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

UNIVERSITÉ DE LORRAINE

École doctorale IAEM Lorraine

Toward Adaptive and Robust Machine Learning Based Network Intrusion Detection Systems

THÈSE

présentée et soutenue publiquement le 11 mai 2026

pour l'obtention du

Doctorat de l'Université de Lorraine

(mention informatique)

par

Omar Anser

Composition du jury

Président :	Bernardetta Addis	Professeure, Université de Lorraine, LORIA
Rapporteurs :	Nadjib Aït-Saadi	Professeur des universités, UVSQ, Université Paris-Saclay, DAVID
	Francesco Bronzino	Maître de conférences HDR, ENS de Lyon, LIP
Examinateur :	Guillaume Hiet	Professeur, CentraleSupélec, Inria, équipe SUSHI
Invité :	Grégory Blanc	Maître de conférences, Télécom SudParis, Institut Polytechnique de Paris, SAMOVAR
Encadrants :	Isabelle Chrisment	Professeure, TELECOM Nancy, Université de Lorraine, LORIA
	Jérôme François	Research scientist, University of Luxembourg, SnT, SEDAN

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

Mis en page avec la classe thesul.

Remerciements

Toutes les bonnes choses ont une fin, même les meilleures. Cette thèse fut une expérience très enrichissante, et je souhaite clore cette étape par ces quelques paragraphes pour remercier les personnes avec qui j'ai partagé cette aventure et qui ont été d'une aide précieuse à l'aboutissement de ce travail.

Je tiens tout d'abord à remercier mes deux directeurs de thèse, Jérôme François et Isabelle Chrisment, pour leur aide, leur accompagnement et leur soutien constant. Merci pour la liberté que vous m'avez octroyée dans la réalisation de mes travaux. Je vous serai toujours reconnaissant pour la patience dont vous avez fait preuve lors des périodes où "je cherchais".

Je remercie ensuite les rapporteurs et les examinateurs de cette thèse pour le temps qu'ils ont consacré à l'évaluation de mon travail et pour avoir accepté de faire partie de mon jury. Mes remerciements s'adressent également à Ye-Qiong Song et Miguel Couceiro, les membres de mon comité de suivi individuel.

Mes remerciements vont aussi à mes collègues, membres ou non de l'équipe RESIST. C'était sympa. Un merci tout particulier à Matthews Jose, le numéro 1, à Santiago, le numéro 2, et à Franco Terranova, le numéro 3. Merci à Enzo, qui mérite une mention à part entière en tant que figure responsable et père de famille. Merci à JP, c'était vraiment cool. Merci à Karim parce que je n'ai pas le choix. Merci à Nicolas pour le café. Merci à Aurélie parce que je n'ai pas le choix non plus. Merci à Xavier, le seul, l'unique, et à Philippe pour les moments de sérieux. Merci également à Nacira, une personne formidable que j'ai eu la chance de croiser au cours de ma thèse. Merci à Asma pour sa gentillesse. Merci également aux permanents de l'équipe, et tout particulièrement à Abdelkader Lahmadi pour les discussions intéressantes que nous avons partagées. La liste étant longue et craignant d'oublier certains noms, que toutes les personnes de l'équipe non citées trouvent ici l'expression de ma sincère gratitude. Je voudrais aussi exprimer ma reconnaissance à Virginie, des ressources humaines d'Inria, pour son aide lors de mes galères administratives.

Je remercie enfin mes amis, mes parents, ma sœur et mon frère pour leur présence, leur accompagnement et leur soutien sans faille. J'associe également à ces remerciements mes deux tantes pour leurs encouragements constants.

À vous toutes et tous, un immense merci.

Résumé

Contexte général

Les systèmes de détection d'intrusion réseau (NIDS, pour *Network Intrusion Detection Systems*) constituent un élément central des architectures de défense modernes. Leur rôle est d'observer le trafic, d'en extraire des signaux pertinents et de signaler les comportements susceptibles de correspondre à une activité malveillante. Historiquement, ces systèmes se sont largement appuyés sur des signatures, des règles et des modèles écrits par des experts. Cette approche reste utile lorsque les attaques sont bien connues et que leurs traces sont stables, mais elle atteint rapidement ses limites lorsque les comportements changent, lorsque les infrastructures évoluent ou lorsque les attaquants adaptent leurs techniques pour contourner les règles existantes.

L'apprentissage automatique a donc été largement étudié pour compléter ces approches. Les modèles entraînés à partir d'exemples peuvent apprendre des régularités, détecter des écarts statistiques et généraliser au-delà des signatures explicitement écrites. Dans la littérature, cette famille de NIDS obtient souvent d'excellentes performances sur des jeux de données publics. Ces résultats ont contribué à établir l'idée que l'apprentissage automatique pourrait rendre la détection plus flexible, plus apte au passage à l'échelle et mieux adaptée aux nouvelles attaques.

