specific vulnerability landscapes affect the defender's optimization behavior, we compare the learned trade-offs across cloud scenarios generated from different CTI-derived vulnerability distributions. As described in our methodology (Section 7.5.4), the

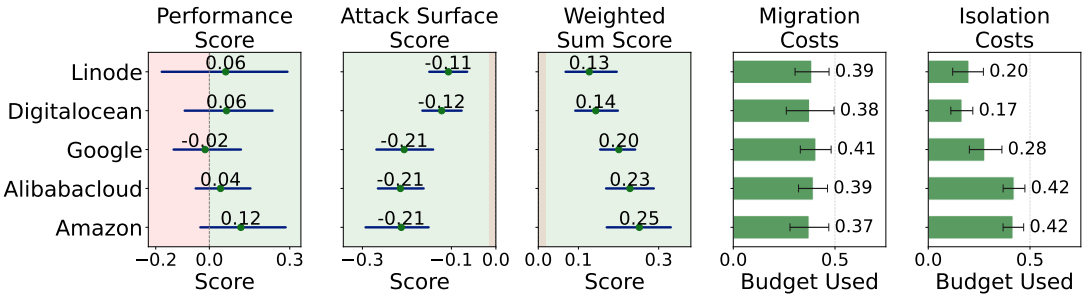


Figure 7.8: Average variation of metrics after reconfiguration across cloud scenarios.

vulnerability distributions of various cloud providers can be approximated using CTI data. These differences shape the vulnerability sets present on each node and the resulting attack chains available to the attacker, potentially affecting the defender's ability to balance performance and attack surface. We study different scenario sets based on this dimension as a splitting dimension.

Analysis. Figure 7.8 reports the defender's results across the considered cloud-provider scenarios. Both performance and attack-surface metrics exhibit noticeable variations across providers, highlighting the strong influence of the underlying vulnerability landscape on the defender's behavior. In addition to performance and attack-surface scores, the experiments also report migration and isolation costs as fractions of the available resource budget.
Results show that providers with a higher proportion of hypervisor-related vulnerabilities, such as *Amazon* and *Alibaba Cloud* (see Table 7.2), trigger more frequent isolation actions by the defender. This suggests that the learned policies adapt their mitigation strategies according to the characteristics of the attack surface induced by each provider-specific vulnerability distribution. Despite these variations, the defender consistently reduces the attack surface across all providers, while the weighted-sum objective remains positive overall, indicating beneficial security–performance trade-offs.

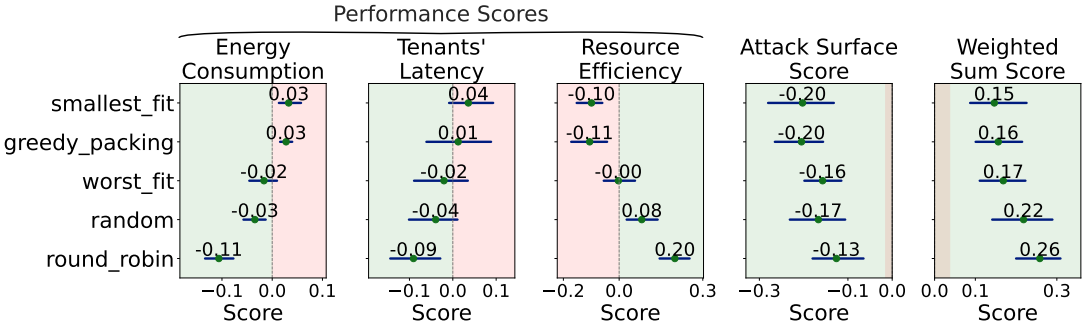


Figure 7.9: Average variation of metrics after reconfiguration across initial allocations.

7.6.5 Impact of initial allocation strategies

To analyze the influence of the initial VMP configuration on the defender's optimization behavior, we compare the resulting performance–security trade-offs across different allocation strategies. Because the initial placement determines the baseline values for both performance and attack-surface metrics—biasing the starting state toward either higher efficiency or higher security—the

defender is required to reconfigure scenarios in ways that improve upon this baseline, if possible, on both aspects.

Analysis. Figure 7.9 reports the average results obtained across the considered experimental conditions while decomposing the performance objective into its three main components.
The results show that the defender adapts significantly depending on the initial placement strategy. *Round-robin* allocations produce the largest improvements in performance metrics, including resource utilization, energy consumption, and latency. However, these scenarios also exhibit the smallest reductions in attack surface. In contrast, tightly packed allocations such as *greedy-packing* and *smallest-fit* produce lower performance improvements but enable substantially larger reductions in attack exposure, preventing approximately 20% of targeted exploits on average.
These observations further illustrate the inherent tension between infrastructure efficiency and security-aware placement decisions.

7.6.6 Scalability study

To assess whether the proposed framework can maintain effective optimization behavior as infrastructure scale increases, we evaluate the defender across progressively larger scenarios. This experiment evaluates the scalability of the approach across 16 scenarios, generated by varying the number of VMs and PMs in the range $[25, 100]$. The number of training iterations was scaled with scenario size, defined as $N_{\text{iter}} = \min(\text{num}_{\text{VMs}} \times \text{num}_{\text{PMs}} \times 100, 200000)$, capping the iteration costs for large scenarios. After training, agents were evaluated on the same training scenario (using $N_D = 20$), to assess the defender's ability to maintain effective trade-offs.

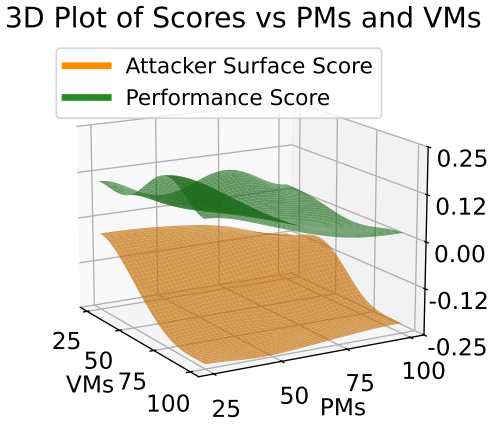


Figure 7.10: Average variation of metrics as a function of the number of VMs and PMs.

Trade-off performance analysis. Figure 7.10 illustrates how the learned metrics evolve as the infrastructure size increases. Attack-surface reduction improves from approximately 7% to 24% as the number of VMs grows, suggesting that larger infrastructures provide more opportunities for meaningful vulnerability mitigation and placement restructuring.
Conversely, performance improvements gradually decrease from roughly 14% to 1% as the number of PMs increases, reflecting the greater difficulty of preserving infrastructure efficiency in larger, more heterogeneous scenarios.
Importantly, despite these variations, both security and performance metrics remain within positive ranges across all tested configurations, demonstrating that the proposed framework maintains beneficial trade-offs even at larger scales.

Runtime analysis. To provide a runtime reference point, a configuration containing 25 VMs and 25 PMs using $N_D = 1$, $N_A = 1$, and 100 iterations per episode required approximately 36.1 ± 3.1 seconds per episode on the benchmark machine[5].

[5] x86_64 AMD Ryzen 9 5900X 12-Core Processor, 125 GiB RAM, Ubuntu 22.04.5 LTS

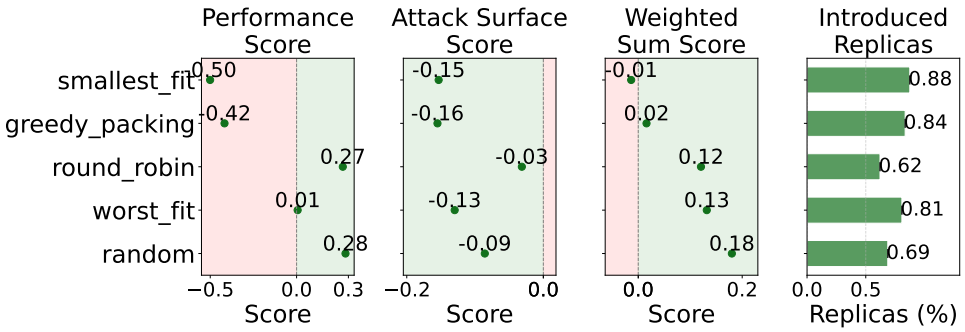


Figure 7.11: Average variation of metrics and % of replicas deployed across initial allocation strategies.

To quantitatively analyze scalability, a two-dimensional log-linear model was fitted (Equation 7.3), where T denotes the execution time.

$$\log T = k_{\mathrm{VM}} \log(\mathrm{VM}) + k_{\mathrm{PM}} \log(\mathrm{PM}) + c \tag{7.3}$$

The fitted coefficients indicate that doubling the number of VMs increases runtime by approximately $1.79\times$, whereas doubling the number of PMs increases runtime by approximately $1.24\times$. These results indicate sub-quadratic empirical scaling behavior and suggest that the proposed framework remains computationally feasible for substantially larger environments.

7.6.7 Replica pool study

To investigate how the defender behaves when replica allocation is incorporated into the optimization process, we extend the action space to include replica deployment decisions, in addition to standard VMP actions. Introducing replicas can increase tenant satisfaction by improving service availability and redundancy. However, replicas also expand the attack surface and may negatively affect performance metrics such as energy consumption.
To study these trade-offs, we extend the defender's decision process by allowing it to observe, for each VM, between zero and two potential replicas as part of the observation space (incorporated as additional nodes in the graph). The defender can then decide whether and when to deploy these replicas—represented as additional points in the continuous action space—alongside standard VM reallocations across PMs. Tenant satisfaction is incorporated into the reward function with equal weighting to the other two components.

Analysis. Figure 7.11 summarizes the resulting trade-offs across different initial allocation strategies. Tightly packed allocations, such as *smallest-fit* and *greedy-packing*, lead to the highest replica deployment rates but also the largest performance degradation.
As expected, increasing the number of replicas generally reduces infrastructure performance due to additional resource consumption. Nevertheless, the attack surface consistently decreases despite the increased infrastructure redundancy, indicating that the defender learns to deploy replicas in ways that improve availability while simultaneously mitigating critical attack paths. Overall, weighted-sum scores remain positive across the evaluated settings, with an average replica deployment rate of approximately 77%, demonstrating the defender's ability to balance availability, performance, and security objectives simultaneously.

7.7 Deployment pipeline

We present a proof-of-concept offline decision-support pipeline that uses the solution to assist system administrators by recommending VMP reconfigurations and patching priorities. The defender can mitigate a subset of vulnerabilities, while the remaining ones are ranked for patching based on attacker activity, prioritizing those most frequently targeted, as they are the most likely to be exploited for that threat model.

1. The *system administrator* provides performance *data* (e.g., PM capacity, power use), VM priorities, and policy constraints (e.g., whether replication is allowed). Scanners are used to identify potential vulnerabilities in the OS, services, and hypervisors, while administrators mark the vulnerabilities that have already been patched.
2. The administrator sets *trade-off weights* and selects *threat models* to defend against, defining the reward functions used for training.
3. The *defender agent* is *trained* either (i) on synthetic scenarios generated from empirical distributions (in case of privacy concerns) or (ii) directly on a digital twin of the real network.
4. Once trained, the defender agent executes N_D episodes on the digital twin, generating N_D candidate VMP solutions at *inference time*. Each candidate solution is evaluated by running the attacker agent for N_A episodes both before and after the network reconfiguration, as described in Section 7.5.2. This process identifies a corresponding set of critical attack paths, with N_A controlling the number of paths sampled and, consequently, the accuracy of the attack-surface estimate.
5. Candidate *solutions* are *ranked* by their weighted sum score, and the resource manager selects one that balances the objectives as desired.
6. The selected *VMP solution* is *deployed*, applying reallocation, isolation, and replica strategies as required. Remaining unpatched vulnerabilities can be patched, with the patching order optionally inferred from statistics derived from the N_A attacker episodes.

The execution of defender actions assumes the availability of interfaces that enable VM reallocation across PMs, along with mechanisms that support the application of isolation measures.

Analysis. We tested the pipeline on the digital twin of a virtualized real-world infrastructure consisting of 5 PMs hosting 27 VMs. Vulnerability data were gathered using Debsecan for OS and services, and the NVD for hypervisors, totaling 25,957 vulnerabilities (25,666 for OS/services, 291 for hypervisors), with 27 distinct OS versions and 2 hypervisor versions (ESXi 7.0U3, Proxmox 8.3.0). In addition, system and resource usage information (e.g., memory) was gathered using standard Unix utilities. Using the *control* threat model and equal weighting across performance and attack surface, the re-trained agents in the digital twin have been tested for $N_D = 10$ and $N_A = 3$, resulting in the following best configuration:

- **Performance:** Energy consumption (-3%), resource efficiency (+3%), and latency (-23%)
- **Attack surface:** Attack surface score dropped by 17%, reflecting a reduced set of attack chains

These results demonstrate that the proposed framework can simultaneously improve infrastructure performance and reduce the attack surface, highlighting the effectiveness of incorporating threat-aware optimization into VMP decisions.

7.8 Summary

This chapter presents a potential approach for deriving defense mechanisms directly from learned attacker strategies. The proposed framework is applied specifically to virtualized infrastructures and the VMP problem, while introducing a more general methodology that can be transferred to a wide range of security-oriented optimization and defense scenarios.
After revisiting the *secure* VMP problem, this chapter introduces a MARL-based solution comprising an attacker agent and a defender agent. The attacker agent is trained to identify the most critical attack paths based on a specified threat model. This trained attacker is then used during defender training to provide a reward signal—complemented by performance objectives—guiding the defender in performing more secure and efficient VMP reconfigurations starting from an initial proposed solution. Experimental results demonstrate the defender agent's ability to jointly optimize for both performance and attack-surface objectives.
Against diverse vulnerability profiles, covering typical services, OSs, and hypervisors of real cloud providers, and varying initial placement strategies, the improvement averages a joint 17% in attack surface reduction and 5% in performance score increase.

Link to the thesis objectives

This chapter further connects and validates the core objectives of the thesis with a specific use case by reusing the scalable latent-space RL framework introduced in Chapter 5 together with the critical attack path discovery agent proposed in Chapter 6. More specifically, the attacker agent is integrated into a MARL setting alongside a defender agent that performs the GCO task of VMP, with the objective of minimizing the attack surface while simultaneously optimizing performance objectives. This chapter therefore demonstrates how the proposed latent-space representations and scalable RL methodologies can be transferred beyond critical attack path prediction to a broader class of security-aware networking optimization problems. To support this application domain, the *C-CyberBattleSim* framework introduced in Chapter 3 was further extended with virtualization-aware modeling capabilities and features enabling realistic attacker–defender interactions in virtualized infrastructures.

CONCLUSIONS

Assessing the security posture of complex networked infrastructures remains an open and challenging research problem. The continuous growth in infrastructure scale, heterogeneity, interconnected services, and software dependencies has significantly increased the complexity of modern systems, making the manual design of security metrics and attack modeling strategies increasingly impractical.

This thesis investigated the development of context-aware security metrics by modeling attacker behavior as a sequential decision-making process within networked environments. The objective was to capture vulnerability-chaining dynamics and assess security from the perspective of the potential impact of real-world attacks. Rather than treating vulnerabilities as isolated entities, the proposed approach accounts for their interdependencies and propagation mechanisms across network infrastructure, enabling the identification of critical attack paths without requiring exhaustive enumeration of the attack graph.

Motivated by the growing adoption of ML techniques to replace manually engineered heuristics with data-driven approaches, this work relied on RL as the primary learning paradigm. RL naturally models sequential decision-making processes and therefore provides an appropriate framework for representing adversarial behaviors and attack progression strategies within network infrastructures.

More specifically, this thesis addressed several key challenges commonly encountered in RL-based cybersecurity applications, including the design of realistic simulation environments, the modeling of representative network scenarios, and the development of RL solutions that achieve both scalability and generalization across heterogeneous infrastructures. In addition, the proposed RL-oriented methodologies were designed not only for cybersecurity applications but also as more general RL solutions for GCO problems, when combined with appropriate graph-based learning approaches.

Part I introduced the background necessary to understand vulnerability data sources (Chapter 1) and attack planning methodologies (Chapter 2). These chapters highlighted the growing adoption of ML techniques to solve problems across these two domains and positioned the contributions of this thesis relative to existing approaches in a broader scope, while discussing their inherent limitations.

The subsequent Part II presents two chapter contributions centered on a simulation framework designed to support the learning of critical attack strategies.

Chapter 3 proposes a POMDP formulation for automated critical attack path discovery. The objective of this chapter was to address two major challenges: generating realistic synthetic network scenarios suitable for RL training and evaluation, and modeling realistic vulnerability-exploitation dynamics that represent attacker behavior within complex infrastructures.

Chapter 4 extended this research direction by introducing a multi-label learning framework for automatically modeling vulnerability semantics and dynamics from historical vulnerability data. The proposed approach reduces dependence on manually engineered expert knowledge by automatically inferring security-relevant metadata for newly disclosed vulnerabilities. This inferred metadata enriches the POMDP-based simulation framework by predicting related weaknesses, attack patterns, and adversarial techniques that may be enabled by a given vulnerability. As a result, the framework can model and assess vulnerabilities even when explicit annotations are unavailable, thereby improving its scalability and applicability to emerging threats.

Subsequently, the thesis shifts its focus to scalability and generalization in RL over graph-

structured environments in Part III, which constitute the underlying representation of the considered network scenarios.

Chapter 5 proposed a general RL framework for GCO problems, with particular emphasis on improving scalability and generalization independently of the specific benchmark task. The proposed methodologies were evaluated on multiple benchmark optimization problems to demonstrate their broader applicability. Building upon these foundations, Chapter 6 adapted these approaches specifically to the problem of critical attack path discovery, enabling scalable attack planning across complex network topologies while learning strategies that transfer effectively across heterogeneous scenarios. Finally, Chapter 7 demonstrated the applicability of the proposed critical attack path discovery framework within practical networked optimization problems that increasingly rely on security-aware metrics. More specifically, the chapter considered the use case of VMP as a representative benchmark scenario. The proposed approach showed how attacker behavior modeling and critical attack path reasoning can be integrated into infrastructure optimization via a multi-agent RL formulation that jointly models attacker and network configuration behaviors, enabling security-aware resource allocation strategies.

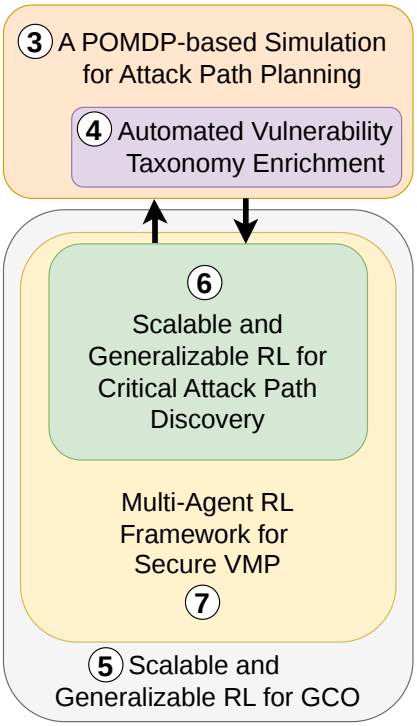


Figure 2: Overview of the relationships between the main contributions presented in this thesis and their dependencies.

The relationships between the thesis contributions are summarized in Figure 2. Chapter 3 introduces the simulation environment, which is enhanced in Chapter 4 through automated vulnerability enrichment. This enriched POMDP environment serves as the benchmark for developing and evaluating the RL methodology presented in Chapter 6. Building on this, Chapter 7 applies the proposed RL framework to the VMP, where the attacker agent becomes the security objective considered for deriving mitigation strategies. While Chapters 6 and 7 focus on two specific GCO problems, Chapter 5 abstracts the underlying methodology into a general framework applicable to a broader class of tasks.

A primary limitation of this thesis is that, although the individual components are evaluated independently within their respective experimental settings, the complete framework has not yet been validated through an end-to-end evaluation combining all contributions. In particular, the interactions between the simulation environment, vulnerability enrichment, RL-based attack-path planning, and subsequent security optimization have not been systematically assessed as an integrated system. Nevertheless, the individual evaluations provide evidence for the validity and effectiveness of each component and establish the foundations for their future integration and end-to-end assessment. The following section therefore discusses the main limitations identified for each contribution and outlines perspectives for future research.

Perspectives for future research

Although this thesis proposed several contributions toward addressing the challenges of critical attack path discovery and security-aware optimization, these contributions are not intended to provide exhaustive solutions to the problem space. The complexity, scale, and evolving nature of modern networked infrastructures still leave many research challenges open. Consequently, several limitations remain across the proposed methodologies, motivating numerous opportunities for future research and further improvements.
The following sections discuss the main limitations associated with each contribution chapter and outline promising directions for future work.

Extensions to the attack path planning framework

Chapter 3 introduced a POMDP-based simulation framework for automated critical network attack path discovery. The proposed framework enables the modeling of sequential attacker behavior under partial observability while improving the realism and automation capabilities of existing cybersecurity simulation environments.
Although the proposed framework represents a step toward more scalable and realistic attack path modeling, several limitations remain and motivate future research directions:

- **Limitation:** Incorporating vulnerability metadata records makes the simulated environment more plausible, but does not necessarily ensure high-fidelity representations of real-world attack behavior. The simulation still relies on approximate mappings between vulnerabilities and a limited set of abstract simulation actions. Moreover, these mappings may be affected by inaccuracies or inconsistencies in expert-provided vulnerability labels and annotations.
  Potential improvement: Future research could investigate more systematic and fine-grained mappings between vulnerability characteristics and simulation actions, potentially combining expert knowledge with empirical data from real-world historical exploitation behavior to improve simulation fidelity.
- **Limitation:** Vulnerability distributions are currently approximated using publicly observable Internet-scale statistics. However, such distributions may not accurately capture the internal operational characteristics of real infrastructures, since many hosts, services, configurations, and security policies are not exposed through Internet-wide scanning platforms.
  Potential improvement: Future research could investigate richer cyber-terrain modeling methodologies that incorporate organization-specific internal network configurations and contextual infrastructure properties beyond internet-wide scanning platforms.
- **Limitation:** Extending the framework with additional environmental dynamics, defensive mechanisms, or infrastructure features still requires substantial manual engineering and expert-driven modeling efforts.
  Potential improvement: Future work could explore automated logic-construction pipelines capable of dynamically generating and extending POMDP components directly from operational infrastructure data rather than manually defining simulation dynamics.
- **Limitation:** Vulnerability-level actions remain abstract representations of exploitation behavior. While this abstraction improves generalizability across the evolving threat landscapes, it does not directly capture the executable exploits and operational constraints encountered in real-world PT workflows.

Potential improvement: Future research could further bridge the gap between simulation and operational environments by allowing the incorporation of exploit-specific characteristics, such as exploit availability, reliability, and execution feasibility, into the POMDP dynamics. This extension would enable more realistic modeling of attack execution while preserving the flexibility and scalability of the current vulnerability-centric framework.

- **Limitation:** Vulnerability outcomes are currently modeled using MITRE ATT&CK Tactics in `C-CyberBattleSim`, which provide relatively coarse-grained representations of attacker behavior.
  Potential improvement: Future research could investigate finer-grained formulations based on ATT&CK techniques and procedures to more accurately capture realistic attacker behavior and multi-stage attack progression. Such an approach would provide a richer representation of adversarial dynamics at the cost of defining transition logic for each technique. To ensure scalability, automated methods for inferring and encoding transitions across these large-scale taxonomies could be explored, thereby reducing reliance on continuous manual knowledge engineering.

Further techniques for enhancing the vulnerability classification

Chapter 4 introduced a hierarchical multi-label classification framework for automatically learning vulnerability dynamics from textual vulnerability descriptions. Although the proposed framework achieved strong classification performance, several limitations remain and motivate future research directions:

- **Limitation:** The CVE-to-CAPEC mappings used in this study may contain false positives or other labeling inconsistencies, as also highlighted in previous studies [208]. Nevertheless, our results indicate that the models are able to learn meaningful CVE-to-CAPEC mapping patterns from the available data, suggesting that improvements in the quality and reliability of the labels could further enhance their performance.
  Potential improvement: Future work could construct a curated gold-standard dataset in which CVE-to-CAPEC mappings are manually reviewed and validated by cybersecurity experts. Such a dataset could provide higher-quality supervision for training and a more reliable benchmark for evaluation, while also enabling the systematic identification and removal of false-positive mappings.
- **Limitation:** The hierarchical classification strategy increases training cost, as multiple LMs must be fine-tuned across different taxonomy subsets.
  Potential improvement: Future work could investigate lighter architectures, parameter-efficient fine-tuning strategies, or shared hierarchical pre-trained representations to reduce computational requirements.
- **Limitation:** The current study primarily focused on classification performance and did not extensively examine long-term model maintenance in the face of continuously evolving cybersecurity taxonomies. Although the proposed hierarchical formulation partially mitigates this issue by restricting updates to specialized components of the hierarchy, the problem of efficiently adapting these models to evolving taxonomies, as well as evaluating the associated maintenance and retraining costs, remains largely open.
  Potential improvement: Future work could explore continual learning [293] and incremental adaptation techniques, combined with more efficient parameterization strategies, to enable scalable model updates without requiring complete retraining while preserving previously acquired knowledge.

- **Limitation:** As mentioned in the previous point, the current framework relies on fixed output layers instantiated from predefined class sets, requiring additional fine-tuning whenever new taxonomy elements are introduced. Although hierarchical taxonomies can alleviate this limitation by reducing the amount of retraining required, proper fine-tuning strategies for smaller models in the hierarchy remain necessary to effectively incorporate previously unseen classes.
  Potential improvement: Future research could explore more flexible architectures, such as pointer-based [294] or retrieval-based classification methods that dynamically score candidate classes without requiring fixed output dimensions.
- **Limitation:** Interpretability remains limited due to the use of standard feed-forward classification heads in the proposed methodology.
  Potential improvement: Future research could investigate uncertainty-aware learning methods [295] and explainable AI techniques to improve confidence estimation and prediction transparency.
- **Limitation:** The current models rely exclusively on textual descriptions of vulnerabilities as input features.
  Potential improvement: Future work could explore multimodal learning approaches that combine textual vulnerability descriptions with additional contextual information, such as exploit availability, vendor metadata, affected products, or related vulnerabilities.

More robust and efficient strategies for RL-GCO

Chapter 5 introduced a scalable and generalizable RL framework for GCO, significantly improving performance compared to existing RL-based approaches.
Nevertheless, several limitations remain and motivate future research directions:

- **Limitation:** The study currently relied on a single encoding strategy for each action-space component and a single decoding strategy for projection agents, both selected empirically from a limited set of alternatives.
  Potential improvement: Future work could explore adaptive or dynamically learned encoding-decoding strategies to improve flexibility and generalization.
- **Limitation:** The unsupervised action embeddings are not jointly optimized with the downstream RL process. This design choice was made to ensure fair comparisons across methods by using a common embedding space; however, it may limit the maximum achievable performance, as the learned representations are not adapted to the specific objectives of the RL agent.
  Potential improvement: Future research could investigate end-to-end optimization strategies that jointly learn action representations and RL policies. Such approaches would enable a systematic comparison between purely unsupervised embeddings and task-adaptive embeddings obtained through joint fine-tuning, providing insights into the trade-offs between generality, fairness of evaluation, and downstream performance.
- **Limitation:** Although the current evaluation covered 101 scenarios with 20 runs per configuration, the benchmark remains representative rather than exhaustive and focuses primarily on RL-based methods.
  Potential improvement: Future work could extend comparisons to larger scenario sets, larger scenario sizes, as well as non-RL optimization techniques, including heuristic search, metaheuristics, or differentiable optimization methods.
- **Limitation:** Dynamically changing action spaces remain a major challenge for projection-

based agents, as current sub-linear retrieval methods in the case of an NN decoding strategy require rebuilding NN indices whenever the action space changes.
Potential improvement: Future research could explore dynamic indexing methods or online retrieval structures that more efficiently support evolving action spaces.

Limiting action space explosion in RL for critical attack path discovery

Chapter 6 adapted the proposed latent-space RL framework to the task of critical attack path discovery, demonstrating significant improvements in scalability and generalization over existing approaches.
Despite these improvements, several limitations remain and motivate future research directions:

- **Limitation:** The expanded action space may still contain many unrealistic or suboptimal actions that could be easily filtered by incorporating well-established human knowledge on the task.
  Potential improvement: Future work could integrate domain knowledge and expert-driven heuristics to proactively filter out infeasible or strategically irrelevant actions based on specific threat models.
- **Limitation:** The current framework relies on flat action selection strategies over the hierarchical selection choice of (source node, target node, vulnerability, outcome) actions. Although latent-space representations significantly improve scalability, further advances are still required to support infrastructures containing thousands or millions of nodes.
  Potential improvement: Future research could investigate hierarchical RL [296, 297] approaches that decompose attacker decisions into distinct hierarchical stages, thereby modeling the iterative conditional selection of the source node, target node, vulnerability, and outcome. Moreover, future work could investigate graph partitioning strategies or efficient representations to further improve scalability to the aforementioned scales.

Toward realistic deployment of security-aware VMP

Chapter 7 introduced a MARL framework for proactive VMP balancing security and performance objectives. Although the proposed framework demonstrated promising results, several limitations remain and motivate future research directions:

- **Limitation:** The proposed approach is evaluated exclusively in synthetic or simulated digital-twin environments, where network configurations and attack dynamics are simplified representations of real infrastructure. Building accurate digital twins of large-scale virtualized enterprise and IaaS networks is challenging due to the need for detailed, potentially sensitive infrastructure information. Moreover, the approach has not been evaluated with a real actuation layer, particularly for executing defender-side VMP reallocation actions under realistic operational constraints.
  Potential improvement: Future work could incorporate richer network telemetry, heterogeneous infrastructure, and realistic operational constraints, and extend the framework with a controlled actuation layer allowing attacker and defender agents to interact with real or sandboxed virtualized networks. This would enable VMP reallocation decisions to be evaluated under realistic resource, performance, and service-level constraints.
- **Limitation:** Execution time remains a major limitation due to the realistic attacker modeling strategy, where agents evaluate complete vulnerability chains (through several RL episodes) instead of relying on simplified aggregate security metrics.

Potential improvement: Future work could explore approximate attacker modeling strategies leveraging offline attack-frequency statistics to reduce computational cost while preserving realism.

- **Limitation:** The current study focused primarily on proactive placement strategies and did not extensively investigate dynamic migration scenarios.
  Potential improvement: Future research could investigate dynamic migration strategies and compare proactive consolidation against reactive migration approaches under active attack scenarios.
- **Limitation:** The proposed MOOP framework remains highly sensitive to the weighting coefficients used to balance the different optimization objectives. Determining appropriate weight configurations is challenging and often requires substantial manual tuning depending on the target infrastructure and optimization priorities.
  Potential improvement: Future work could investigate automated weight adaptation and optimization strategies capable of dynamically determining suitable trade-offs between objectives based on higher-level optimization criteria and operational constraints.

RÉSUMÉ EN FRANÇAIS

Cette section propose un résumé en français de ce manuscrit de thèse. Son objectif est de présenter les principaux enjeux, approches et contributions de ces travaux dans une forme synthétique et accessible aux lecteurs francophones. Elle s'inscrit également dans une démarche de diffusion scientifique en langue française. Les sections suivantes introduisent le contexte et les objectifs de la thèse, puis présentent une synthèse des différents chapitres ainsi que des contributions majeures développées au cours de ces travaux.

Introduction

Au cours des dernières décennies, les infrastructures interconnectées ont connu une croissance considérable en termes de taille et de complexité. Elles constituent aujourd'hui un élément essentiel des organisations modernes, des systèmes industriels et des infrastructures critiques. Parallèlement, les couches logicielles et les actifs numériques déployés au sein de ces environnements se sont également multipliés, entraînant une augmentation continue de la surface d'attaque ainsi qu'un nombre toujours plus important de vulnérabilités publiquement divulguées. Cette évolution rend la gestion des risques cyber de plus en plus difficile, en particulier pour les infrastructures de grande échelle.
L'un des principaux défis actuels pour les organisations concerne la priorisation des vulnérabilités à corriger. Face au volume croissant de failles identifiées, il est rarement possible de toutes les traiter immédiatement. Les méthodes traditionnelles de classification reposent principalement sur des métriques génériques de sévérité et ne considèrent pas suffisamment le contexte propre à chaque infrastructure, tel que la topologie du réseau, les exigences opérationnelles ou le modèle de menace considéré. Pourtant, la criticité réelle d'une vulnérabilité dépend fortement de son environnement d'exploitation, notamment des interconnexions entre actifs, de son accessibilité par un attaquant, des services exposés et du rôle stratégique des ressources compromises.
Cette thèse s'intéresse à l'identification des chemins d'attaque critiques au sein d'infrastructures réseau afin d'améliorer la priorisation des vulnérabilités. L'objectif est d'identifier, parmi l'ensemble des vulnérabilités présentes dans un système, celles qui jouent un rôle déterminant dans les scénarios d'attaque les plus impactants, en tenant compte de la topologie du réseau et du modèle de menace considéré. Une telle approche permet de dépasser les métriques statiques de sévérité en développant des évaluations de sécurité contextuelles, capables de mieux représenter la surface d'attaque spécifique à chaque infrastructure.
Les graphes d'attaque ont largement été étudiés pour représenter les séquences d'exploitation possibles dans un réseau et analyser les chemins permettant à un attaquant d'atteindre ses objectifs. Cependant, les approches traditionnelles reposent généralement sur l'énumération préalable de l'ensemble des chemins d'attaque possibles avant d'identifier les scénarios les plus critiques. Cette exploration exhaustive devient rapidement impraticable dans les infrastructures modernes, caractérisées par un nombre croissant d'hôtes, de services, de vulnérabilités et d'interconnexions.
Afin de dépasser ces limitations, des approches récentes cherchent à remplacer l'exploration exhaustive des graphes d'attaque par des stratégies de décision apprises à partir de données. Dans ce contexte, l'apprentissage par renforcement (Reinforcement Learning, RL) apparaît particulièrement prometteur, car il permet d'apprendre des heuristiques de recherche directe-

ment à partir d'interactions avec des environnements simulés. En exploitant des représentations numériques de la topologie du réseau, des vulnérabilités et de leurs relations, les agents RL peuvent apprendre à prioriser efficacement les actions d'attaque et à identifier des chemins critiques sans nécessiter une exploration complète de l'espace des possibilités. Bien que ces approches ne garantissent pas nécessairement l'optimalité globale des solutions obtenues, elles offrent un compromis intéressant entre qualité des stratégies découvertes et capacité à traiter rapidement des environnements complexes à grande échelle.
Toutefois, l'application du RL à la découverte de chemins d'attaque critiques reste confrontée à plusieurs défis majeurs. D'une part, le RL nécessite un grand nombre d'interactions, ce qui impose l'utilisation de simulateurs capables de fournir des exécutions rapides tout en représentant de manière suffisamment réaliste les infrastructures réseau, les vulnérabilités présentes et leurs dynamiques d'exploitation. Ces environnements doivent également permettre la génération automatisée de scénarios variés à grande échelle, afin de soutenir à la fois l'apprentissage des agents et leur évaluation dans des contextes diversifiés. D'autre part, les politiques apprises doivent être capables de généraliser au-delà des scénarios observés durant l'entraînement, le réentraînement complet étant difficilement envisageable pour chaque nouvelle infrastructure ou évolution du système. Elles doivent ainsi rester efficaces face à des réseaux présentant des tailles, configurations et topologies différentes, tout en répondant aux contraintes de passage à l'échelle imposées par les infrastructures modernes.
Cette thèse vise ainsi à répondre à ces limitations en développant des approches basées sur l'apprentissage automatique pour la découverte et la priorisation de chemins d'attaque critiques. Les contributions proposées s'articulent autour de quatre axes principaux : la conception d'environnements de simulation réalistes et automatisés pour l'entraînement et l'évaluation des agents RL, l'enrichissement automatique des connaissances liées aux vulnérabilités afin de faciliter l'intégration de nouvelles menaces dans ces environnements, le développement de stratégies RL scalables et généralisables adaptées à des infrastructures complexes, et l'étude de l'exploitation des comportements appris pour soutenir des stratégies de mitigation prenant en compte le contexte opérationnel.

Contexte et travaux connexes

La découverte automatisée de chemins d'attaque critiques repose sur deux défis complémentaires : la capacité à représenter efficacement les connaissances liées aux vulnérabilités et aux comportements adverses, et la capacité à explorer des espaces d'attaque complexes à grande échelle. Les sources de connaissances cyber, telles que les bases de vulnérabilités et les taxonomies d'attaques, constituent des éléments essentiels pour modéliser les interactions entre vulnérabilités, techniques adverses et composants d'une infrastructure au sein d'environnements de simulation. Cependant, ces sources restent hétérogènes, incomplètes et partiellement interconnectées, limitant leur exploitation automatique pour construire des représentations suffisamment riches et réalistes. Plusieurs travaux ont ainsi étudié l'enrichissement ou la classification automatique des vulnérabilités vers ces taxonomies, mais des limitations persistent notamment en termes de couverture, de granularité et de disponibilité des annotations pour les vulnérabilités récemment publiées.
Au-delà de la représentation des connaissances, l'identification des chemins d'attaque critiques constitue également un défi majeur. Les approches traditionnelles fondées sur les graphes d'attaque permettent de représenter les séquences d'exploitation possibles au sein d'une infrastructure. Cependant, l'identification des chemins les plus critiques repose souvent sur

l'exploration ou l'analyse d'un espace de scénarios potentiellement très large, ce qui limite leur applicabilité aux environnements modernes caractérisés par des infrastructures de grande taille, hétérogènes et dynamiques. Ces limitations ont motivé l'émergence d'approches fondées sur l'apprentissage automatique, capables d'apprendre des stratégies de recherche et de priorisation sans nécessiter l'exploration exhaustive de l'espace des chemins possibles.
Dans cette thèse, ces deux problématiques sont abordées de manière complémentaire. Le premier axe porte sur le développement de mécanismes automatisés d'enrichissement des vulnérabilités, visant à extraire les informations sémantiques nécessaires à la construction d'environnements de simulation réalistes, notamment pour modéliser les conséquences potentielles de leur exploitation. Le second axe consiste à formaliser la découverte de chemins d'attaque comme un problème d'optimisation combinatoire sur graphes (Graph Combinatorial Optimization, GCO), et à étudier l'apprentissage par renforcement (Reinforcement Learning, RL) comme un moyen de découvrir des heuristiques efficaces, capables de se généraliser à différents scénarios et de passer à l'échelle sur des infrastructures complexes.
Ces deux axes s'inscrivent dans le prolongement de travaux existants portant, d'une part, sur l'enrichissement automatisé des vulnérabilités à partir de leurs descriptions textuelles et, d'autre part, sur l'analyse des graphes d'attaque pour la priorisation des risques. Malgré les avancées réalisées, plusieurs limites persistent, notamment en matière de couverture des informations extraites, de généralisation des méthodes proposées et de passage à l'échelle face à la complexité des infrastructures réelles. L'objectif de cette thèse est ainsi de contribuer à une approche intégrée de l'analyse des risques cyber, en articulant l'extraction automatisée de connaissances à partir des descriptions de vulnérabilités avec l'apprentissage de stratégies de recherche de chemins d'attaque. Cette approche vise à améliorer la représentation des scénarios d'exploitation, à faciliter l'estimation des risques et à permettre une priorisation des vulnérabilités mieux adaptée au contexte opérationnel des infrastructures étudiées.