Ces résultats doivent toutefois être interprétés avec prudence. De nombreux résultats expérimentaux sont obtenus selon un protocole relativement statique, dans lequel un modèle est entraîné sur un jeu de données, évalué sur un jeu de test séparé, puis comparé à d'autres modèles à l'aide de métriques agrégées. Ce protocole facilite la comparaison des méthodes, mais il simplifie fortement les conditions de déploiement. Dans un réseau réel, les applications utilisées changent, de nouveaux services sont déployés, le volume de trafic peut augmenter ou diminuer, les politiques de sécurité sont mises à jour et les attaquants adaptent leurs stratégies. Un détecteur performant au moment de son évaluation initiale peut donc perdre en efficacité lorsque le contexte opérationnel évolue.

Cette thèse part de ce décalage entre l'évaluation expérimentale et l'usage opérationnel. Elle défend l'idée que la robustesse d'un NIDS ne se limite pas à la précision obtenue sur un jeu de test fixe. Elle dépend aussi de la manière dont le détecteur est configuré lorsque le contexte change, et de la qualité du protocole utilisé pour l'évaluer avant son déploiement. Autrement dit, un NIDS robuste doit être capable de maintenir des performances acceptables dans des conditions variables, mais il doit aussi être évalué sur des cas suffisamment divers et difficiles pour révéler ses faiblesses avant qu'elles n'apparaissent en production.

Problématique et objectifs

La première difficulté étudiée concerne la configuration des détecteurs fondés sur l'apprentissage automatique. Le comportement d'un modèle ne dépend pas uniquement de son algorithme ou de

ses données d'entraînement. Il dépend aussi de nombreux choix de configuration, notamment les hyperparamètres, les seuils de décision, les prétraitements, les fonctions de score et les stratégies d'échantillonnage. Ces choix peuvent avoir un impact majeur sur les performances, en particulier pour les détecteurs d'anomalies. Une configuration pertinente dans un contexte donné peut devenir sous-optimale lorsque la distribution du trafic change.

L'optimisation d'hyperparamètres permet en principe de rechercher une bonne configuration, mais elle reste coûteuse. Les approches classiques, telles que la recherche exhaustive, la recherche aléatoire ou l'optimisation bayésienne, nécessitent de nombreuses évaluations du modèle. Dans un contexte de préparation au déploiement ou de reconfiguration fréquente, ce coût peut rendre ces approches difficiles à appliquer en pratique. Le problème est encore plus complexe pour les détecteurs d'anomalies, car les données d'attaque sont souvent rares, non représentatives ou indisponibles au moment de configurer le système. Une méthode de configuration pratique doit donc réduire le coût de calcul tout en tenant compte de la rareté des exemples d'attaque étiquetés.

La seconde difficulté concerne l'évaluation avant déploiement. Les jeux de test publics sont indispensables pour comparer les méthodes, mais ils ne garantissent pas nécessairement que le détecteur sera confronté à des cas difficiles. Un jeu de test peut être trop proche des données d'entraînement, couvrir seulement une partie de l'espace des comportements ou contenir des attaques dont les caractéristiques sont faciles à séparer. Dans ce cas, une excellente performance agrégée peut masquer une fragilité importante. Le détecteur semble robuste parce que le test ne le met pas réellement sous contrainte.

L'objectif général de cette thèse est donc de renforcer deux dimensions complémentaires de la robustesse des NIDS fondés sur l'apprentissage automatique. La première dimension concerne la robustesse de configuration et consiste à choisir rapidement une configuration adaptée lorsque le contexte de déploiement change. La seconde dimension porte sur la robustesse de l'évaluation et vise à déterminer si un jeu de test est suffisamment exigeant et, si nécessaire, à l'enrichir avec du trafic réaliste ciblant les faiblesses potentielles du détecteur.

Ces objectifs se déclinent en trois questions de recherche. Premièrement, il s'agit d'étudier comment automatiser la configuration des NIDS sans relancer une optimisation coûteuse à chaque changement de contexte. Deuxièmement, il s'agit de définir des métriques permettant de caractériser la qualité d'un jeu de test, non seulement en termes de taille ou de distribution globale, mais aussi en termes de diversité et de proximité avec les zones de décision critiques. Troisièmement, il s'agit de proposer une méthode d'augmentation ciblée capable de produire du trafic réseau réaliste et conforme aux protocoles afin d'améliorer la couverture du test.