Contribution 1 — Un environnement de simulation réaliste pour la découverte de chemins d'attaque critiques

Les approches fondées sur le RL nécessitent des environnements d'entraînement et d'évaluation capables de représenter fidèlement le système cible. Dans le contexte de la découverte de chemins d'attaque, un apprentissage directement sur des infrastructures réelles est peu envisageable, en raison des risques associés à l'exploitation de vulnérabilités, des coûts de déploiement d'environnements dédiés et du temps nécessaire à l'exécution des interactions requises par les algorithmes de RL. La simulation constitue donc l'approche privilégiée, à condition de limiter l'écart entre l'environnement simulé et les propriétés des infrastructures réelles.
La première contribution de cette thèse est un environnement de simulation reposant sur une formulation du problème sous la forme d'un processus de décision markovien partiellement observable (POMDP). Celui-ci modélise le caractère séquentiel des attaques, l'évolution progressive des connaissances de l'attaquant sur le réseau et les effets des exploitations de vulnérabilités. Développé comme une extension de `Microsoft CyberBattleSim`, le framework introduit une modélisation plus réaliste des états, des transitions et des actions en s'appuyant sur des métadonnées issues de vulnérabilités réelles.
Une seconde contribution de ce chapitre concerne la génération automatisée de scénarios de simulation. Le framework s'appuie sur des statistiques issues de bases de vulnérabilités publiques et de sources de cyber threat intelligence afin de reproduire des distributions réalistes de logiciels, de services et de vulnérabilités. À partir de ces distributions, il permet de paramétrer

de nombreuses caractéristiques des scénarios générés, telles que la topologie du réseau, les configurations des hôtes, ou encore les règles d'allocation des logiciels et des vulnérabilités. Cette approche permet de produire automatiquement un grand nombre d'environnements variés, tout en conservant un ancrage dans des statistiques représentatives des infrastructures réelles, pour l'entraînement et l'évaluation des agents RL.
Cet environnement constitue la base expérimentale sur laquelle s'appuient les contributions présentées dans la suite de cette thèse.

Contribution 2 — Enrichissement automatique des vulnérabilités par classification multi-taxonomie

La capacité à modéliser de manière réaliste le comportement des vulnérabilités est au cœur du simulateur proposé dans cette thèse. Son principe repose sur l'exploitation de métadonnées associées aux vulnérabilités, notamment les taxonomies CWE, CAPEC et ATT&CK, afin de représenter les comportements d'exploitation et de définir les transitions d'état ainsi que les récompenses du POMDP.
Cependant, ces annotations sont encore largement produites manuellement, ce qui entraîne des délais importants et empêche l'intégration rapide des vulnérabilités nouvellement divulguées dans le simulateur. Cette contribution propose donc une approche automatisée fondée sur des modèles de langage (Language Models, LM) afin de prédire ces associations à partir des descriptions textuelles des vulnérabilités, seules informations systématiquement disponibles au moment de leur publication.
Le problème est formulé comme une tâche de classification multi-label à grande cardinalité couvrant les niveaux les plus fins des taxonomies CWE, CAPEC et ATT&CK. À cette fin, cette thèse construit le plus grand jeu de données multi-taxonomie disponible à ce jour (228 000 vulnérabilités) et propose une méthodologie expérimentale unifiée pour l'évaluation de ces tâches. Une approche de classification hiérarchique exploitant la structure des taxonomies est également introduite, permettant d'améliorer les performances tout en limitant le coût d'inférence grâce à une stratégie de représentations partagées entre les différents niveaux de la hiérarchie.
Les résultats montrent que cette approche améliore significativement la qualité des prédictions et généralise efficacement à des vulnérabilités récemment publiées. Elle permet ainsi de produire automatiquement des annotations provisoires pour de nouvelles vulnérabilités et de les intégrer immédiatement dans le simulateur, où elles sont utilisées pour modéliser de manière plus réaliste les comportements d'exploitation et les conséquences des attaques.

Contribution 3 — Apprentissage par renforcement scalable et généralisable pour l'optimisation combinatoire sur graphes

La découverte de chemins d'attaque critiques peut être formulée comme un problème de GCO. Plus généralement, de nombreux problèmes complexes issus de domaines variés peuvent être représentés sous forme de graphes. Cependant, leur nature combinatoire rend les méthodes exactes rapidement inapplicables à grande échelle, comme c'est également le cas pour le problème de découverte de chemins d'attaque considéré dans cette thèse. Les approches heuristiques constituent alors une alternative permettant d'obtenir des solutions efficaces en temps raisonnable, mais leur conception repose généralement sur des règles définies manuellement et difficilement généralisables à de nouveaux scénarios.

Cette contribution s'intéresse à l'apprentissage automatique d'heuristiques pour les problèmes GCO, avec pour objectif principal d'améliorer la scalabilité et la capacité de généralisation des approches fondées sur RL. Contrairement aux méthodes existantes, dont les représentations et espaces d'action restent souvent fortement dépendants de la taille et de la structure des graphes considérés, nous proposons une méthodologie générale permettant aux agents RL d'exploiter des espaces d'action latents continus construits à partir de représentations invariantes aux graphes, notamment via l'utilisation de réseaux de neurones sur graphes (Graph Neural Networks, GNN). L'approche proposée introduit des agents par projection, capables de générer directement une représentation latente de l'action recherchée, puis de la projeter vers l'action valide la plus proche dans l'espace discret. Cette formulation découple la politique apprise de la taille explicite de l'espace d'action, dont la croissance peut être combinatoire avec celle du graphe, permettant ainsi une meilleure capacité de transfert vers de nouvelles instances tout en réduisant le coût d'inférence. La méthodologie est évaluée sur plusieurs problèmes GCO, incluant la découverte de chemins d'attaque critiques, caractérisés par des espaces de décision complexes et fortement croissants. Les résultats démontrent des améliorations significatives en termes de temps d'inférence et de généralisation sur des instances non vues, par rapport aux approches RL existantes, qu'il s'agisse de méthodes discrètes dont les politiques restent dépendantes de la taille de l'espace d'action, ou d'approches latentes itératives nécessitant une évaluation successive des actions candidates. Ces travaux sont accompagnés de `LaGCO-RL`, une bibliothèque unifiée facilitant l'intégration, l'évaluation et la comparaison de méthodes RL appliquées aux problèmes de GCO.
Cette contribution fournit ainsi un cadre général pour la conception d'heuristiques RL plus scalables et transférables, qui est ensuite adapté dans cette thèse au problème spécifique de découverte de chemins d'attaque critiques.

Contribution 4 — Découverte de chemins d'attaque critiques par apprentissage par renforcement généralisable

Cette contribution applique et étend la méthodologie RL pour GCO introduite précédemment au problème central de cette thèse : la découverte de chemins d'attaque critiques. Ce problème présente des défis spécifiques liés à l'hétérogénéité des infrastructures réseau, à la dynamique des environnements considérés et à la complexité des espaces d'action, qui peuvent évoluer fortement selon la topologie, les configurations et les vulnérabilités présentes.
L'approche proposée adapte les agents par projection en construisant des représentations latentes des observations et des actions indépendantes des caractéristiques spécifiques de chaque scénario. Les représentations des infrastructures réseau sont obtenues à l'aide de modèles GNN, tandis que les vulnérabilités sont encodées à partir de leurs descriptions textuelles grâce à des LMs pré-entraînés utilisés comme extracteurs de représentations. Cette formulation permet aux agents RL d'apprendre des stratégies d'attaque transférables entre des environnements présentant des topologies, des tailles de graphes et des ensembles de vulnérabilités différents, sans nécessiter un réentraînement complet pour chaque nouvelle configuration. Elle améliore ainsi simultanément les capacités de généralisation et de passage à l'échelle des agents face à des scénarios cyber complexes.
Une évaluation extensive sur un ensemble varié de scénarios réseau intégrant des vulnérabilités issues de cas réels montre que l'approche proposée améliore significativement la scalabilité des agents RL tout en conservant de bonnes capacités de généralisation vers des environnements non observés. Des expérimentations orientées vers le déploiement démontrent également la

possibilité de transférer les politiques apprises en simulation vers des environnements construits à partir de données issues d'infrastructures réelles ou émulées.
Enfin, les politiques apprises sont comparées à des heuristiques manuelles couramment utilisées pour la priorisation des chemins d'attaque. Les résultats montrent que les agents RL peuvent découvrir des stratégies plus efficaces tout en répondant aux contraintes de scalabilité et d'adaptation nécessaires à une utilisation pratique dans des infrastructures cyber évolutives.

Contribution 5 — Placement de machines virtuelles sécurisé par apprentissage multi-agents

Cette contribution explore l'application des approches de découverte de chemins d'attaque critiques à un problème d'optimisation réseau majeur : le placement de machines virtuelles (Virtual Machine Placement, VMP) dans les infrastructures virtualisées. Alors que les stratégies classiques de placement cherchent principalement à optimiser l'utilisation des ressources et la consommation énergétique, elles peuvent également augmenter la surface d'attaque en favorisant la colocalisation de machines virtuelles et l'exposition aux vulnérabilités de la couche de virtualisation.
Afin d'intégrer explicitement la dimension de sécurité dans ce problème d'optimisation, cette thèse propose une approche par apprentissage multi-agents (Multi-Agent Reinforcement Learning, MARL) combinant un agent attaquant et un agent défenseur. L'agent attaquant exploite le modèle de découverte de chemins d'attaque critiques développé précédemment afin d'évaluer les risques associés à une configuration donnée, tandis que l'agent défenseur apprend à adapter le placement des machines virtuelles pour réduire la surface d'attaque tout en conservant des objectifs de performance et d'efficacité opérationnelle.
Le problème est ainsi formulé comme une optimisation multi-objectifs intégrant à la fois des critères classiques de gestion d'infrastructure et une estimation dynamique du risque basée sur des comportements d'attaque appris.
Les résultats expérimentaux montrent que l'approche proposée permet d'apprendre des stratégies de placement capables d'améliorer simultanément la sécurité et les performances de l'infrastructure. Cette contribution illustre ainsi comment les comportements adverses appris par RL peuvent être transformés en informations exploitables pour guider des mécanismes de défense adaptatifs dans des problèmes d'optimisation réseau plus larges.

Conclusions et perspectives

Cette thèse a étudié la découverte de chemins d'attaque critiques dans les infrastructures réseau à travers la modélisation du comportement des attaquants par RL. L'objectif était de développer des approches capables d'identifier les vulnérabilités jouant un rôle déterminant dans les chemins d'attaque les plus critiques, en tenant compte du contexte de l'infrastructure, des dépendances entre vulnérabilités et de la nature séquentielle des stratégies adverses.
Les contributions proposées couvrent l'ensemble de la chaîne, depuis la conception d'environnements de simulation réalistes et l'enrichissement automatique des connaissances liées aux vulnérabilités, jusqu'au développement d'agents RL capables de généraliser à des scénarios variés et de passer à l'échelle sur des infrastructures complexes. Les résultats obtenus démontrent le potentiel de ces approches pour dépasser les limites des heuristiques manuelles et fournir des mécanismes adaptatifs pour la priorisation des risques et le support à la décision en cybersécurité.

Plusieurs perspectives restent néanmoins ouvertes, notamment l'amélioration continue du réalisme des environnements simulés, le développement de stratégies permettant de limiter davantage l'explosion des espaces d'action, ainsi que l'optimisation des coûts d'exécution pour permettre un déploiement à plus grande échelle. L'adaptation continue des modèles face à l'évolution des infrastructures et des menaces constitue également un enjeu majeur. Enfin, ces travaux ouvrent des perspectives plus larges vers l'exploitation de comportements offensifs appris pour concevoir des mécanismes de défense automatisés et adaptatifs.

Hyperparameter optimization and experimental configuration

This annex summarizes the hyperparameter configurations used throughout the experimental studies presented in this thesis. To facilitate reproducibility, we distinguish between hyperparameters selected through systematic optimization procedures and those specified manually based on preliminary experimentation, domain knowledge, or implementation constraints. Given the diversity of the considered methodologies, application domains, and learning paradigms, hyperparameter settings are reported separately for each study in the corresponding sections of this annex.
All hyperparameter optimization experiments were conducted using the Optuna framework [298]. Unless otherwise stated, optimization was performed using the Tree-structured Parzen Estimator (TPE) algorithm [299] over 25 trials.

A.1 Language models for CWE, CAPEC, and ATT&CK classification

The hyperparameter optimization of LMs was conducted exclusively on each main variant (BERT and ELECTRA). To reduce the computational overhead of K-fold CV during hyperparameter search, we used a single train–validation-test split. The hyper-parameter ranges reported in Table A.1 were explored. The optimization objective was to maximize the validation micro-F1 score, using CWE classification as the reference task for determining the optimal hyperparameters. CWE was prioritized because correctly identifying a CWE may allow manual mapping to CAPEC and ATT&CK techniques, making it the most relevant classification task. Standard hyperparameters—including batch size, learning rate (LR), maximum gradient norm (MGN), and warm-up steps percentage (WU%)—were explored, alongside several loss functions including both standard objectives and imbalance-aware variants: Asymmetric Loss (ASL), Focal Loss (FL), PolyLoss (Poly), Distribution-Balanced Loss (DB), Distribution-Balanced Focal Loss (DB-Focal), Binary Cross-Entropy (BCE), and Weighted BCE (WBCE).

Table A.1: Search space for hyperparameters and loss functions (bold = best)

	Batch Size	LR	MGN	WU%	Losses
BERT	8, 16, 32, **64**	1e-5, 2e-5, **5e-5**	**0.25**, 0.5, 0.75	**1**, 5, 10	ASL, FL, Poly, DB, **DB-Focal**, BCE, WBCE
ELECTRA	**8**, 16, 32, 64	1e-5, 2e-5, **5e-5**	0.25, 0.5, **0.75**	**1**, 5, 10	ASL, FL, **Poly**, DB, DB-Focal, BCE, WBCE

The hyperparameters identified for each model were applied consistently across all classification taxonomies. Moreover, the hyperparameters found for BERT were used for all its variants, ensuring a fair comparison among them.
Other, less critical hyperparameters were kept fixed across all experiments to ensure consistency and reproducibility:

- *Gradient accumulation steps*: 1

- *Weight decay*: 0.01
- *Fixed seed*: 42
- *Sigmoid threshold*: 0.5
- *Lowercase*: True
- *Padding*: max length
- *Truncation*: True
- *Maximum sequence length*: 512 tokens

The number of epochs was set to 1 for the main experiments due to the large dataset size, and proportionally increased for slave models (smaller datasets) or smaller temporal splits, up to a maximum of 30 epochs (see Section 4.5.2). All remaining hyperparameters were left at their library default values.

A.2 RL and auto-encoder models for GCO

	Learning Rate	Batch Size	γ	N Steps	Ent Coef	Max Grad Norm	Tau	Target Update Interval	Num Layers	NN Channels	Out Channels	Activation
Discrete PPO	0.00001, 0.0001, **0.001**	32, **64**, 128	**0.9**, 0.95, 0.99	512, 1024, **2048**	**0.01**, 0.1, 0.2	0.1, 0.3, **0.5**	–	–	–	–	–	–
Projection PPO	0.00001, 0.0001, **0.001**	32, **64**, 128	**0.9**, 0.95, 0.99	512, 1024, **2048**	**0.01**, 0.1, 0.2	0.1, 0.3, **0.5**	–	–	–	–	–	–
Iterative IDQN	0.00001, 0.00005, **0.0001**	**32**, 64, 128	0.9, **0.95**, 0.99	1, 3, **5**	–	–	0.01, **0.05**, 0.1	2000, 5000, **10000**	–	–	–	–
GAE	0.00005, 0.0001, 0.001, **0.01**, 0.05	16, **32**, 64	–	–	–	–	–	–	**2**, 3	16, **32**	**16**, 32, 64	LeakyReLU, **ReLU**, null

Table A.2: Hyperparameter ranges explored for each method with best configurations in **bold**.

The hyperparameter optimization of the RL and autoencoder models used in the experiments of Chapter 5 was conducted exclusively on the *TSP* benchmark. This choice was made to reduce the computational cost of the optimization process and limit the size of the search space, which would otherwise increase substantially if the benchmark dimension was also included.
Table A.2 reports the hyperparameter search space explored for each optimized model, along with the best configuration identified. Regarding discrete solutions, this process has been performed only on the P-Discrete-M variant for simplicity, and hyperparameters were used across all other discrete variants.
The following additional hyperparameters were tuned manually and selected based on empirical evaluation:

- **Node embeddings dimensionality:** 16
- **Nearest neighbor search (action space):** FAISS Flat Index
- **Distance metric (action space search):** Cosine similarity
- **Action vector processing:**

  - No normalization in the iterative approach
  - Z-score normalization in the projection approach (ensures projection outputs remain within a suitable range for policy-based methods)
- **Action space bounds:** Extended by a margin of $(-1, +1)$ (to the z-score bounds) relative to the empirical latent space range discovered across the training instances per benchmark
- **Episode length:** To maintain consistency across varying problem scenarios, the episode length is scaled proportionally to the instance complexity. Concretely, it is defined as a coefficient multiplied by the number of decision variables (e.g., number of VMs in *Placement*), ensuring that agents are given sufficient interaction steps with respect to scenario size:
  - Default: $1\times$ scenario size
  - Attack-Path / traffic scenarios: $3\times$ scenario size
  - OSPF engineering scenarios: $2\times$ scenario size
  - Maxcut scenarios: $2\times$ scenario size
- **Reward normalization:** Min-max normalization (Stable-Baselines3 default)
- **Feature normalization:**
  - Node features are normalized independently using min-max scaling based on their respective ranges within the graph, ensuring feature scale independence
  - L1 normalization applied to language model embeddings when used as features, as ranges can hardly be determined
- **Scenario switching during training (only for *V* experiments):**
  - Interval: every 50 episodes
  - Strategy: random selection
- **Policy architecture:**
  - Hidden layers: [128, 64]
  - Subsequent output layer proportional to the action space and agent formulation used
  - Activation: LeakyReLU
  - Optimizer: Adam ($\epsilon = 10^{-7}$, weight decay $= 10^{-4}$, AMSGrad disabled)
- **Graph Autoencoder:**
  - Final layer: no activation function, with normalization enabled to ensure properly scaled embeddings
  - Loss functions (all adjusted to have the same scale, and weights all set to 1):
    - Continuous features: Mean Squared Error
    - Adjacency matrix: Contrastive loss ($\tau = 0.5$)
    - Binary features: Binary Cross-Entropy with logits
    - Multi-categorical features: Cross-Entropy loss
- **Observation design:** Latent-observation agents use a fixed-dimensional representation combining pooled graph embeddings (mean, max, min, and sum pooling) with global structural descriptors (number of nodes, edges, average degree, and density). For projection-based agents, an additional pooled representation of the latent action space is included. Padding-based baselines instead employ benchmark-specific fixed-size observations encoding the relevant node-, edge-, graph-, or path-level features while preserving consistency

with the information available to GNN-based approaches.

Seeds. The random seeds used in the experiments are defined as follows. A seed of 42 is used for the scenario generation, initial experimental setup, and splitting strategies. For the generalization study, five independent runs per strategy and RL solution are performed with the following seeds used during training: 42, 100, 123, 200, and 300. For the testing phase, seeds are generated based on the number of episodes per scenario instance, ensuring a one-to-one correspondence and symmetry across test episode identifiers of several scenario instances. The resulting sequence of test seeds is: 42, 100, 123, 200, followed by increments of 100 up to 5000 (i.e., 400, 500, 600, ..., 5000). The same sequence is used for assessing the distribution of action-selection time.
All other hyperparameters were set to their respective library default values and are provided in the accompanying configuration files.

A.3 RL and auto-encoder models for attack path prioritization

Table A.3: Search space for on-policy RL hyperparameters (bold = best)

	Learning Rate	Batch Size	γ	N Steps	Ent Coef	Value Coef	Max Grad Norm	GAE(λ)
PPO	**0.00001**, 0.0001, 0.001	32, 64, **128**	**0.9**, 0.95, 0.99	512, **1024**, 2048	0.01, **0.1**, 0.2	–	0.1, **0.3**, 0.5	–
TRPO	0.0001, **0.001**, 0.01	**64**, 128, 256	0.9, **0.95**, 0.99	1024, 2048, **4096**	–	–	–	**0.9**, 0.95, 0.99
A2C	0.0001, **0.001**, 0.01	–	0.9, **0.95**, 0.99	5, 10, **20**	**0.01**, 0.1, 0.2	0.5, **1.0**, 1.5	0.3, **0.5**, 0.7	–

Table A.4: Search space for off-policy RL and GAE hyperparameters (bold = best)

	Learning Rate	Batch Size	γ	Tau	Buffer Size	Target Ent.	Pol. Noise	Layers	Channels	Out Ch.	Activation
SAC	**0.00001**, 0.0001, 0.001	32, 64, **128**	0.9, **0.95**, 0.99	**0.005**	**100000**, 500000, 1000000	-2, -1.5, **-1.0**	–	–	–	–	–
TD3	**0.0001**, 0.001, 0.01	64, 128, **256**	0.9, **0.95**, 0.99	**0.001**, 0.005, 0.01	**100000**, 500000, 1000000	–	0.1, **0.2**, 0.3	–	–	–	–
TQC	**0.0001**, 0.001, 0.01	**64**, 128, 256	0.9, 0.95, **0.99**	0.001, **0.005**, 0.01	100000, **500000**, 1000000	–	–	–	–	–	–
GAE	0.000001, 0.00001, 0.00005, 0.0001, 0.001, **0.01**	16, **32**, 64, 128	–	–	–	–	–	**2**, 3, 4	**16**, 32, 64	16, 32, **64**	LeakyReLU, ReLU, **null**

For the critical attack path discovery benchmark investigated in Chapter 6, hyperparameter optimization was performed separately for each RL algorithm, while a common set of GAE hyperparameters was optimized and subsequently used across all RL methods. Table A.3 and

Table A.4 summarize the search ranges explored for each algorithm during the optimization process.
The objective used for the GAE is the minimization of the reconstruction loss on the validation set. For RL algorithms, the objective used is the maximization of the AUC of the validation goal scores (averaging across the three experimental goals). The following hyperparameters are instead kept constant throughout the study for the algorithms:

- **NN layer sizes**: For all DRL algorithms, the sizes of the DNN layers were set to [256, 128, 64]. For RecurrentPPO, an LSTM layer was added of size [32].
- **Activation function**: LeakyReLU was used as the activation function for the DNN.
- **Optimizer**: The Adam optimizer was used for both DRL algorithms and GAE. A value of 1×10^{-7} was set for the epsilon parameter. The weight decay was fixed at 1×10^{-4}.
- **GAE architecture**: The GAE consisted of one NNConv layer as the first layer. This ensures that edge feature vectors are used in the computation of the embeddings. Subsequent layers are GCNConv layers.

All other hyperparameters, not explicitly mentioned, were left at their default values as defined by the respective libraries.

Graph combinatorial optimization benchmarks

This appendix provides complementary information on the GCO benchmarks used in Chapter 5, including detailed descriptions of their underlying logic, domain-randomization procedures, and the manually designed heuristics employed for extensive exploration sweeps.

B.1 Logic of GCO benchmarks

This section describes the benchmark environments used in the experimental study of Section 5.6 in terms of extended logic and dynamics, their graph representations, and the score used for solution evaluation.

Traveling salesman problem. Construct a minimum-length tour that visits each city exactly once (hard constraint).

- **Parameters:** Number of cities N and coordinate range defining (x, y) positions.
- **Graph:** Nodes represent cities, and edges encode distances. The graph is fully connected but sparsified by retaining the $K = 10$ NNs per node to avoid a fully connected graph that would find issues with GNN embeddings.
- **Node features:** Visited flag $\in \{0, 1\}$ and spatial coordinates (x, y).
- **Edge features:** Euclidean distances between cities.
- **Reset:** A starting city is randomly selected and marked as visited, while all others remain unvisited.
- **Step helper:** The selected next city is appended to the tour, removed from the action space, and marked as visited.
- **Termination:** The episode ends once all N cities have been visited.
- **Reward:** Total length of the constructed tour in terms of the sum of Euclidean distances, normalized as a relative score. If incomplete, remaining cities are inserted (padding) using a worst-case heuristic (e.g., farthest insertion with respect to last city) to estimate final cost, ensuring the agent prioritizes the completion of the tour.
- **Score:** Zero if the tour does not cover all cities exactly once; otherwise, the score is the tour length normalized between best and worst (empirically) known solutions.

Minimum vertex cover. Select the smallest subset of nodes such that every edge is incident to at least one selected node (hard constraint).

- **Parameters:** Number of nodes N and edge probability p.
- **Graph:** Erdős–Rényi random graph where edges must be covered by selected nodes.
- **Node features:** Binary selected flag $\in \{0, 1\}$.
- **Edge features:** Binary covered flag $\in \{0, 1\}$.

- **Reset:** All nodes are initialized as unselected and all edges as uncovered.
- **Step helper:** Selecting a node marks it as used, removes it from the action space, and updates newly incident edges as covered.
- **Termination:** The episode ends when all edges are covered or a predefined cutoff is reached.
- **Reward:** Number of newly covered edges, with a penalty proportional to the size of the selected covered nodes (normalized to a relative score).
- **Score:** Zero if coverage is incomplete; otherwise, normalized cover size relative to best and worst (empirically) known solutions.

Maximum cut. Partition nodes into two sets to maximize the total weight of edges crossing between partitions.

- **Parameters:** Number of nodes N and edge weight range.
- **Graph:** Weighted graph with weights randomly sampled across every pair of nodes. The graph is sparsified by retaining $K = 10$ NN per node to avoid a fully connected graph that would find issues with GNN embeddings.
- **Node features:** Partition assignment $\in \{0, 1\}$.
- **Edge features:** Edge weight as scalar value.
- **Reset:** All nodes are initially assigned to partition 0.
- **Step helper:** A node is moved to the complementary partition (inspired by [144]), updating the cut structure accordingly.
- **Termination:** The episode ends when the (empirically) known best cut is reached (only during training) or a cutoff is exceeded.
- **Reward:** Incremental change in cut value based on whether edges become crossing or internal (normalized to a relative score).
- **Score:** Normalized cut value between best and worst (empirically) known solutions.

Virtual machine placement. Assign VMs to PMs to jointly optimize utilization, energy efficiency, packing efficiency, load balancing, and security risk (representing the attack surface), all under capacity constraints.

- **Parameters:** Number of VMs, PMs, tenants; VM resource demands (CPU, MIPS, RAM, storage); PM capacities; traffic, latency, energy, and security characteristics.
- **Graph:** Nodes represent VMs and PMs; edges encode allocation (VM $\rightarrow$ PM), VM-to-VM traffic, and PM-to-PM latency.
- **Node features:** VM nodes encode resource requirements, while PM nodes encode utilization and power consumption.
- **Edge features:** Traffic demands and latency values.
- **Reset:** System initialized with the (empirically) known worst allocation to ensure the exploration of a large set of positive action sequences.
- **Step helper:** A (VM, PM) assignment is validated against capacity constraints and applied if feasible.
- **Termination:** Episode ends when an (empirically) known optimal allocation is reached (only during training) or a cutoff is exceeded.

- **Reward:** Weighted variation of normalized metrics plus migration cost.
- **Score:** Weighted sum of the final normalized metric values.

OSPF engineering. Adjust link weights to minimize congestion by reducing maximum link utilization under routing constraints.

- **Parameters:** Number of nodes, capacity range, traffic range, weight bounds, graph distribution, and Equal-cost multi-path routing (ECMP) routing flag.
- **Graph:** Spanning graph with undirected communication edges and directed traffic flows.
- **Node features:** Aggregated incoming and outgoing traffic (to avoid an empty feature vector, needed for GNNs).
- **Edge features:** Capacity, utilization, weight, and traffic load.
- **Reset:** Initialized with the (empirically) known worst-case weight configuration to ensure the exploration of a large set of positive action sequences.
- **Step helper:** Modify a link weight within bounds and recompute routing and utilization (according to ECMP usage).
- **Termination:** Episode ends when the (empirically) known optimal configuration is reached (only during training) or a cutoff is exceeded.
- **Reward:** Change in maximum link utilization.
- **Score:** Reduction in maximum link utilization relative to the initial configuration.

Traffic engineering. Route traffic demands through feasible paths to minimize congestion while respecting capacity constraints. This represents a variant of OSPF engineering without relying on OSPF as an intermediary helper.

- **Parameters:** Number of nodes, capacity range, traffic characteristics, graph distribution, and maximum path length (filtering set of communication paths to be used as solutions to a maximum, and avoiding solutions unlikely to be optimal).
- **Graph:** Spanning graph with communication and traffic edges.
- **Node features:** Aggregated incoming and outgoing traffic (to avoid an empty feature vector, needed for GNNs).
- **Edge features:** Capacity, utilization, and traffic load.
- **Reset:** Initialized with the (empirically) known worst-case allocation configuration to ensure the exploration of a large set of positive action sequences.
- **Step helper:** Assign traffic demands to feasible paths while respecting capacity constraints.
- **Termination:** Episode ends when (empirically) known optimal routing is achieved (only during training) or a cutoff is exceeded.
- **Reward:** Change in maximum link utilization.
- **Score:** Reduction in maximum link utilization relative to the initial configuration.

B.2 Domain randomization

As described in Section 6.5.3, the generalization study of RL-GCO solutions relies on 101 generated instances obtained by randomly sampling the parameters of the environment within

predefined ranges. These ranges are designed to induce variability across instances and are reported below; all other parameters are held constant as indicated.

Traveling salesman problem
`num_cities`:[10,100]
`max_coord`:[100,1000]

Maximum cut
`num_nodes`:[10,100]
`max_weight`:[10,100]

Minimum vertex cover
`num_nodes`:[10,50]
`edge_prob`:[0.1,0.4]

OSPF engineering
`num_nodes`:[10,30]
`communication_edge_ratio`:[0.1,0.3]
`non_zero_traffic_ratio`:[0.1,0.3]
`min_capacity`:[10,100]
`max_capacity`:[500,1000]
`max_traffic`:[25,50]
`graph_distribution`: `spanning tree`
`min_weight`: 1
`max_weight`: 5

Traffic engineering
`num_nodes`:[10,25]
`communication_edge_ratio`:[0.1,0.2]
`non_zero_traffic_ratio`:[0.1,0.2]
`min_capacity`:[10,100]
`max_capacity`:[500,1000]
`max_traffic`:[25,50]
`max_path_len`: 4
`graph_distribution`: `spanning tree`

Virtual machine placement
`n_vms`:[10,50]
`n_pms`:[10,50]
`n_tenants`:[2,5]
`vm_vuln_prob_min`:[0.01,0.1]
`vm_vuln_prob_max`:[0.2,1.0]
`pm_escape_prob_min`:[0.01,0.1]
`pm_escape_prob_max`:[0.2,0.33]
`pm_capacity_memory_min`:[32,64]
`pm_capacity_memory_max`:[128,512]
`vm_demand_memory_min`:[1,8]
`vm_demand_memory_max`:[16,24]
`pm_capacity_storage_min`:[100,500]
`pm_capacity_storage_max`:[1000,5000]
`vm_demand_storage_min`:[10,50]
`vm_demand_storage_max`:[60,100]
`pm_capacity_pe_min`:[8,16]
`pm_capacity_pe_max`:[32,128]
`vm_demand_pe_min`:[2,4]
`vm_demand_pe_max`:[8,16]
`pm_capacity_mips_min`:[1000,5000]
`pm_capacity_mips_max`:[10000,50000]
`vm_demand_mips_min`:[100,500]
`vm_demand_mips_max`:[750,1000]
`latency_min`:[0.1,1.0]
`latency_max`:[2,8]
`p_idle`:[50,100]
`p_peak`:[150,300]
`coefficients`: 1
`min_traffic`: 1
`max_traffic`: 10
`traffic_density`: 0.8

Critical attack path discovery
`n_nodes`:[10,20]
`n_vulns_per_node`:5
`vulns_overlap`:[0.0,0.1]
`p_data_present`:[0.6,0.9]
`p_feature_visible`:[0.5,0.7]
`p_recon`:[0.2,0.4]
`p_detection`:[0.05,0.2]
`goal`: control
`fully_connected`: true
`outcome_selection`: false

Scalability experiments in Section 5.6.2 present an evolving set of values for the main dimension (number of nodes) and the same ranges for secondary features.

B.3 Empirical heuristic exploration

Scenario instances are first exhaustively explored using heuristic-guided search to estimate empirical upper and lower performance bounds. These bounds are then used to define a shared reward function across benchmarks, computed as the normalized relative improvement over the estimated worst- and best-case scores. This normalization provides a consistent learning signal across scenarios, although rewards are not strictly bounded and may exceed 1.
During training, the best-known score is additionally employed as a termination criterion to accelerate convergence, but this mechanism is disabled during testing to avoid introducing supervision.
The number of exploration sweeps was determined through preliminary experiments (10,000 for TSP and MVC, 5,000 for Placement, and 2,000 for MaxCut, OSPF, and Traffic). The Attack-Path scenario is excluded from this procedure, as its bounds can be directly derived from the number of nodes. The heuristics used for exploration were initially generated using a LLM (GPT-5.4) and subsequently refined through manual validation.

- *TSP*: We generate random tours and iteratively refine them using stochastic 2-opt moves. Improvements (or degradations) are sampled from candidate swaps, biased by their impact on tour length. This allows simultaneous estimation of both best and worst-case tour costs.
- *MinVertex*: A greedy-stochastic strategy is used to iteratively select nodes covering the largest number of uncovered edges. At each step, selection is randomized among the top-k candidates to maintain diversity. A pruning phase removes redundant nodes to refine solutions. The best cover size is tracked across multiple sweeps. The worst is set to the overall number of nodes.
- *MaxCut*: Random graph partitions are initialized and improved via stochastic local search. Node flips that increase the cut value are identified, and one is selected probabilistically based on gain. This process continues until no further improvements are possible. The maximum cut value observed across samples defines the upper bound. The worst value is set to 0.
- *Placement*: Allocations are constructed using heuristics that either maximize or balance resource utilization across physical machines. Initial solutions are built by prioritizing high-demand VMs, followed by local refinements through stochastic reassignment. Both best and worst allocations are identified through repeated sweeps.
- *OSPF*: OSPF link-weight settings are explored through stochastic search over candidate configurations. Each sweep starts either from a random solution or from a perturbed elite solution sampled from small best/worst pools, allowing intensified search without losing diversity. Neighboring configurations are obtained by randomly modifying a subset of

link weights, and are selected with a bias toward lower and higher utilization to estimate both worst and best. Previously explored configurations are excluded.

- *Traffic*: Same idea as *OSPF* but optimizing path allocations and not edge weights.

Attack path benchmark

This annex describes the characteristics of the simulated scenarios used in the critical attack path discovery experiments presented in Section 6.5, as well as the complexity metric used to quantify the difficulty of the generated scenarios.

C.1 Domain randomization

To better reflect the complexity of real-world environments, the scenarios incorporate several additional features beyond those considered in the simpler benchmark settings of Section 5.6. These features are grouped into two categories: graph-related parameters and cyber-terrain parameters. Table C.1 provides a detailed description of each feature and its role in defining a scenario, together with the corresponding service and vulnerability allocations.
For each generated scenario, feature values are sampled from the ranges specified in the table. Depending on the feature, sampling is either derived from the CVSS attributes of the associated vulnerabilities or performed randomly at the scenario level, as indicated in the corresponding table column.

C.2 Complexity metric

The complexity categorization column of Table C.1 further specifies how each feature contributes to the overall scenario difficulty, indicating whether increasing its value results in a higher (ascending feature) or lower (descending feature) complexity score.
This complexity score is defined as in Equation C.1. This score is subsequently used to support scenario partitioning and evaluation strategies based on environment complexity.

$$\begin{aligned} \textit{Complexity Score} = & \sum_{i \in \text{Ascending}} w_i \cdot \text{Norm}_{[0,1]}(f_i) \\ & - \sum_{j \in \text{Descending}} w_j \cdot (1 - \text{Norm}_{[0,1]}(f_j)) \end{aligned} \tag{C.1}$$

Each feature range has been normalized using min-max scaling to the range $[0, 1]$ in order to ensure consistent scaling across all features. The sums of components determining the score can be adjusted with appropriate weights to emphasize certain factors over others in determining the overall complexity. In our experiments, all these weights were set to 1, giving equal importance to each feature.

Feature	Description	Range used	Sampling strategy	Complexity categorization
Number of nodes	Number of graph nodes in the scenario	[10, 100]	Random	Ascending
Number of services per node	Number of services sampled per node	[1, 3]	Random	Ascending
Homogeneity value	Real value determining the amount of homogeneity present in the scenario in service assignment	[0.05, 0.33]	Random	Descending
Percentage of nodes per type	Proportion of nodes belonging to each category (e.g., *Windows*) in the generated scenario	[0.05, 0.95]	Random	N/A
Firewall incoming rule probability	Probability that for every service of every node, there is a blocking rule incoming	[0, 0.33]	Random	Ascending
Firewall outgoing rule probability	Probability that for every service of every node, there is a blocking rule outgoing	[0, 0.33]	Random	Ascending
Probability of neighbor information release	Probability applied to each vulnerability that reveals neighboring node IDs, determining how many nodes will be discovered exploiting this vulnerability	[0.2, 0.8]	Determined based on the *Confidentiality Impact* of the vulnerability	Descending
Probability of data presence	Probability that each node has data to collect (applied only if the node has at least one vulnerability allowing "Collection" outcome)	[0.2, 0.8]	Random	Descending
Probability of partial visibility on nodes	Probability that each node will be partially visible once owned (applied only if the node has at least one vulnerability allowing "Discovery" outcome)	[0.2, 0.8]	Random	Ascending
Probability of need for escalation	Probability that the node, when owned, starts with a "basic user" level and not "root" level (applied only if the node has at least one vulnerability allowing "Privilege Escalation" outcome)	[0.2, 0.8]	Random	Ascending
Probability of service shutdown	Probability that every service of every node will start with a shutdown status	[0, 0.2]	Random	Ascending
Success rate	Probability representing the success rate for every vulnerability, used to simulate failure of vulnerability exploitation	[0.6, 1]	Determined based on the *Attack Complexity* of the vulnerability	Descending
Node value	Value assigned to each node, representing its importance in the scenario	[1, 100]	Random	N/A

Table C.1: Description of scenario features, including information about which range has been sampled during scenario generation, with which sampling strategy, and the resulting complexity categorization.

Virtual machine placement benchmark

This annex describes the characteristics of the simulated scenarios used in the VMP experiments presented in Section 7.6, as well as the performance and attack surface metrics targeted for optimization.

D.1 Domain randomization

Table D.1 outlines the set of configurable parameters used to generate the scenarios evaluated in the experimental section. Each feature is characterized by its functional description, sampling range, the strategy used for its selection, and its influence on the scenario complexity metric used for training/test splitting of scenario sets. The same parameters have been used for the generation of the larger-scale scenarios of Section 7.6.6, but increasing the number of VMs and PMs.

D.2 Performance and security metrics

This section includes the metrics contributing to the different components of the reward signal provided to the defender agent of the VMP problem. For each metric, we provide its mathematical formulation and a brief explanation of the rationale behind it.