Analyse de l'état de l'art

La première partie de la thèse examine les fondements des NIDS et les mécanismes d'adaptation proposés dans la littérature. Les systèmes à base de règles représentent une approche mature et interprétable ; ils reposent sur une connaissance explicite des comportements à détecter. Les approches par apprentissage automatique et apprentissage profond cherchent au contraire à apprendre cette connaissance à partir de données. Elles peuvent exploiter différents niveaux d'observation, comme les paquets, les flux, les séries temporelles, les graphes ou les représentations hybrides. Chaque granularité offre un compromis entre coût de collecte, richesse de l'information et difficulté de modélisation.

L'analyse des méthodes d'adaptation montre que de nombreux travaux se concentrent sur la mise à jour du modèle lui-même. Certains systèmes détectent les changements de distribution du trafic, d'autres utilisent l'apprentissage en ligne, le transfert de domaine, l'apprentissage actif, le

pseudo-étiquetage ou l'apprentissage continu. Ces mécanismes sont importants, car ils permettent au détecteur de s'adapter à de nouvelles données. Néanmoins, ils laissent souvent inchangée une partie essentielle du système, à savoir sa configuration. Un modèle peut être réentraîné ou ajusté, mais ses hyperparamètres, son seuil ou son espace de recherche restent fréquemment fixés à partir d'une phase initiale.

Cette observation révèle une limite récurrente de la littérature. Dans de nombreux scénarios, la configuration est traitée comme un détail expérimental ou comme une opération ponctuelle réalisée avant l'évaluation. Pourtant, elle conditionne la capacité du détecteur à séparer correctement le trafic bénin du trafic malveillant. La thèse montre donc que l'adaptation d'un NIDS ne doit pas seulement concerner ses paramètres appris, mais aussi les choix qui déterminent la manière dont il apprend et prend ses décisions.

La seconde partie de l'état de l'art examine la manière dont l'augmentation de données est utilisée dans les NIDS. Trois grandes orientations se dégagent de cette littérature. Certains enrichissent les données d'entraînement pour améliorer les performances, notamment lorsque les classes sont déséquilibrées ou que certains types d'attaques sont rares. D'autres cherchent à générer un trafic plus réaliste, en respectant les contraintes des protocoles et la structure temporelle des échanges. Enfin, une partie des méthodes utilise l'augmentation pour tester la robustesse du détecteur, en produisant des cas difficiles que le jeu de test initial ne couvre pas suffisamment. Cette thèse s'inscrit dans ce dernier objectif, avec l'idée que l'enrichissement du test doit rester réaliste et utile pour révéler des faiblesses opérationnelles.

Méthode de configuration par méta-apprentissage

La première contribution de la thèse est une méthode de configuration des NIDS par méta-apprentissage. Elle réutilise l'expérience accumulée lors d'optimisations passées afin de prédire rapidement une configuration adaptée à un nouveau contexte. Au lieu de considérer chaque déploiement comme un problème entièrement nouveau, la méthode apprend une relation entre les caractéristiques d'un contexte réseau et les configurations qui ont bien fonctionné dans des contextes similaires.

La méthode est organisée en deux phases. La phase hors ligne construit une base de connaissances à partir d'épisodes de configuration précédents. Chaque épisode associe un contexte, décrit par des méta-caractéristiques, à une configuration de référence obtenue par une procédure d'optimisation coûteuse. Ces méta-caractéristiques résument la structure statistique des données observées, notamment les propriétés de distribution, la séparabilité approximative, la complexité du problème ou les caractéristiques liées aux classes lorsque les étiquettes sont disponibles. Cette base de connaissances sert ensuite à entraîner un modèle prédictif, nommé méta-modèle dans la suite. Ce méta-modèle apprend à relier les méta-caractéristiques d'un contexte aux configurations qui ont donné de bons résultats dans des contextes comparables.

La phase en ligne intervient lorsqu'un nouveau contexte doit être traité. Plutôt que de relancer une optimisation complète, le système calcule les méta-caractéristiques du nouveau jeu de données et utilise le méta-modèle entraîné pendant la phase hors ligne. La configuration est ainsi obtenue en une seule étape, ou avec un coût très réduit, ce qui permet de diminuer fortement le temps de reconfiguration. Cette approche est particulièrement adaptée aux environnements où l'on dispose d'un historique d'expériences de configuration et où les changements de trafic rendent nécessaire une adaptation régulière.