- **Resources Utilization (RU)**: Encourages either full utilization or idleness of PMs by penalizing usage near 50% (half-loaded). The utilization RU is calculated by summing the minimum between used and available resources across all PMs resources ($\mathcal{R}$), normalizing by total available resources.

$$RU = 1 - \frac{2 \sum_{pm \in \mathcal{PM}\text{-set}} \sum_{r \in \mathcal{R}} \min(\text{used } r, \text{total } r - \text{used } r)}{\sum_{pm \in \mathcal{PM}\text{-set}} \sum_{r \in \mathcal{R}} \text{total } r}$$

- **Latency minimization (L)**: Measures intra-tenant latency by averaging the shortest path latency for all VM pairs inside each tenant of the set $\mathcal{T}$, normalized by the number of pairs.

$$L = \frac{1}{|\mathcal{T}|} \sum_{tenant \in \mathcal{T}} \frac{\sum_{i,j} \text{shortest_path_latency}(VM_i, VM_j)}{\binom{n_{tenant}}{2}}$$

- **Energy consumption (E)**: Models energy consumption of PMs based on resource utilization RU_t. Idle PMs consume P_{idle}, while active PMs consume a power proportional to utilization. The energy consumption of PMs is linearly approximated using the values of P_{start} and P_{max} to interpolate between idle and full-load conditions.

$$E_{PM} = \begin{cases} P_{\text{idle}}, & \text{if } RU_t = 0 \\ (P_{\text{max}} - P_{\text{start_idle}}) \cdot RU_t + P_{\text{start_idle}}, & \text{if } 0 < RU_t \leq 1 \end{cases}$$

Feature	**Description**	**Sampling range**	**Sampling strategy**	**Complexity trend**
Number of VMs	Total number of virtual machines instantiated in the scenario	[8, 15]	Random	Ascending
Number of PMs	Total number of physical machines available in the scenario	[7, 15]	Random	Ascending
Number of Tenants	Distinct tenant entities owning VMs in the environment	[2, 5]	Random	Ascending
Services per VM	Number of network services assigned to each VM	[1, 2]	Random	Ascending
Replica Pool Size	Average number of duplicated VMs across the scenario	[0, 2] (Replica Study) / [0] (Others)	Random	Ascending
Bandwidth per Edge	Communication bandwidth assigned to each network edge (in Mbps)	[10, 100]	Random	N/A
Average Mesh Connectivity	Probability of edge existence between any pair of PMs in a mesh topology	[0.1, 0.5]	Random	Descending
Tenant–VM Allocation	Strategy used to assign VMs to tenants	N/A	Random	N/A
VM/PM Type Assignment	Assignment of machine types to virtual or physical machines	N/A	Random	N/A
Firewall Incoming Rule Probability	Likelihood that an inbound firewall rule blocks each service on a node	[0, 0.33]	Random	Ascending
Firewall Outgoing Rule Probability	Likelihood that an outbound firewall rule blocks each service on a node	[0, 0.33]	Random	Ascending
Data Presence Probability	Likelihood that a node contains collectible data (conditional on "Collection" exploitability)	[0.2, 0.8]	Random	Descending
Partial Visibility Probability	Likelihood that an owned node reveals only partial topology (conditional on "Discovery" exploitability)	[0.2, 0.8]	Random	Ascending
Privilege Escalation Requirement	Likelihood that a node is initially accessible with basic privileges (conditional on "Privilege Escalation" exploitability)	[0.2, 0.8]	Random	Ascending
Service Shutdown Probability	Likelihood that a node's service starts in an inactive (shutdown) state	[0, 0.2]	Random	Ascending
Node Value	Importance score assigned to each node in the scenario	[1, 100]	Random	N/A

Table D.1: Scenario configuration features, including parameter descriptions, sampling ranges, selection strategies, and their implications for scenario complexity.

$$E = \frac{1}{|\mathcal{PM}\text{-set}|} \sum_{\text{PM} \in \mathcal{PM}\text{-set}} E_{PM}$$

- **Isolation costs (I)**: Calculates the cost of isolation decisions taken for VMs. For each isolation action a in the set of actions $\mathcal{A}$, the isolation cost function $C_{\text{isolation}}(a)$ is applied.

$$I = \sum_{a \in \mathcal{A} \text{ if isolation}(a)} C_{\text{isolation}}(a)$$

- **Migration costs (M)**: Measures the cost of VM migrations proportional to the shortest path bandwidth between source and destination PMs, normalized by the maximum observed shortest path latency.

$$M = \sum_{a \in \mathcal{A} \text{ if migration}(a)} \frac{\text{sp}_{\text{PMsrc,PMdest}}}{\text{sp}_{\text{max}}}$$

- **Replica-level Difference (RD)**: Represents the proportion of VM replicas introduced from the replica pool.

$$RD = \frac{\text{\# of VMs introduced}}{\text{\# of VMs in replica pool}}$$

- **Attack surface Score (S)**: Measures the number of vulnerabilities selected being exploitable, used to understand the defender's capability in making attacker-selected vulnerabilities unexploitable.

$$S = \frac{\text{\# of vulnerabilities successful}}{\text{\# of vulnerabilities selected by attacker}}$$

- **Aggregated scores**: The following two aggregated score differences have been used to represent the variation in scores after the reconfiguration.

$$\text{Performance Score } P = w_{RU} \cdot \Delta RU + w_L \cdot \Delta L + w_E \cdot \Delta E$$

$$\text{Weighted Sum Score} = \frac{w_P}{3} \cdot \Delta P + w_S \cdot \Delta S$$

Some performance metrics (RD, I, and M) are budget-oriented indicators that quantify the amount of resources introduced into the environment. These do not capture relative improvement but rather reflect the deployment cost of the solution. In contrast, some performance metrics (RU, L, and E) focus on performance differences before and after reconfiguration and are therefore considered in the computation of the performance score. Budget-oriented metrics are instead separated and represented through bar plots in the experimental section to illustrate the extent of resource usage.

BIBLIOGRAPHY

[1] Pratyusa Manadhata and Jeannette M Wing. Measuring a system's attack surface. Technical report, 2004. No. CMUCS04102.

[2] MingJian Tang, Mamoun Alazab, and Yuxiu Luo. Big data for cybersecurity: Vulnerability disclosure trends and dependencies. *IEEE Transactions on Big Data*, 2019.

[3] Otgonpurev Mendsaikhan, Hirokazu Hasegawa, Yukiko Yamaguchi, and Hajime Shimada. Automatic mapping of vulnerability information to adversary techniques. In *Proceedings of the Fourteenth International Conference on Emerging Security Information, Systems and Technologies* (*SECURWARE*), 2020.

[4] Ruchi Sharma, Ritu Sibal, and Sangeeta Sabharwal. Software vulnerability prioritization using vulnerability description. *International Journal of System Assurance Engineering and Management*, 2021.

[5] Christian Fruhwirth and Tomi Mannisto. Improving cvss-based vulnerability prioritization and response with context information. In *2009 3rd International Symposium on Empirical Software Engineering and Measurement*, 2009.

[6] Vladislav D. Veksler, Norbou Buchler, Blaine E. Hoffman, Daniel N. Cassenti, Char Sample, and Shridat Sugrim. Simulations in cyber-security: A review of cognitive modeling of network attackers, defenders, and users. *Frontiers in Psychology*, 2018.

[7] Cynthia Phillips and Laura P. Swiler. A graph-based system for network vulnerability analysis. In *Proceedings of the 1998 Workshop on New Security Paradigms*, 1998.

[8] Onur Sheyner, Joshua Haines, Somesh Jha, Richard Lippmann, and Jeannette M. Wing. Automated generation and analysis of attack graphs. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, 2002.

[9] Kyle Ingols, Richard Lippmann, and Keith Piwowarski. Practical attack graph generation for network defense. In *Proceedings of the 22nd Annual Computer Security Applications Conference* (*ACSAC'06*), 2006.

[10] Xinming Ou, Sudhakar Govindavajhala, and Andrew W. Appel. MulVAL: A logic-based network security analyzer. In *14th USENIX Security Symposium* (*USENIX Security 05*), 2005.

[11] Paul Ammann, Jason Pamula, Richard Ritchey, and John Street. A host-based approach to network attack chaining analysis. In *Proceedings of the 21st Annual Computer Security Applications Conference* (*ACSAC'05*), 2005.

[12] Alaa T. Al Ghazo and Ratnesh Kumar. Critical attacks set identification in attack graphs for computer and scada/ics networks. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 2023.

[13] Alaa T. Al Ghazo and Ratnesh Kumar. Identification of critical-attacks set in an attack-graph. In *2019 IEEE 10th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference* (*UEMCON*), 2019.

[14] Hui Xie, Kun Lv, and Changzhen Hu. An improved monte carlo graph search algorithm for optimal attack path analysis. In *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/ 12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*, 2018.

[15] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 1996.

[16] Huynh Phuong Thanh Nguyen, Kento Hasegawa, Kazuhide Fukushima, and Razvan Beuran. Pengym: Realistic training environment for reinforcement learning pentesting agents. *Comput. Secur.*, 2025.

[17] Mohamed C. Ghanem and Thomas M. Chen. Reinforcement learning for intelligent penetration testing. In *2018 Second World Conference on Smart Trends in Systems, Security and Sustainability (WorldS4)*, 2018.

[18] David Ackley and Michael Littman. Generalization and scaling in reinforcement learning. *Advances in neural information processing systems*, 1989.

[19] Natalia Vesselinova, Rebecca Steinert, Daniel F. Perez-Ramirez, and Magnus Boman. Learning combinatorial optimization on graphs: A survey with applications to networking. *IEEE Access*, 2020.

[20] Sahil Manchanda, Akash Mittal, Anuj Dhawan, Sourav Medya, Sayan Ranu, and Ambuj Singh. Gcomb: Learning budget-constrained combinatorial algorithms over billion-sized graphs. In *Advances in Neural Information Processing Systems*, 2020.

[21] Yang Yu. Towards sample efficient reinforcement learning. In *IJCAI*, 2018.

[22] Wenshuai Zhao, Jorge Peña Queralta, and Tomi Westerlund. Sim-to-real transfer in deep reinforcement learning for robotics: a survey. In *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*, 2020.

[23] Tyler Cody. A layered reference model for penetration testing with reinforcement learning and attack graphs. In *2022 IEEE 29th Annual Software Technology Conference (STC)*, 2022.

[24] George E. Monahan. State of the art—a survey of partially observable markov decision processes: Theory, models, and algorithms. *Management Science*, 1982.

[25] Franco Terranova, Sana Rekbi, Abdelkader Lahmadi, and Isabelle Chrisment. Multi-taxonomy vulnerability classification with hierarchically finetuned language models. In *Proceedings of the 23rd Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA '26)*, 2026.

[26] Franco Terranova, Guillermo Bernardez, Albert Cabellos-Aparicio, Nina Miolane, and Abdelkader Lahmadi. Projecting latent rl actions: Towards generalizable and scalable graph combinatorial optimization, 2026. Preprint arXiv:2605.19721.

[27] Franco Terranova, Abdelkader Lahmadi, and Isabelle Chrisment. Scalable and generalizable rl agents for attack path discovery via continuous invariant spaces. In *2025 28th International Symposium on Research in Attacks, Intrusions and Defenses (RAID)*, 2025.

[28] Franco Terranova, Abdelkader Lahmadi, and Isabelle Chrisment. Leveraging deep reinforcement learning for cyber-attack paths prediction: Formulation, generalization, and evaluation. In *Proceedings of the 27th International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2024)*, 2024.

[29] Franco Terranova, Noura Limam, Abdelkader Lahmadi, Isabelle Chrisment, and Raouf Boutaba. When performance meets security: Reducing attack paths in virtual machine placement via reinforcement learning. In *AISec 2026 - 19th ACM Workshop on Artificial Intelligence and Security*, The Hague, Netherlands, November 2026. HAL: hal-05749810.

[30] Alexander D Kent. Cyber security data sources for dynamic network research. In *Dynamic networks and cyber-security*. 2016.

[31] Iqbal H Sarker, ASM Kayes, Shahriar Badsha, Hamed Alqahtani, Paul Watters, and Alex Ng. Cybersecurity data science: an overview from machine learning perspective. *Journal of Big data*, 2020.

[32] Rodney Lay and Gabor Strasser. The mitre corporation. *Risk Assessment and Management*, 2013.

[33] Harold Booth, Doug Rike, and Gregory A Witte. The national vulnerability database (nvd): Overview. 2013.

[34] Dietmar PF Möller. Nist cybersecurity framework and mitre cybersecurity criteria. In *Guide to cybersecurity in digital transformation: Trends, methods, technologies, applications and best practices*. 2023.

[35] Blake E Strom, Andy Applebaum, Doug P Miller, Kathryn C Nickels, Adam G Pennington, and Cody B Thomas. Mitre att&ck: Design and philosophy. In *Technical report*. The MITRE Corporation, 2018.

[36] Thomas D. Wagner, Khaled Mahbub, Esther Palomar, and Ali E. Abdallah. Cyber threat intelligence sharing: Survey and research directions. *Computers & Security*, 2019.

[37] Hakan Kekül, Burhan Ergen, and Halil Arslan. Comparison and analysis of software vulnerability databases. *International Journal of Engineering and Manufacturing*, 2022.

[38] Mark A Williams, Sumi Dey, Roberto Camacho Barranco, Sheikh Motahar Naim, M Shahriar Hossain, and Monika Akbar. Analyzing evolving trends of vulnerabilities in national vulnerability database. In *2018 IEEE International Conference on Big Data (Big Data)*, 2018.

[39] Su Zhang, Xinming Ou, and Doina Caragea. Predicting cyber risks through national vulnerability database. *Information Security Journal: A Global Perspective*, 2015.

[40] Temasek Laboratories, Sufatrio, Roland H. C. Yap, and Liming Zhong. A machine-oriented vulnerability database for automated vulnerability detection and processing. In *Proceedings of the 18th USENIX Conference on System Administration*, 2004.

[41] Karen Scarfone and Peter Mell. An analysis of cvss version 2 vulnerability scoring. In *2009 3rd International Symposium on Empirical Software Engineering and Measurement*, 2009.

[42] MITRE Corporation. Common Weakness Enumeration (CWE), 2006.

[43] Andrew Buttner and Neal Ziring. Common platform enumeration (cpe) specification. 2009.

[44] David Waltermire and Karen Scarfone. Guide to using vulnerability naming schemes. Technical report, National Institute of Standards and Technology (NIST), 2011.

[45] David W Baker, Steven M Christey, William H Hill, and David E Mann. The development of a common enumeration of vulnerabilities and exposures. In *Recent advances in intrusion detection*, 1999.

[46] Jiahuei Lin, Bram Adams, and Ahmed E. Hassan. On the coordination of vulnerability fixes. *Empirical Software Engineering*, 2023.

[47] Offensive Security. Exploit Database (ExploitDB), 2009.

[48] Allen D Householder, Garret Wassermann, Art Manion, and Chris King. The cert guide to coordinated vulnerability disclosure. Technical report, 2017.

[49] Thea Riebe, Marc-André Kaufhold, and Christian Reuter. The impact of organizational structure and technology use on collaborative practices in computer emergency response teams: An empirical study. *Proceedings of the ACM on human-computer interaction*, 2021.

[50] Ratsameetip Wita, Nattanatch Jiamnapanon, and Yunyong Teng-amnuay. An ontology for vulnerability lifecycle. In *2010 Third International Symposium on Intelligent Information Technology and Security Informatics*, 2010.

[51] Shuhan Liu, Jiayuan Zhou, Xing Hu, Filipe Roseiro Cogo, Xin Xia, and Xiaohu Yang. An empirical study on vulnerability disclosure management of open source software systems. *ACM Trans. Softw. Eng. Methodol.*, August 2025.

[52] Giovane C. M. Moura and John Heidemann. Vulnerability disclosure considered stressful. *SIGCOMM Comput. Commun. Rev.*, page 2–10, July 2023.

[53] Shengyi Pan, Jiayuan Zhou, Filipe Roseiro Cogo, Xin Xia, Lingfeng Bao, Xing Hu, Shanping Li, and Ahmed E. Hassan. Automated unearthing of dangerous issue reports. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2022.

[54] Jukka Ruohonen. A look at the time delays in cvss vulnerability scoring. *Applied Computing and Informatics*, 2019.

[55] Artur Balsam, Maciej Nowak, Michal Walkowski, Jacek Oko, and Sławomir Sujecki. Comprehensive comparison between versions cvss v2.0, cvss v3.x and cvss v4.0 as vulnerability severity measures. In *2024 24th International Conference on Transparent Optical Networks (ICTON)*, 2024.

[56] Jonathan Spring, Eric Hatleback, Allen Householder, Art Manion, and Deana Shick. Time to change the cvss? *IEEE Security & Privacy*, 2021.

[57] Assad Ali, Pavol Zavarsky, Dale Lindskog, and Ron Ruhl. A software application to analyze the effects of temporal and environmental metrics on overall cvss v2 score. In *2011 World Congress on Internet Security (WorldCIS-2011)*, 2011.

[58] Laurent Gallon. On the impact of environmental metrics on cvss scores. In *2010 IEEE second international conference on social computing*, 2010.

[59] Jean C Digitale, Jeffrey N Martin, and Medellena Maria Glymour. Tutorial on directed acyclic graphs. *Journal of clinical epidemiology*, 2022.

[60] Yan Wu, Irena Bojanova, and Yaacov Yesha. They know your weaknesses–do you?: Reintroducing common weakness enumeration. *CrossTalk*, 2015.

[61] MITRE Corporation. Common Attack Pattern Enumeration and Classification (CAPEC), 2007.

[62] John Matherly. Complete guide to shodan. *Shodan, LLC*, 2015.

[63] Zakir Durumeric, Hudson Clark, Jeff Cody, Elliot Cubit, Matt Ellison, Liz Izhikevich, and Ariana Mirian. Censys: A map of internet hosts and services. In *Proceedings of the ACM SIGCOMM 2025 Conference*, 2025.

[64] Christopher Bennett, A Abdou, and Paul C van Oorschot. Empirical scanning analysis of censys and shodan. In *Workshop on Measurements, Attacks, and Defenses for the Web*, 2021.

[65] Joana Cabral Costa, Tiago Roxo, João B. F. Sequeiros, Hugo Proenca, and Pedro R. M. INÁCIO. Predicting cvss metric via description interpretation. *IEEE Access*, 2022.

[66] Francesco Marchiori, Denis Donadel, and Mauro Conti. Can llms classify cves? investigating llms capabilities in computing cvss vectors. In *2025 IEEE Symposium on Computers and Communications (ISCC)*, 2025.

[67] Mustafizur R. Shahid and Hervé Debar. Cvss-bert: Explainable natural language processing to determine the severity of a computer security vulnerability from its description. In *2021 20th IEEE International Conference on Machine Learning and Applications (ICMLA)*, 2021.

[68] Philipp Kühn, David N. Relke, and Christian Reuter. Common vulnerability scoring system prediction based on open source intelligence information sources. *Computers & Security*, 2023.

[69] Yasuhiro Yamamoto, Daisuke Miyamoto, and Masaya Nakayama. Text-mining approach for estimating vulnerability score. In *2015 4th International Workshop on Building Analysis Datasets and Gathering Experience Returns for Security (BADGERS)*, 2015.

[70] Sho Isogai, Shinpei Ogata, Yutaro Kashiwa, Satoshi Yazawa, Kozo Okano, Takao Okubo, and Hironori Washizaki. Comparison of methods for automatically predicting cvss base vector. In *2024 IEEE 48th Annual Computers, Software, and Applications Conference (COMPSAC)*, 2024.

[71] Clément Elbaz, Louis Rilling, and Christine Morin. Fighting n-day vulnerabilities with automated cvss vector prediction at disclosure. In *Proceedings of the 15th International Conference on Availability, Reliability and Security*, ARES '20, 2020.

[72] Zhuobing Han, Xiaohong Li, Zhenchang Xing, Hongtao Liu, and Zhiyong Feng. Learning to predict severity of software vulnerability using only vulnerability description. In *2017 IEEE International Conference on Software Maintenance and Evolution*, 2017.

[73] Dinesh T Vasireddy, Dakota S Dale, and Qinghua Li. Cvss base score prediction using an optimized machine learning scheme. In *2023 Resilience Week (RWS)*, 2023.

[74] Maciej Nowak, Michał Walkowski, and Sławomir Sujecki. Conversion of cvss base score from 2.0 to 3.1. In *2021 International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*, pages 1–3, 2021.

[75] Maciej Nowak, Michał Walkowski, and Sławomir Sujecki. Machine learning algorithms for conversion of cvss base score from 2.0 to 3.x. In *Computational Science – ICCS 2021*, 2021.

[76] Michel Edkrantz and Alan Said. Predicting cyber vulnerability exploits with machine learning. In *SCAI*, 2015.

[77] Mehran Bozorgi, Lawrence K. Saul, Stefan Savage, and Geoffrey M. Voelker. Beyond heuristics: learning to classify vulnerabilities and predict exploits. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '10, 2010.

[78] Yong Fang, Yongcheng Liu, Cheng Huang, and Liang Liu. Fastembed: Predicting vulnerability exploitation possibility based on ensemble machine learning algorithm. *PLOS ONE*, 2020.

[79] Siddhartha Shankar Das, Edoardo Serra, Mahantesh Halappanavar, Alex Pothen, and Ehab Al-Shaer. V2w-bert: A framework for effective hierarchical multiclass classification of software vulnerabilities. In *2021 IEEE 8th International Conference on Data Science and Advanced Analytics*, 2021.

[80] Masaki Aota, Hideaki Kanehara, Masaki Kubo, Noboru Murata, Bo Sun, and Takeshi Takahashi. Automation of vulnerability classification from its description using machine learning. In *2020 IEEE Symposium on Computers and Communications (ISCC)*, 2020.

[81] Ashraf Haddad, Najwa Aaraj, Preslav Nakov, and Septimiu Fabian Mare. Automated mapping of cve vulnerability records to mitre cwe weaknesses, 2023.

[82] Ehsan Aghaei, Waseem Shadid, and Ehab Al-Shaer. Threatzoom: Hierarchical neural network for cves to cwes classification. In *Security and Privacy in Communication Networks*, 2020.

[83] Georgios Aivatoglou, Mike Anastasiadis, Georgios Spanos, Antonis Voulgaridis, Konstantinos Votis, and Dimitrios Tzovaras. A tree-based machine learning methodology to automatically classify software vulnerabilities. In *2021 IEEE International Conference on Cyber Security and Resilience (CSR)*, 2021.

[84] Xin Liu, Yuan Tan, Zhenghang Xiao, Jianwei Zhuge, and Rui Zhou. Not the end of story: An evaluation of ChatGPT-driven vulnerability description mappings. In *Findings of the Association for Computational Linguistics: ACL 2023*, 2023.

[85] Stefano Simonetto, Ronan Oostveen, Thijs van Ede, Peter Bosch, and Willem Jonker. Knowing your weaknesses is your greatest strength: Mapping cve to cwe by leveraging cwe hierarchy and fine-tuned llms. In *Proceedings of the ACM Asia Conference on Computer and Communications Security*, ASIA CCS '26, 2026.

[86] Nikita Mosievskiy. Fine-tuning roberta for cve-to-cwe classification: A 125m parameter model competitive with llms, 2026.

[87] Andrea Ciavotta, Alessandro Palma, Simone Lenti, and Silvia Bonomi. Threatlinker: An nlp-based methodology to automatically estimate cve relevance for capec attack patterns, 2026. To appear in the 21st European Dependable Computing Conference (EDCC 2026).

[88] Siddhartha Shankar Das, Ashutosh Dutta, Sumit Purohit, Edoardo Serra, Mahantesh Halappanavar, and Alex Pothen. Towards automatic mapping of vulnerabilities to attack patterns using large language models. In *2022 IEEE International Symposium on Technologies for Homeland Security (HST)*, 2022.

[89] Saad El Jaouhari, Nouredine Tamani, and Rohan Isaac Jacob. Guardlink: Dynamic linking of cve to mitre att&ck techniques using machine learning. In *GLOBECOM 2024 - 2024 IEEE Global Communications Conference*, 2024.

[90] Aditya Kuppa, Lamine Aouad, and Nhien-An Le-Khac. Linking cve's to mitre att&ck techniques. In *Proceedings of the 16th International Conference on Availability, Reliability and Security*, ARES '21, 2021.

[91] Benjamin Ampel, Sagar Samtani, Steven Ullman, and Hsinchun Chen. Linking common vulnerabilities and exposures to the mitre att&ck framework: A self-distillation approach. In *Proceedings of 2021 ACM Conference Knowledge Discovery and Data Mining (KDD' 21) Workshop on AI-enabled Cybersecurity Analytics*, 2021.

[92] Basel Abdeen, Ehab Al-Shaer, Anoop Singhal, Latifur Khan, and Kevin Hamlen. Smet: Semantic mapping of cve to att&ck and its application to cybersecurity. In *Data and Applications Security and Privacy XXXVII: 37th Annual IFIP WG 11.3 Conference, DBSec 2023*, 2023.

[93] Zefang Liu, Jialei Shi, and John F Buford. Cyberbench: A multi-task benchmark for evaluating large language models in cybersecurity. AAAI-24 Workshop on Artificial Intelligence for Cyber Security (AICS), 2024.

[94] Sevval Simsek, Varsha Athreya, and David Starobinski. Fixv2w: Correcting invalid cve-cwe mappings with knowledge graph embeddings, 2026.

[95] Şevval Şimşek, Howell Xia, Jonah Gluck, David Sastre Medina, and David Starobinski. Fixing invalid cve-cwe mappings in threat databases. In *2025 IEEE 49th Annual Computers, Software, and Applications Conference (COMPSAC)*, 2025.

[96] Jiamou Sun, Zhenchang Xing, Hao Guo, Deheng Ye, Xiaohong Li, Xiwei Xu, and Liming Zhu. Generating informative cve description from exploitdb posts by extractive summarization, 2021.

[97] Wei You, Peiyuan Zong, Kai Chen, XiaoFeng Wang, Xiaojing Liao, Pan Bian, and Bin Liang. Semfuzz: Semantics-based automatic generation of proof-of-concept exploits. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, 2017.

[98] Emanuele Iannone, Dario Di Nucci, Antonino Sabetta, and Andrea De Lucia. Toward automated exploit generation for known vulnerabilities in open-source libraries. In *2021 IEEE/ACM 29th International Conference on Program Comprehension (ICPC)*, 2021.

[99] Saad Ullah, Praneeth Balasubramanian, Wenbo Guo, Amanda Burnett, Hammond Pearce, Christopher Kruegel, Giovanni Vigna, and Gianluca Stringhini. From cve entries to verifiable exploits: An automated multi-agent framework for reproducing cves, 2026.

[100] Daan Hommersom, Antonino Sabetta, Bonaventura Coppola, Dario Di Nucci, and Damian A. Tamburri. Automated mapping of vulnerability advisories onto their fix commits in open source repositories. *ACM Trans. Softw. Eng. Methodol.*, 2024.

[101] Henning Perl, Sergej Dechand, Matthew Smith, Daniel Arp, Fabian Yamaguchi, Konrad Rieck, Sascha Fahl, and Yasemin Acar. Vccfinder: Finding potential vulnerabilities in open-source projects to assist code audits. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, 2015.

[102] Khyati Panchal, Siddhartha Shankar Das, Luis De La Torre, John Miller, Robert Rallo, and Mahantesh Halappanavar. Efficient clustering of software vulnerabilities using self organizing map (som). In *2022 IEEE International Symposium on Technologies for Homeland Security (HST)*, 2022.

[103] Fabian M Teichmann and Sonia R Boticiu. An overview of the benefits, challenges, and legal aspects of penetration testing and red teaming. *International Cybersecurity Law Review*, 2023.

[104] Matt Bishop. About penetration testing. *IEEE Security & Privacy*, 2007.

[105] Aileen G Bacudio, Xiaohong Yuan, Bei-Tseng Bill Chu, and Monique Jones. An overview of penetration testing. *International Journal of Network Security & Its Applications*, 2011.

[106] Matthew Denis, Carlos Zena, and Thaier Hayajneh. Penetration testing: Concepts, attack methods, and defense strategies. In *2016 IEEE Long Island Systems, Applications and Technology Conference (LISAT)*, 2016.

[107] Ivan Arce and Gary McGraw. Why attacking systems is a good idea. *IEEE Security & Privacy Magazine*, 2004.

[108] Ankur Chowdhary, Dijiang Huang, Jayasurya Sevalur Mahendran, Daniel Romo, Yuli Deng, and Abdulhakim Sabur. Autonomous security analysis and penetration testing. In *2020 16th International Conference on Mobility, Sensing and Networking (MSN)*, 2020.

[109] Xiangmin Shen, Lingzhi Wang, Zhenyuan Li, Yan Chen, Wencheng Zhao, Dawei Sun, Jiashui Wang, and Wei Ruan. Pentestagent: Incorporating llm agents to automated penetration testing. In *Proceedings of the 20th ACM Asia Conference on Computer and Communications Security*, ASIA CCS '25, 2025.

[110] Zhenguo Hu, Razvan Beuran, and Yasuo Tan. Automated penetration testing using deep reinforcement learning. In *2020 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, 2020.

[111] Yaroslav Stefinko, Andrian Piskozub, and Roman Banakh. Manual and automated penetration testing. benefits and drawbacks. modern tendency. In *2016 13th International Conference on Modern Problems of Radio Engineering, Telecommunications and Computer Science (TCSET)*, 2016.

[112] Gordon Lyon. Nmap: Network mapper, 2024.

[113] Tenable, Inc. Nessus vulnerability scanner, 2024.

[114] Greenbone Networks. Openvas – open vulnerability assessment system, 2024.

[115] PortSwigger Ltd. Burp suite: Web vulnerability scanner and testing platform, 2024.

[116] Rapid7. Metasploit framework, 2024.

[117] Alyzia-Maria Konsta, Alberto Lluch Lafuente, Beatrice Spiga, and Nicola Dragoni. Survey: Automatic generation of attack trees and attack graphs. *Computers & Security*, 2024.

[118] Carlos Sarraute, Olivier Buffet, and Jörg Hoffmann. Pomdps make better hackers: accounting for uncertainty in penetration testing. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence*, AAAI'12, 2012.

[119] Steven Noel, Sushil Jajodia, Brian O'Berry, and Michael Jacobs. Efficient minimum-cost network hardening via exploit dependency graphs. In *19th Annual Computer Security Applications Conference, 2003. Proceedings.*, 2003.

[120] Xinming Ou, Wayne F. Boyer, and Miles A. McQueen. A scalable approach to attack graph generation. In *Proceedings of the 13th ACM Conference on Computer and Communications Security*, CCS '06, 2006.

[121] David Tayouri, Nick Baum, Asaf Shabtai, and Rami Puzis. A survey of mulval extensions and their attack scenarios coverage. *IEEE Access*, 2023.

[122] Antoine Boudermine, Rida Khatoun, and Jean-Henri Choyer. Dynamic logic-based attack graph for risk assessment in complex computer systems. *Computer Networks*, 2023.

[123] Heqing Huang, Su Zhang, Xinming Ou, Atul Prakash, and Karem Sakallah. Distilling critical attack graph surface iteratively through minimum-cost sat solving. In *Proceedings of the 27th Annual Computer Security Applications Conference*, 2011.

[124] Jorge Lucangeli, Carlos Sarraute, and Gerardo Richarte. Attack planning in the real world. In *Workshop on Intelligent Security (SecArt), AAAI 2010*, 2010.

[125] Mehdi Yousefi, Nhamo Mtetwa, Yan Zhang, and Huaglory Tianfield. A reinforcement learning approach for attack graph analysis. In *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/ 12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*, 2018.

[126] Alessandro Palma and Marco Angelini. It is time to steer: A scalable framework for analysis-driven attack graph generation. In *Computer Security – ESORICS 2024*, 2024.

[127] Rohit Gangupantulu, Tyler Cody, Paul Park, Abdul Rahman, Logan Eisenbeiser, Dan Radke, and Ryan Clark. Using cyber terrain in reinforcement learning for penetration testing, 2022.

[128] Rohit Gangupantulu, Tyler Cody, Abdul Rahma, Christopher Redino, Ryan Clark, and Paul Park. Crown jewels analysis using reinforcement learning with attack graphs. In *2021 IEEE Symposium Series on Computational Intelligence (SSCI)*, 2021.

[129] Christos Papadimitriou and Kenneth Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. 1982.

[130] Kaiwen Li, Tao Zhang, Rui Wang, Yuheng Wang, Yi Han, and Ling Wang. Deep reinforcement learning for combinatorial optimization: Covering salesman problems. *IEEE Transactions on Cybernetics*, 2022.

[131] Zhuwen Li, Qifeng Chen, and Vladlen Koltun. Combinatorial optimization with graph convolutional networks and guided tree search. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NeurIPS'18, page 537–546, Red Hook, NY, USA, 2018. Curran Associates Inc.

[132] Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. Machine learning for combinatorial optimization: A methodological tour d'horizon. *European Journal of Operational Research*, 2021.

[133] Martin L. Puterman. Markov decision processes: Discrete stochastic dynamic programming. 1994.

[134] Runzhong Wang, Zhigang Hua, Gan Liu, Jiayi Zhang, Junchi Yan, Feng Qi, Shuang Yang, Jun Zhou, and Xiaokang Yang. A bi-level framework for learning to solve combinatorial optimization on graphs. In *Proceedings of the 35th International Conference on Neural Information Processing Systems*, NeurIPS '21, 2021.

[135] Quentin Cappart, Didier Chételat, Elias B. Khalil, Andrea Lodi, Christopher Morris, and Petar Veličković. Combinatorial optimization and reasoning with graph neural networks. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, 2021.

[136] Sebastian Sanokowski, Wilhelm Berghammer, Sepp Hochreiter, and Sebastian Lehner. Variational annealing on graphs for combinatorial optimization. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NeurIPS '23, 2023.

[137] Zhiqing Sun and Yiming Yang. Difusco: Graph-based diffusion solvers for combinatorial optimization. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, 2023.

[138] Andoni I. Garmendia, Josu Ceberio, and Alexander Mendiburu. Neural improvement heuristics for graph combinatorial optimization problems. *IEEE Transactions on Neural Networks and Learning Systems*, 2024.

[139] Haoyu Wang, Nan Wu, Hang Yang, Cong Hao, and Pan Li. Unsupervised learning for combinatorial optimization with principled objective relaxation. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NeurIPS '22, 2022.

[140] Nikolaos Karalias and Andreas Loukas. Erdos goes neural: an unsupervised learning framework for combinatorial optimization on graphs. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NeurIPS '20, 2020.

[141] Nasimeh Heydaribeni, Xinrui Zhan, Ruisi Zhang, Tina Eliassi-Rad, and Farinaz Koushanfar. Distributed constrained combinatorial optimization leveraging hypergraph neural networks. *Nature Machine Intelligence*, 2024.

[142] Martin J. A. Schuetz, J. Kyle Brubaker, and Helmut G. Katzgraber. Combinatorial optimization with physics-inspired graph neural networks. *Nature Machine Intelligence*, 2022.

[143] Ahmed Hussein, Mohamed Medhat Gaber, Eyad Elyan, and Chrisina Jayne. Imitation learning: A survey of learning methods. *ACM Comput. Surv.*, 2017.

[144] Thomas D. Barrett, William R. Clements, Jakob N. Foerster, and Alex I. Lvovsky. Exploratory combinatorial optimization with reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2020.

[145] Orly Stan, Ron Bitton, Michal Ezrets, Moran Dadon, Masaki Inokuchi, Yoshinobu Ohta, Tomohiko Yagyu, Yuval Elovici, and Asaf Shabtai. Heuristic approach for countermeasure selection using attack graphs. In *2021 IEEE 34th Computer Security Foundations Symposium (CSF)*, 2021.

[146] Huan Wang, Zhanfang Chen, Jianping Zhao, Xiaoqiang Di, and Dan Liu. A vulnerability assessment method in industrial internet of things based on attack graph and maximum flow. *IEEE Access*, 2018.

[147] Zang Yichao, Zhou Tianyang, Ge Xiaoyue, and Wang Qingxian. An improved attack path discovery algorithm through compact graph planning. *IEEE Access*, 2019.

[148] Carlos Sarraute, Gerardo Richarte, and Jorge Lucángeli Obes. An algorithm to find optimal attack paths in nondeterministic scenarios. In *Proceedings of the 4th ACM Workshop on Security and Artificial Intelligence*, AISec '11, 2011.

[149] Reginald E. Sawilla and Xinming Ou. Identifying critical attack assets in dependency attack graphs. In *Computer Security - ESORICS 2008*, 2008.

[150] Liang Lu, Rei Safavi-Naini, Markus Hagenbuchner, Willy Susilo, Jeffrey Horton, Sweah Liang Yong, and Ah Chung Tsoi. Ranking attack graphs with graph neural networks. In *Information Security Practice and Experience*, 2009.

[151] Rajendra Shivaji Patil, Ashish Kumar Dwivedi, and Vinay Sachidananda. Targetnoderank — target-specific node ranking in attack graphs. In *Machine Learning, Image Processing, Network Security and Data Sciences*, 2026.

[152] Alexandre Cottineau. *Using Machine Learning to Visualize and Analyze Attack Graphs*. Dissertation, KTH Royal Institute of Technology, 2021. Retrieved from KTH DiVA portal.

[153] Mehdi Yousefi, Nhamo Mtetwa, Yan Zhang, and Huaglory Tianfield. A novel approach for analysis of attack graph. In *2017 IEEE International Conference on Intelligence and Security Informatics (ISI)*, 2017.

[154] Chao Liu, Buhong Wang, Fan Li, Jiwei Tian, Yong Yang, Peng Luo, and Zhouzhou Liu. Optimal attack path planning based on reinforcement learning and cyber threat knowledge graph combining the attck for air traffic management system. *IEEE Transactions on Transportation Electrification*, 2024.

[155] Mariam Ibrahim, Ruba Elhafiz, and Abdallah Al-Wadi. Reinforcement learning-based attack graph analysis for wastewater treatment plant. *IEEE Transactions on Industry Applications*, 2023.

[156] Mohamed C. Ghanem, Thomas M. Chen, and Erivelton G. Nepomuceno. Hierarchical reinforcement learning for efficient and effective automated penetration testing of large networks. *Journal of Intelligent Information Systems*, 2023.

[157] Mohamed Nadhir Abid, Mounir Beggas, and Abdelkader Laouid. Reinforcement learning approach for iot security using cyberbattlesim: A simulation-based study. In *2024 6th International Conference on Pattern Analysis and Intelligent Systems (PAIS)*, 2024.

[158] Min Geun Song, Jaewoong Choi, and Huy Kang Kim. Gat-apg: Graph attention network-based attack path generation for security simulation. *IEEE Access*, 2025.

[159] Rajendra Shivaji Patil, Markus Dypbukt Källman, and Viktoria Fodor. Defencerank - ranking based attack graph analysis and defence prioritization. In *2024 IEEE International Conference on Cyber Security and Resilience (CSR)*, 2024.

[160] Juliano Vacaro, Guilherme Marques, Bruna Oliveira, Gabriel Paz, Thomas Paula, Wagston Staehler, and David Murphy. Sim-to-real in reinforcement learning for everyone. In *2019 Latin American Robotics Symposium (LARS), 2019 Brazilian Symposium on Robotics (SBR) and 2019 Workshop on Robotics in Education (WRE)*, 2019.

[161] R.S. Sutton and A.G. Barto. Reinforcement learning: An introduction. *IEEE Transactions on Neural Networks*, 1998.

[162] Fabio Muratore, Michael Gienger, and Jan Peters. Assessing transferability from simulation to reality for reinforcement learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

[163] Yunfei Wang, Shixuan Liu, Wenhao Wang, Changling Zhou, Chao Zhang, Jiandong Jin, and Cheng Zhu. A unified modeling framework for automated penetration testing, 2025.

[164] Callum Baillie, Maxwell Standen, Jonathon Schwartz, Michael Docking, David Bowman, and Junae Kim. Cyborg: An autonomous cyber operations research gym, 2020.

[165] Sean Oesch, Amul Chaulagain, Brian Weber, Matthew Dixson, Amir Sadovnik, Benjamin Roberson, Cory Watson, and Phillipe Austria. Towards a high fidelity training environment for autonomous cyber defense agents. In *Proceedings of the 17th Cyber Security Experimentation and Test Workshop*, CSET '24, 2024.

[166] Alexandre Légère, Li Li, François Rivest, and Ranwa Al Mallah. Training environments for reinforcement learning cybersecurity agents. In *2024 International Conference on Computing, Internet of Things and Microwave Systems (ICCIMS)*, 2024.