Un point important de la thèse concerne les détecteurs d'anomalies entraînés uniquement sur du trafic bénin. Dans de nombreux scénarios réalistes, il est difficile, voire impossible, de

disposer d'un jeu de validation contenant des attaques représentatives au moment de configurer le détecteur. Les approches classiques d'optimisation sont alors fragilisées, car elles évaluent les configurations à partir d'une mesure de performance qui suppose souvent la présence d'exemples malveillants. La thèse propose donc d'étendre cette méthode de configuration afin d'inférer conjointement les hyperparamètres et le seuil de décision à partir d'informations disponibles sur le trafic bénin.

Cette contribution transforme la configuration en un problème supervisé au niveau méta. Le système apprend, à partir d'expériences passées, quelles configurations tendent à être efficaces pour des contextes présentant certaines propriétés. En phase de déploiement, il n'a plus besoin de tester de nombreuses configurations sur le nouveau contexte. Il s'appuie sur la similarité entre contextes pour transférer une connaissance de configuration. Cette stratégie ne prétend pas fournir une configuration optimale dans tous les cas, mais elle cherche à obtenir rapidement une configuration proche de l'optimum, avec un coût nettement inférieur à celui d'une optimisation exhaustive.

Analyse empirique de la robustesse de configuration

La thèse complète la méthode de méta-apprentissage par une étude empirique de la robustesse de configuration. L'objectif est de quantifier l'effet d'une configuration fixe lorsque le contexte change. Cette question est essentielle, car de nombreux protocoles expérimentaux supposent implicitement que la configuration trouvée sur un jeu de données ou une période donnée reste valable sur d'autres jeux de données ou à d'autres périodes. En pratique, cette hypothèse peut être fausse.

L'étude s'appuie sur les ensembles de données publics IDS2017 et IDS2018, largement utilisés. Leur organisation par journées de collecte fournit plusieurs contextes expérimentaux. Nous exploitons ces contextes pour évaluer si une configuration reste stable lorsque le trafic et les attaques varient d'une journée à l'autre. Nous comparons alors plusieurs stratégies de configuration, notamment la réutilisation d'une configuration par défaut, le transfert d'une configuration apprise dans un autre contexte et le recours à une configuration de référence issue d'une optimisation plus coûteuse.

Les résultats montrent que la réutilisation d'une configuration peut entraîner une baisse significative des performances lorsque le contexte change. Certaines configurations restent relativement stables, mais d'autres se dégradent fortement. Cette variabilité confirme que la robustesse ne dépend pas seulement du choix du modèle, mais aussi de la stabilité de sa configuration face aux changements de distribution. Une configuration performante sur une journée ou sur un type de trafic n'est pas nécessairement robuste sur une autre journée ou sur un autre jeu de données.

Cette étude justifie l'intérêt d'une reconfiguration automatisée. Elle montre que l'optimisation ponctuelle ne suffit pas si le détecteur est destiné à être utilisé dans un environnement dynamique. Elle souligne aussi la nécessité de protocoles expérimentaux qui évaluent la sensibilité à la configuration, plutôt que de comparer uniquement des scores obtenus après optimisation. En ce sens, la thèse contribue à déplacer l'attention de la performance maximale vers la performance maintenue malgré les changements de contexte.

Méthode d'évaluation et d'augmentation ciblée des jeux de test

La deuxième contribution principale est TATA, acronyme de *Test-set Assessment and Targeted Augmentation*. TATA vise à évaluer la qualité d'un jeu de test et à l'enrichir lorsqu'il ne met pas

suffisamment le détecteur sous contrainte. Le principe est le suivant. Un bon jeu de test ne se limite pas à des exemples qui n'ont pas servi à l'entraînement. Il doit aussi placer le détecteur face à des situations variées et difficiles. Pour évaluer cette exigence, TATA projette les données d'entraînement et de test dans un espace latent structuré, plus exploitable que l'espace brut des caractéristiques. Dans cet espace, la qualité du jeu de test est décrite par trois propriétés complémentaires. La diversité mesure la redondance des exemples, la proximité identifie les cas situés près de zones où les comportements sont difficiles à séparer, et la couverture indique si le jeu de test parcourt l'ensemble des frontières de décision. Ces métriques indépendantes du détecteur évalué complètent les mesures classiques de performance, comme la précision et le rappel. Elles aident à distinguer un résultat solide, obtenu sur un test exigeant, d'un résultat plus fragile, obtenu sur un test trop simple pour révéler les faiblesses du détecteur.

Lorsque les métriques indiquent que le jeu de test manque de couverture ou de difficulté, TATA déclenche une phase d'augmentation ciblée. Cette phase utilise un agent d'apprentissage par renforcement connecté à un banc d'essai réseau. L'agent configure des générateurs de trafic afin de produire de nouveaux flux qui ciblent les régions insuffisamment couvertes ou proches des frontières. Le but n'est pas de générer n'importe quel point difficile dans l'espace des caractéristiques, mais de produire un trafic réaliste et conforme aux contraintes protocolaires.