[167] Microsoft Defender Research Team. Cyberbattlesim, 2021. Created by Seifert C., Betser M., Blum W., Bono J., Farris K., Goren E., Grana J., Holsheimer K., Marken B., Neil J., Nichols N., Parikh J., Wei H.

[168] Li Li, Raed Fayad, and Adrian Taylor. Cygil: A cyber gym for training autonomous agents over emulated network systems. 2021.

[169] The MITRE Corporation. Brawl, 2018.

[170] Jonathon Schwartz and Hanna Kurniawatti. Nasim: Network attack simulator, 2019.

[171] Mitchell Kiely, Metin Ahiskali, Etienne Borde, Benjamin Bowman, David Bowman, Dirk van Bruggen, KC Cowan, Prithviraj Dasgupta, Erich Devendorf, Ben Edwards, et al. Exploring the efficacy of multi-agent reinforcement learning for autonomous cyber defence: A cage challenge 4 perspective. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2025.

[172] Jaromír Janisch, Tomáš Pevný, and Viliam Lisý. Nasimemu: Network attack simulator & emulator for training agents generalizing to novel scenarios, 2023.

[173] Marc-Antoine Faillon, Julien Francq, Nora Boulahia-Cuppens, Frédéric Cuppens, and Reda Yaich. Crlf: A sim2real reinforcement learning environment for automated it/ot pentesting. In *Computer Security. ESORICS 2025 International Workshops: DPM 2025, CBT 2025, CyberICPS 2025, Part I*, 2025.

[174] Andres Molina-Markham, Cory Miniter, Becky Powell, and Ahmad Ridley. Network environment design for autonomous cyberdefense, 2021.

[175] Kim Hammar. Csle: A reinforcement learning platform for autonomous security management, 2026.

[176] Michael Lanier and Yevgeniy Vorobeychik. Cygym: A simulation-based game-theoretic analysis framework fornbsp;cybersecurity. In *Game Theory and AI for Security: 16th International Conference, GameSec 2025, Athens, Greece, October 13–15, 2025, Proceedings, Part I*, 2025.

[177] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017.

[178] Marian Gawron, Feng Cheng, and Christoph Meinel. Automatic vulnerability classification using machine learning. In *Risks and Security of Internet and Systems*, 2018.

[179] Thrasyvoulos Giannakopoulos and Konstantinos Maliatsos. On the usage of nlp on cve descriptions for calculating risk. In *Computer Security. ESORICS 2023 International Workshops*, 2024.

[180] Rohit Babbar, Ioannis Partalas, Eric Gaussier, and Massih R. Amini. On flat versus hierarchical classification in large-scale taxonomies. In *Proceedings of the 27th International Conference on Neural Information Processing Systems*, 2013.

[181] Grigorios Tsoumakas and Ioannis Katakis. Multi-label classification: An overview. *International Journal of Data Warehousing and Mining (IJDWM)*, 2007.

[182] Yue Zhu, James T. Kwok, and Zhi-Hua Zhou. Multi-label learning with global and local label correlation. *IEEE Transactions on Knowledge and Data Engineering*, 2018.

[183] Mengqi Yuan, Jinke Xu, and Zhongnian Li. Long tail multi-label learning. In *2019 IEEE Second International Conference on Artificial Intelligence and Knowledge Engineering (AIKE)*, 2019.

[184] Min-Ling Zhang and Zhi-Hua Zhou. A review on multi-label learning algorithms. *IEEE Transactions on Knowledge and Data Engineering*, 2014.

[185] Mohaimenul Azam Khan Raiaan, Md Saddam Hossain Mukta, Kaniz Fatema, Nur Mohammad Fahad, Sadman Sakib, Most Marufatul Jannat Mim, Jubaer Ahmad, Mohammed Eunus Ali, and Sami Azam. A review on large language models: Architectures, applications, taxonomies, open issues and challenges. *IEEE access*, 2024.

[186] Liang Wang, Nan Yang, Xiaolong Huang, Linjun Yang, Rangan Majumder, and Furu Wei. Improving text embeddings with large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics* (*Volume 1: Long Papers*), 2024.

[187] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 2017.

[188] Evans Kotei and Ramkumar Thirunavukarasu. A systematic review of transformer-based pre-trained language models through self-supervised learning. *Information*, 2023.

[189] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, 2020.

[190] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 2020.

[191] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2019.

[192] Marcus Ma, Georgios Chochlakis, Niyantha Maruthu Pandiyan, Jesse Thomason, and Shrikanth Narayanan. Large language models do multi-label classification differently. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, 2025.

[193] Stefano Simonetto and Peter Bosch. Comprehensive threat analysis and systematic mapping of CVEs to MITRE framework. In *Proceedings of the First International Conference on Natural Language Processing and Artificial Intelligence for Cyber Security* (*NLPAICS*), 2024.

[194] Galeax. CVE2CAPEC: Automated pipeline linking cves, cwe, capec, and mitre att&ck.

[195] Rafael B. Pereira, Alexandre Plastino, Bianca Zadrozny, and Luiz H.C. Merschmann. Correlation analysis of performance measures for multi-label classification. *Information Processing & Management*, 2018.

[196] Zhiqiang Wang, Yiran Pang, and Yanbin Lin. Large language models are zero-shot text classifiers, 2023.

[197] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. 2023.

[198] Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. 2024.

[199] Max Peeperkorn, Tom Kouwenhoven, Dan Brown, and Anna Jordanous. Is temperature the creativity parameter of large language models?, 2024.

[200] Konstantinos Sechidis, Grigorios Tsoumakas, and Ioannis Vlahavas. On the stratification of multi-label data. In *Machine Learning and Knowledge Discovery in Databases. ECML PKDD*, 2011.

[201] Kevin Clark, Minh-Thang Luong, Quoc V. Le, and Christopher D. Manning. Electra: Pre-training text encoders as discriminators rather than generators. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2020.

[202] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter, 2020.

[203] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach, 2019.

[204] Ehsan Aghaei, Xi Niu, Waseem Shadid, and Ehab Al-Shaer. Securebert: A domain-specific language model for cybersecurity. In *Security and Privacy in Communication Networks*, 2023.

[205] Kun. Secbert: A pretrained language model for cyber security text, 2024.

[206] Markus Bayer, Philipp Kuehn, Ramin Shanehsaz, and Christian Reuter. Cysecbert: A domain-adapted language model for the cybersecurity domain. *ACM Trans. Priv. Secur.*, 2024.

[207] David R. Roberts, Volker Bahn, Simone Ciuti, Mark S. Boyce, Jane Elith, Gurutzeta Guillera-Arroita, Severin Hauenstein, José J. Lahoz-Monfort, Boris Schröder, Wilfried Thuiller, David I. Warton, Brendan A. Wintle, Florian Hartig, and Carsten F. Dormann. Cross-validation strategies for data with temporal, spatial, hierarchical, or phylogenetic structure. *Ecography*, 2017.

[208] Cédric Bonhomme and Alexandre Dulaunoy. Mapping cves to mitre attck techniques: A curated gold-set classifier and the limits of llm-assisted label expansion, 2026.

[209] Victor-Alexandru Darvariu, Stephen Hailes, and Mirco Musolesi. Graph reinforcement learning for combinatorial optimization: A survey and unifying perspective. 2024.

[210] Richard M. Karp. *Reducibility among Combinatorial Problems*, pages 85–103. Springer US, Boston, MA, 1972.

[211] Shen Lin. Computer solutions of the traveling salesman problem. *Bell System Technical Journal*, 1965.

[212] Nina Mazyavkina, Sergey Sviridov, Sergei Ivanov, and Evgeny Burnaev. Reinforcement learning for combinatorial optimization: A survey. *Computers & Operations Research*, 2021.

[213] Quentin Cappart, Thierry Moisan, Louis-Martin Rousseau, Isabeau Prémont-Schwarz, and Andre A. Cire. Combining reinforcement learning and constraint programming for combinatorial optimization. *Proceedings of the AAAI Conference on Artificial Intelligence*, 2021.

[214] Yifan Peng, Byungki Choi, and Jun Xu. Graph learning for combinatorial optimization: A survey of state-of-the-art. *Data Science and Engineering*, 2021.

[215] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 2009.

[216] Zhi-Peng Li, Si-Guo Wang, Qin-Hu Zhang, Yi-Jie Pan, Nai-An Xiao, Jia-Yang Guo, Chang-An Yuan, Wen-Jian Liu, and De-Shuang Huang. Graph pooling for graph-level representation learning: a survey. *Artificial Intelligence Review*, 2024.

[217] Hado van Hasselt and Marco A. Wiering. Using continuous action spaces to solve discrete problems. In *Proceedings of the 2009 International Joint Conference on Neural Networks*, IJCNN'09, 2009.

[218] Yu Chen, Yingfeng Chen, Zhipeng Hu, Tianpei Yang, Changjie Fan, Yang Yu, and Jianye Hao. Learning action-transferable policy with action embedding. 2021.

[219] Gabriel Dulac-Arnold, Richard Evans, Hado van Hasselt, Peter Sunehag, Timothy Lillicrap, Jonathan Hunt, Timothy Mann, Theophane Weber, Thomas Degris, and Ben Coppin. Deep reinforcement learning in large discrete action spaces. 2016.

[220] Hanjun Dai, Elias B. Khalil, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NeurIPS'17, 2017.

[221] Yang Xiao, Jun Liu, Jiawei Wu, and Nirwan Ansari. Leveraging deep reinforcement learning for traffic engineering: A survey. *IEEE Communications Surveys & Tutorials*, 2021.

[222] Yixin Liu, Yu Zheng, Daokun Zhang, Hongxu Chen, Hao Peng, and Shirui Pan. Towards unsupervised deep graph structure learning. In *Proceedings of the ACM Web Conference 2022*, WWW '22, 2022.

[223] Mingshuo Nie, Dongming Chen, and Dongqi Wang. Reinforcement learning on graphs: A survey. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 2023.

[224] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 2021.

[225] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 2013.

[226] Guoqiang Zhong, Li-Na Wang, Xiao Ling, and Junyu Dong. An overview on data representation learning: From traditional feature learning to recent deep learning. *The Journal of Finance and Data Science*, 2016.

[227] Daokun Zhang, Jie Yin, Xingquan Zhu, and Chengqi Zhang. Network representation learning: A survey. *IEEE Transactions on Big Data*, 2020.

[228] Sascha Lange and Martin Riedmiller. Deep auto-encoder neural networks in reinforcement learning. In *The 2010 International Joint Conference on Neural Networks (IJCNN)*, 2010.

[229] Yasi Wang, Hongxun Yao, and Sicheng Zhao. Auto-encoder based dimensionality reduction. *Neurocomputing*, 2016.

[230] Zewen Li, Fan Liu, Wenjie Yang, Shouheng Peng, and Jun Zhou. A survey of convolutional neural networks: analysis, applications, and prospects. *IEEE transactions on neural networks and learning systems*, 2021.

[231] Larry R Medsker, Lakhmi Jain, et al. Recurrent neural networks. *Design and applications*, 2001.

[232] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Message passing neural networks. In *Machine learning meets quantum physics*. 2020.

[233] Daniele Grattarola, Daniele Zambon, Filippo Maria Bianchi, and Cesare Alippi. Understanding pooling in graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 2024.

[234] Si Zhang, Hanghang Tong, Jiejun Xu, and Ross Maciejewski. Graph convolutional networks: a comprehensive review. *Computational Social Networks*, 2019.

[235] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. 2017.

[236] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 2017.

[237] David Ha and Jürgen Schmidhuber. Recurrent world models facilitate policy evolution. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, 2018.

[238] Thomas N. Kipf and Max Welling. Variational graph auto-encoders, 2016.

[239] Sahil Manchanda, Akash Mittal, Anuj Dhawan, Sourav Medya, Sayan Ranu, and Ambuj Singh. Learning heuristics over large graphs via deep reinforcement learning. 2020.

[240] Irit Dinur and Samuel Safra. On the hardness of approximating minimum vertex cover. *Annals of mathematics*, 2005.

[241] Guillermo Bernárdez, José Suárez-Varela, Albert López, Xiang Shi, Shihan Xiao, Xiangle Cheng, Pere Barlet-Ros, and Albert Cabellos-Aparicio. Magnneto: A graph neural network-based multi-agent system for traffic engineering. *IEEE Transactions on Cognitive Communications and Networking*, 2023.

[242] Sunil Arya, David M. Mount, Nathan S. Netanyahu, Ruth Silverman, and Angela Y. Wu. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *J. ACM*, 1998.

[243] Hans L. Bodlaender and Klaus Jansen. On the complexity of the maximum cut problem. *Nordic J. of Computing*, 2000.

[244] John Healy and Leland McInnes. Uniform manifold approximation and projection. *Nature Reviews Methods Primers*, 2024.

[245] Saurabh Kumar, Aviral Kumar, Sergey Levine, and Chelsea Finn. One solution is not all you need: Few-shot extrapolation via structured maxent rl. In *Advances in Neural Information Processing Systems*, volume 33, 2020.

[246] Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron Courville, and Marc G. Bellemare. Deep reinforcement learning at the edge of the statistical precipice. In *Proceedings of the 35th International Conference on Neural Information Processing Systems*, NeurIPS '21, 2021.

[247] Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data*, 2019.

[248] Seungoh Choi, Jeong-Han Yun, and Sin-Kyu Kim. A comparison of ics datasets for security research based on attack paths. In Eric Luiijf, Inga Žutautaitė, and Bernhard M. Hämmerli, editors, *Critical Information Infrastructures Security*, 2019.

[249] Qianyu Li, Ruipeng Wang, Dong Li, Fan Shi, Min Zhang, Anupam Chattopadhyay, Yi Shen, and Yang Li. Dynpen: Automated penetration testing in dynamic network scenarios using deep reinforcement learning. *Trans. Info. For. Sec.*, 2024.

[250] Shicheng Zhou, Jingju Liu, Yuliang Lu, Jiahai Yang, Dongdong Hou, Yue Zhang, and Shulong Hu. April: Towards scalable and transferable autonomous penetration testing in large action space via action embedding. *IEEE Transactions on Dependable and Secure Computing*, 2025.

[251] Andy Applebaum, Camron Dennler, Patrick Dwyer, Marina Moskowitz, Harold Nguyen, Nicole Nichols, Nicole Park, Paul Rachwalski, Frank Rau, Adrian Webster, and Melody Wolk. Bridging automated to autonomous cyber defense: Foundational analysis of tabular q-learning. In *Proceedings of the 15th ACM Workshop on Artificial Intelligence and Security*, AISec'22, 2022.

[252] Xiaotong Guo, Jing Ren, Jiangong Zheng, Jianxin Liao, Chao Sun, Hongxi Zhu, Tongyu Song, Sheng Wang, and Wei Wang. Automated penetration testing with fine-grained control through deep reinforcement learning. *Journal of Communications and Information Networks*.

[253] Qian Yao, Yongjie Wang, Xinli Xiong, and Yang Li. Intelligent penetration testing in dynamic defense environment. In *Proceedings of the 2022 International Conference on Cyber Security*, 2023.

[254] Pan Jia, Chuanjun Yi, and Zhengkun Chen. Pbrs for sim-to-real cyber rl: Training agents in a high-fidelity cyberbattlesim. In *2025 6th International Symposium on Computer Engineering and Intelligent Communications (ISCEIC)*, 2025.

[255] Yixuan Cao, Aniket Mahanti, and Ranesh Naha. A real-time defense framework using pppo in deep reinforcement learning for cyberbattlesim. In *2025 IEEE 24th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, 2025.

[256] Thomas Kunz, Christian Fisher, James La Novara-Gsell, Christopher Nguyen, and Li Li. A multiagent cyberbattlesim for rl cyber operation agents. In *2022 International Conference on Computational Science and Computational Intelligence (CSCI)*. IEEE, 2022.

[257] Kim Bum-Sok, Hye-Won Suk, Choi Yong-Hoon, Moon Dae-Sung, and Kim Min-Suk. Optimal cyber attack strategy using reinforcement learning based on common vulnerability scoring system. *Computer Modeling in Engineering & Sciences*, 2024.

[258] Aritran Piplai, Mike Anoruo, Kayode Fasaye, Anupam Joshi, Tim Finin, and Ahmad Ridley. Knowledge guided two-player reinforcement learning for cyber attacks and defenses. In *2022 21st IEEE International Conference on Machine Learning and Applications (ICMLA)*, 2022.

[259] Yang Yang and Ran Pengcheng. Integrating cyber deception into multi-agent reinforcement learning for proactive network defense. In *2025 22nd International Computer Conference on Wavelet Active Media Technology and Information Processing (ICCWAMTIP)*, 2025.

[260] Abid Mohamed Nadhir, Beggas Mounir, Laouid Abdelkader, and Mohammad Hammoudeh. Enhancing cybersecurity in healthcare iot systems using reinforcement learning. *Transportation Research Procedia*, 2025.

[261] Qianyu Li, Li Wen, Anupam Chattopadhyay, Cheng Tu, Miao Hu, Fan Shi, Min Zhang, Ruipeng Wang, and Zulie Pan. Intelligent penetration testing through integrated knowledge graph and historical decision enhancement. *IEEE Transactions on Dependable and Secure Computing*, 2026.

[262] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust region policy optimization, 2017.

[263] Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods, 2018.

[264] Arsenii Kuznetsov, Pavel Shvechikov, Alexander Grishin, and Dmitry Vetrov. Controlling overestimation bias with truncated mixture of continuous distributional quantile critics, 2020.

[265] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor, 2018.

[266] Marco Pleines, Matthias Pallasch, Frank Zimmer, and Mike Preuss. Generalization, mayhems and limits in recurrent proximal policy optimization, 2022.

[267] Satinder Singh, Tommi Jaakkola, Michael L Littman, and Csaba Szepesvári. Convergence results for single-step on-policy reinforcement-learning algorithms. *Machine learning*, 2000.

[268] Scott Fujimoto, David Meger, and Doina Precup. Off-policy deep reinforcement learning without exploration. In *International conference on machine learning*. PMLR, 2019.

[269] Yuping Xing and Yongzhao Zhan. Virtualization and cloud computing. In *Future Wireless Networks and Information Systems: Volume 1*, 2012.

[270] Andreas Blenk, Arsany Basta, Martin Reisslein, and Wolfgang Kellerer. Survey on network virtualization hypervisors for software defined networking. *IEEE Communications Surveys & Tutorials*, 2015.

[271] Mohammad Masdari, Sayyid Shahab Nabavi, and Vafa Ahmadi. An overview of virtual machine placement schemes in cloud computing. *Journal of Network and Computer Applications*, 2016.

[272] Jianing Pei, Peilin Hong, Miao Pan, Jiangqing Liu, and Jingsong Zhou. Optimal vnf placement via deep reinforcement learning in sdn/nfv-enabled networks. *IEEE Journal on Selected Areas in Communications*, 2019.

[273] Xinlei Han, Raymond Schooley, Delvin Mackenzie, Olaf David, and Wes J. Lloyd. Characterizing public cloud resource contention to support virtual machine co-residency prediction. In *2020 IEEE International Conference on Cloud Engineering (IC2E)*, 2020.

[274] Akhilesh Raj, Swann Perarnau, and Aniruddha Gokhale. A reinforcement learning approach for performance-aware reduction in power consumption of data center compute nodes. In *2023 IEEE International Conference on Cloud Engineering (IC2E)*, 2023.

[275] Jin Han, Wanyu Zang, Songqing Chen, and Meng Yu. Reducing security risks of clouds through virtual machine placement. In *Data and Applications Security and Privacy XXXI: 31st Annual IFIP WG 11.3 Conference, DBSec 2017*, 2017.

[276] Peter Mell, Karen Scarfone, and Sasha Romanosky. Common vulnerability scoring system. *IEEE Security & Privacy*, 2006.

[277] Sattar Feizollahibarough and Mehrdad Ashtiani. A security-aware virtual machine placement in the cloud using hesitant fuzzy decision-making processes. *The Journal of Supercomputing*, 2021.

[278] Luca Caviglione, Mauro Gaggero, Massimo Paolucci, and Roberto Ronco. Deep reinforcement learning for multi-objective placement of virtual machines in cloud datacenters. *Soft Comput.*, 2021.

[279] Mauro Gaggero and Luca Caviglione. Model predictive control for energy-efficient, quality-aware, and secure virtual machine placement. *IEEE Transactions on Automation Science and Engineering*, 2019.

[280] William Voorsluys, James Broberg, Srikumar Venugopal, and Rajkumar Buyya. Cost of virtual machine live migration in clouds: A performance evaluation. In *Cloud Computing*, 2009.

[281] Michael R Garey and David S Johnson. Approximation algorithms for bin packing problems: A survey. In *Analysis and design of algorithms in combinatorial optimization*. 1981.

[282] J. C. Ferreira, C. M. Fonseca, and A. Gaspar-Cunha. Methodology to select solutions from the pareto-optimal set: a comparative study. In *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation*, GECCO '07, 2007.

[283] Xiuxia Yin, Zhiwei Gao, Dong Yue, and Songlin Hu. Cloud-based event-triggered predictive control for heterogeneous nmass under both dos attacks and transmission delays. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 2022.

[284] B. B. Gupta and Omkar P. Badve. Taxonomy of dos and ddos attacks and desirable defense mechanism in a cloud computing environment. *Neural Computing and Applications*, 2017.

[285] Olufogorehan Tunde-Onadele, Yuhang Lin, Xiaohui Gu, and Jingzhu He. Understanding software security vulnerabilities in cloud server systems. In *2022 IEEE International Conference on Cloud Engineering (IC2E)*, 2022.

[286] R Timothy Marler and Jasbir S Arora. The weighted sum method for multi-objective optimization: new insights. *Structural and multidisciplinary optimization*, 2010.

[287] Weichao Ding, Chunhua Gu, Fei Luo, Yaohui Chang, Ulysse Rugwiro, Xiaoke Li, and Geng Wen. Dfa-vmp: An efficient and secure virtual machine placement strategy under cloud environment. *Peer-to-Peer Networking and Applications*, 2018.

[288] Deepika Saxena, Ishu Gupta, Jitendra Kumar, Ashutosh Kumar Singh, and Xiaoqing Wen. A secure and multiobjective virtual machine placement framework for cloud data center. *IEEE Systems Journal*, 2022.

[289] Deepika Saxena, Smruti Rekha Swain, Jatinder Kumar, Sakshi Patni, Kishu Gupta, Ashutosh Kumar Singh, and Volker Lindenstruth. Secure resource management in cloud computing: Challenges, strategies and meta-analysis. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 2025.

[290] Homa Shirafkan and Alireza Shameli-Sendi. Adaptive virtual machine placement: a dynamic approach for energy-efficiency, qos enhancement, and security optimization. *Cluster Computing*, 2024.

[291] Xuebiao Yuchi and Sachin Shetty. Enabling security-aware virtual machine placement in iaas clouds. In *MILCOM 2015 - 2015 IEEE Military Communications Conference*, 2015.

[292] Zhao-Hui Du, Zhiwei Ying, Zhenke Ma, Yufei Mai, Phoebe Wang, Jesse Liu, and Jesse Fang. Secure encrypted virtualization is unsecure, 2017.

[293] Liyuan Wang, Xingxing Zhang, Hang Su, and Jun Zhu. A comprehensive survey of continual learning: Theory, method and application. *IEEE transactions on pattern analysis and machine intelligence*, 2024.

[294] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In *Proceedings of the 29th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'15, 2015.

[295] Antonio Loquercio, Mattia Segu, and Davide Scaramuzza. A general framework for uncertainty estimation in deep learning. *IEEE Robotics and Automation Letters*, 2020.

[296] Andrew G Barto and Sridhar Mahadevan. Recent advances in hierarchical reinforcement learning. *Discrete event dynamic systems*, 2003.

[297] Marc-Antoine Faillon, Baptiste Bout, Julien Francq, Christopher Neal, Nora Boulahia-Cuppens, Frédéric Cuppens, and Reda Yaich. How to better fit reinforcement learning for pentesting: A new hierarchical approach. In *Computer Security – ESORICS 2024*, 2024.

[298] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework, 2019.

[299] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *Advances in neural information processing systems*, 2011.

LIST OF FIGURES

1 Thesis organization into chapters with links to research questions and contributions. 8

1.1 NVD entry highlighting its main components, including CVSS metrics, vulnerability references such as exploit links and vendor advisories, CWEs, and known affected products. . . . 15
1.2 CVE counts per quarter and year from 2000 to Q1 2025. . . . 15
1.3 Overview of the stakeholders involved in coordinated vulnerability disclosure, their respective roles and interactions, and the temporal lifecycle of disclosure. [48] 16

2.1 Example of RL-driven attack path planning using a POMDP. . . . 41

3.1 Overview of the proposed POMDP design and interaction loop between agent and environment, including the roles of the observation function O, transition dynamics T, reward function R, and initial state distribution ρ_0. . . . 56
3.2 Example of a POMDP episode illustrating discovery, exploitation, and impact-driven decision making. . . . 57
3.3 Overview of the automated scenario generation pipeline, where assets and their associated vulnerabilities are collected from a CTI tool and vulnerability database, then stored in the environment database for scenario generation. . . . 59
3.4 Overview of the automated graph scenario construction pipeline: graph initialization and vulnerability allocation. . . . 60
3.5 Overview of the automated graph scenario construction pipeline: cyber-terrain and crown jewels allocation. . . . 61
3.6 Domain randomization strategy supported by proper distributions for each high-level parameter. . . . 62

4.1 Overview of CVE linkage and CWE assignment delay trends across years. . . . 70
4.2 Representation of hierarchical classification with ATT&CK as an example. . . . 74
4.3 Distribution of labels and label co-occurrence in the dataset across the taxonomies. 76
4.4 Inference time comparison among hierarchical, flat, and shared-embedding hierarchical methods. . . . 81
4.5 Training set coverage, unique label coverage, and performance trends under recursive temporal splitting using the flat SecureBERT classifier. . . . 82
4.6 Integration of the LM-based classifier as outcome approximator in the simulation environment. . . . 84

5.1 Joint GNN encoder-decoder framework leveraging valid action-sampled snapshots for node representation learning. . . . 95
5.2 Overview of action space construction from node embeddings, illustrating two application examples and possible representation operator choices. . . . 96
5.3 User-defined logic, attributes, and methods provided for GAE pre-training and RL integration. . . . 97
5.4 Overview of related work on action-space complexity and the operational capabilities of RL methods across several key features. . . . 99

5.5 UMAP projection of benchmark action spaces for a 20-node graph instance, illustrating local geometry and neighborhood proximity with the compactness metric. . . . 101
5.6 Evolution of action-selection time across graph sizes and benchmarks. . . . 104

6.1 Graph representation of the environment scenario, illustrating the design of node and edge feature vectors. . . . 113
6.2 The discovered graph is processed by a GNN to generate node embeddings, which are pooled into a fixed-dimensional graph embedding used as the agent's observation. . . . 115
6.3 The action space is defined as the set of all possible concatenations of source, target, vulnerability, and outcome embeddings. . . . 116
6.4 The normalized average AUC score for agents learning from different spaces, with the curve depicting score progress. . . . 119
6.5 GAE reconstruction training and validation loss. . . . 121
6.6 Rank probability matrix highlighting the likelihood of each DRL algorithm to achieve a certain rank across goals. . . . 122
6.7 TRPO agent performance across different attack goals and scenario sets. . . . 122
6.8 TRPO agent performance on the test set under varying defender intervention probabilities. . . . 123
6.9 TRPO agent, retrained or not, tested against heuristics on both emulated and real-world replica environments. . . . 125

7.1 Overview of the VMP attack surface, illustrating VM escape through hypervisor vulnerabilities and OS-specific configurations. . . . 129
7.2 The MDP for the defender agent trained to produce reconfiguration actions. . . . 134
7.3 The defender receives an episodic reward based on the weighted change in attack surface and performance between the original and new VMP. . . . 135
7.4 Approximation of the attack surface objective for the defender agent. . . . 136
7.5 The scenario generator selects a cloud provider and uses CTI tools to gather statistics for generating realistic simulation scenarios. . . . 137
7.6 Average variation of metrics across different attacker threat models. . . . 140
7.7 Average variation of metrics as a function of β, smoothed with a Savitzky-Golay filter. . . . 141
7.8 Average variation of metrics after reconfiguration across cloud scenarios. . . . 142
7.9 Average variation of metrics after reconfiguration across initial allocations. . . . 142
7.10 Average variation of metrics as a function of the number of VMs and PMs. . . . 143
7.11 Average variation of metrics and % of replicas deployed across initial allocation strategies. . . . 144

2 Overview of the relationships between the main contributions presented in this thesis and their dependencies. . . . 148

LIST OF TABLES

1.1 Overview of automated cybersecurity knowledge linking approaches grouped by source and target knowledge domains. . . . 24

2.1 Comparison of the main attack graph representations based on space complexity, characteristics, and limitations. . . . 30
2.2 Categorization of heuristic and learning-based approaches for attack graph analysis and critical component identification. . . . 42

3.1 Classification of existing cyber-range platforms by execution model and action-space granularity. . . . 50
3.2 Deterministic and stochastic factors enforced by the state transition function in the proposed POMDP. . . . 55

4.1 Comparison of existing automated mapping approaches across key evaluation and training criteria: MS (multiple splits), SS (stratified splits), MM (multi-label metrics), CI (class imbalance handling), and H (hyperparameter optimization/exploration). A dash ("–") denotes missing information. . . . 72
4.2 Zero-shot classification performance of text-generation LLMs across three cybersecurity taxonomy prediction tasks evaluated on the oldest 1,000 CVEs. Reported results represent the average performance across temperature settings of 0, 0.5, and 1.0. . . . 78
4.3 Comparison between ELECTRA and BERT models (and variants) using mean test scores obtained from 5-fold CV across the three classification tasks. . . . 80
4.4 Comparison between hierarchical and flat SecureBERT classifiers using mean test scores obtained from 5-fold CV across the three classification tasks. . . . 81

5.1 Action-space structure across benchmarks, including worst-case growth and the chosen representation function. . . . 101
5.2 IQM of the normalized generalization score across test benchmarks and training strategies. . . . 103
5.3 Scaling of action-selection time with fitted power-law exponent α. . . . 104

6.1 Comparison of benchmark scenario complexity and evaluation dimensions in related work. . . . 112
6.2 Relative increase or decrease of selected scenario characteristics in the validation and test sets with respect to the training set. . . . 121

7.1 Comparison of security-related and performance-related objectives in existing studies for multi-objective VMP. . . . 132
7.2 Scenario statistics per cloud provider, including hypervisor type and associated average counts. . . . 138

A.1 Search space for hyperparameters and loss functions (bold = best) . . . 163
A.2 Hyperparameter ranges explored for each method with best configurations in **bold**. 164
A.3 Search space for on-policy RL hyperparameters (bold = best) . . . 166

A.4 Search space for off-policy RL and GAE hyperparameters (bold = best) 166

C.1 Description of scenario features, including information about which range has been sampled during scenario generation, with which sampling strategy, and the resulting complexity categorization. 176

D.1 Scenario configuration features, including parameter descriptions, sampling ranges, selection strategies, and their implications for scenario complexity. 178

ACRONYMS

ANR ..... Agence Nationale de la Recherche

ATT&CK ..... Adversarial Tactics, Techniques, and Common Knowledge

AUC ..... Area Under the Curve

BCI ..... Bootstrapped Confidence Interval

BERT ..... Bidirectional Encoder Representations from Transformers

CAPEC ..... Common Attack Pattern Enumeration and Classification

CERT ..... Computer Emergency Response Team

CNA ..... CVE Numbering Authority

CO ..... Combinatorial Optimization

CPE ..... Common Platform Enumeration

CV ..... Cross Validation

CVE ..... Common Vulnerabilities and Exposures

CVSS ..... Common Vulnerability Scoring System

CWE ..... Common Weakness Enumeration

CTI ..... Cyber Threat Intelligence

DAG ..... Directed Acyclic Graph

DNN ..... Deep Neural Network

DRL ..... Deep Reinforcement Learning

ECMP ..... Equal-Cost Multi-Path

GAE ..... Graph Autoencoder

GCO ..... Graph Combinatorial Optimization

GPT ..... Generative Pre-trained Transformer

GNN ..... Graph Neural Network

ICS ..... Industrial Control System

IoT ..... Internet of Things

IQM ..... Interquartile Mean

LAG ..... Link Aggregation Group

LM ..... Language Model

LLM ..... Large Language Model

MARL ..... Multi-Agent Reinforcement Learning

MDP ..... Markov Decision Process

MH ..... Manual Heuristics

ML ..... Machine Learning

MinVertex ..... Minimum Vertex

MOOP ..... Multi-Objective Optimization Problem

MRR ..... Mean Reciprocal Rank

MulVAL ..... Multi-host, Multi-stage Vulnerability Analysis Language

NIST ..... National Institute of Standards and Technology

NN ..... Nearest Neighbor

NVD ..... National Vulnerability Database

OS ..... Operating System

OSPF ..... Open Shortest Path First

PDDL ..... Planning Domain Definition Language

PM ..... Physical Machine

PoC ..... Proof of Concept

POMDP ..... Partially Observable Markov Decision Process

PT ..... Penetration Testing

RL ..... Reinforcement Learning

RQ ..... Research Question

T5 ..... Text-To-Text Transfer Transformer

TPE ..... Tree-structured Parzen Estimator

Traffic ..... Traffic Engineering

TSP ..... Traveling Salesman Problem

UR ..... User Requirement

VM ..... Virtual Machine

VMP ..... Virtual Machine Placement

Abstract

As modern networked infrastructures continue to increase in scale, heterogeneity, and complexity, identifying vulnerabilities that are truly critical within a specific operational context has become a major cybersecurity challenge. Traditional vulnerability scoring systems primarily rely on generic severity metrics and often fail to capture contextual factors such as network topology, asset accessibility, dependencies between exploitation steps, and the strategic importance of compromised resources.

This thesis investigates the use of Reinforcement Learning (RL) to model attacker behavior as a sequential decision-making process, with the objective of assessing attack surfaces and identifying critical paths in networked systems. The proposed approach aims to provide context-aware vulnerability prioritization by identifying vulnerabilities that play a dominant role in high-risk attack scenarios. Unlike approaches relying on the prior enumeration of attack graphs, the proposed method directly learns exploration strategies through simulated environments, avoiding exhaustive enumeration of possible attack paths during training. The resulting policies provide adaptive, learning-based heuristics whose fast execution enables scalable risk analysis.

To achieve this objective, the thesis introduces an automated simulation framework capable of generating realistic network scenarios from vulnerability data collected from real-world environments. This framework aims to reduce the gap between simulation and reality while remaining compatible with RL model training. It leverages vulnerability metadata and expert knowledge to represent the preconditions, effects, and dependencies associated with exploitation actions. When expert-provided metadata is unavailable, the thesis proposes an automated annotation and enrichment framework based on multiple cybersecurity taxonomies. This facilitates the integration of new vulnerabilities into the simulation environment while reducing manual effort.

The thesis then develops scalable RL methods for combinatorial optimization on heterogeneous graphs, with the ability to generalize across diverse instances and scale to larger problem sizes within the same problem domain. These methods are tailored to the critical attack path discovery benchmark while introducing general design principles applicable to other graph-based optimization problems. Finally, the thesis investigates how learned attack policies can be leveraged to inform mitigation strategies through a multi-agent RL framework that jointly models offensive and defensive decision-making, using secure virtual machine placement as a case study.

Keywords: Reinforcement learning, Automated attack path analysis, Graph learning, Networked systems, Graph combinatorial optimization

Vulgarized version: Digital infrastructures are becoming increasingly complex, and their evolution is accompanied by a growing number of vulnerabilities exploitable by cyber attackers. However, not all vulnerabilities present the same level of risk: a weakness can become critical when it enables the compromise of essential components within a networked system.
This thesis investigates how artificial intelligence, and more specifically reinforcement learning, can contribute to understanding and anticipating attack strategies in deployed networks and applications. The work introduces realistic simulation environments and scalable learning methods to identify critical attack paths. These paths enable vulnerability prioritization based on their contextual impact rather than their severity alone. Finally, the thesis presents mechanisms for integrating these results into operational contexts.

Résumé

Les infrastructures des systèmes en réseau ne cessent de croître en taille, en hétérogénéité et en complexité, rendant difficile l'identification des vulnérabilités réellement critiques dans un environnement opérationnel donné. Cette évolution représente un défi majeur en cybersécurité. Les méthodes traditionnelles de notation reposent principalement sur des métriques génériques de sévérité et intègrent encore insuffisamment des facteurs contextuels tels que la topologie du réseau, l'accessibilité des actifs, les dépendances entre étapes d'exploitation et l'importance stratégique des ressources compromises.
Cette thèse explore l'utilisation de l'apprentissage par renforcement (RL) pour modéliser le comportement d'un attaquant comme un processus de décision séquentielle, afin d'évaluer la surface d'attaque et d'identifier des chemins critiques dans les systèmes en réseau. Cette approche vise à contextualiser la priorisation des vulnérabilités en déterminant celles qui jouent un rôle prépondérant dans les scénarios d'attaque à haut risque. Contrairement aux approches basées sur l'énumération préalable des graphes d'attaque, la méthode proposée apprend directement des stratégies d'exploration via des environnements simulés, évitant ainsi l'énumération exhaustive des chemins possibles durant l'apprentissage. Les politiques obtenues fournissent des heuristiques adaptatives issues de l'apprentissage, dont l'exécution rapide permet une analyse évolutive des risques.
Pour atteindre cet objectif, la thèse introduit un cadre de simulation automatisé générant des scénarios réseau réalistes à partir de données de vulnérabilités issues d'environnements réels, afin de réduire l'écart entre simulation et réalité tout en restant compatible avec l'entraînement de modèles de RL. Il exploite les métadonnées de vulnérabilités et les connaissances d'experts afin de représenter les préconditions, effets et dépendances des actions d'exploitation. En l'absence de métadonnées expertes, la thèse propose un cadre d'annotation automatisée et d'enrichissement à partir de plusieurs taxonomies de cybersécurité, facilitant l'intégration de nouvelles vulnérabilités dans l'environnement de simulation tout en réduisant l'intervention manuelle.
Elle développe ensuite des méthodes de RL capables de passer à l'échelle et de se généraliser à des problèmes d'optimisation combinatoire sur des graphes hétérogènes. Ces méthodes sont appliquées à la découverte de chemins d'attaque critiques, tout en fournissant un cadre adaptable à d'autres problèmes d'optimisation combinatoire sur graphes. Enfin, la thèse explore l'utilisation des politiques d'attaque apprises pour guider des stratégies de mitigation via un cadre d'apprentissage par renforcement multi-agents intégrant décisions offensives et défensives, avec le placement sécurisé de machines virtuelles comme cas d'étude.

Mots-clés: Apprentissage par renforcement, Analyse automatisée des chemins d'attaque, Apprentissage sur graphes, Systèmes réseau, Optimisation combinatoire sur graphes

Version vulgarisée : Les infrastructures numériques deviennent de plus en plus complexes, et leur évolution s'accompagne d'une augmentation des vulnérabilités exploitables par des cyberattaquants. Toutefois, toutes les vulnérabilités ne présentent pas le même niveau de risque : une faiblesse peut devenir critique lorsqu'elle permet de compromettre des composants essentiels d'un système en réseau.
Cette thèse étudie comment l'intelligence artificielle, et plus particulièrement l'apprentissage par renforcement, peut contribuer à comprendre et anticiper les stratégies d'attaque dans les

réseaux et applications déployés. Elle propose des environnements de simulation réalistes et des méthodes d'apprentissage évolutives permettant d'identifier les chemins d'attaque critiques. Ces derniers permettent de prioriser les vulnérabilités selon leur impact contextuel plutôt que leur seule sévérité. La thèse présente enfin des mécanismes permettant l'intégration de ces résultats dans des contextes opérationnels.