Cette exigence de réalisme distingue TATA d'une augmentation purement synthétique. Pour l'évaluation d'un NIDS, un exemple utile doit pouvoir correspondre à un trafic observable sur un réseau réel. Les paquets doivent respecter les formats attendus, les champs dépendants doivent rester cohérents et les flux doivent conserver une logique temporelle crédible. Sinon, le détecteur peut être évalué sur des artefacts qui ne seraient jamais observés en production. TATA cherche donc à relier la génération de cas difficiles à un environnement de test contrôlé, afin de préserver le lien entre difficulté statistique et plausibilité opérationnelle.

Évaluation expérimentale de la configuration par méta-apprentissage et de TATA

Les deux contributions sont évaluées expérimentalement sur des NIDS orientés flux et sur des jeux de données publics couramment utilisés dans la littérature. Le choix des données IDS2017 et IDS2018 permet de comparer les méthodes dans un contexte expérimental connu, tout en étudiant les limites de ces références expérimentales. Ces ensembles de données offrent une variété d'attaques et de périodes de collecte, mais ils présentent aussi des biais, des répétitions et des structures qui peuvent influencer l'évaluation. La thèse les utilise donc à la fois comme support expérimental et comme objet d'analyse critique.

Pour la configuration, les expériences comparent les configurations prédites par ce méta-modèle à des références obtenues par optimisation. Les résultats mettent en évidence une réduction importante du coût de configuration, avec des performances proches des références dans de nombreux scénarios. Cette réduction du coût est essentielle pour rendre la reconfiguration plus réaliste. Elle permet d'envisager un usage où le système s'adapte régulièrement à de nouveaux contextes sans nécessiter d'importantes ressources de calcul.

Pour l'évaluation, les expériences menées avec TATA analysent la qualité des jeux de test et mesurent l'effet de l'augmentation ciblée. Les résultats montrent que les jeux de test ne sollicitent pas tous le détecteur de la même manière. Certains couvrent mieux l'espace latent, tandis que d'autres contiennent moins de cas proches des frontières. L'augmentation ciblée permet d'ajouter des exemples qui augmentent la difficulté et mettent en évidence des zones de fragilité. Cette démarche complète les mesures de performance globales, car elle aide à comprendre les conditions

dans lesquelles un détecteur réussit ou échoue.

Pris ensemble, ces résultats appuient le message central de la thèse. La robustesse opérationnelle d'un NIDS fondé sur l'apprentissage automatique dépend d'un cycle complet qui associe configuration, entraînement, évaluation et reconfiguration. Optimiser seulement le modèle ou seulement le score final ne suffit pas. Il faut aussi s'assurer que la configuration reste adaptée et que le test exerce une pression suffisante sur le détecteur.

Conclusion

En conclusion, cette thèse montre que la robustesse d'un NIDS fondé sur l'apprentissage automatique ne se résume pas au choix d'un algorithme. Elle dépend aussi de la manière dont le détecteur est configuré, évalué et adapté lorsque les conditions réseau changent. Les contributions proposées vont dans ce sens en reliant configuration automatisée, analyse empirique et qualité des jeux de test.

La principale limite tient au périmètre des validations. Les résultats reposent sur des détecteurs, des jeux de données et des contextes expérimentaux précis, qui ne couvrent pas toute la diversité des réseaux réels ni toutes les familles d'attaques.

Les perspectives consistent à rapprocher ces contributions d'un usage plus continu et plus opérationnel. Cela implique notamment de mieux quantifier l'incertitude des configurations proposées, d'articuler évaluation et reconfiguration au fil du temps, et de poursuivre le travail sur des tests plus réalistes, capables de révéler les faiblesses des détecteurs sans perdre le lien avec le trafic observable en production.

Contents

Résumé **iii**

Introduction **1**

1 Context . . . 1
1.1 Challenges and Problem Statement . . . 3
1.2 Objectives and Research Questions . . . 4
2 Thesis Contributions . . . 5
3 Thesis outline . . . 6
4 Publications . . . 8
4.1 International conferences and symposia . . . 8

Partie I Adaptability in Machine-Learning-Based Network Intrusion Detection Systems **9**

1 Toward Adaptable Network Intrusion Detection Systems **11**
1.1 Introduction . . . 11
1.2 Rule-Based (Signature-Based) Detection Methods . . . 12
1.2.1 Misuse Detection . . . 13
1.2.2 Specification-Based Detection . . . 13
1.2.3 Threshold-Based Anomaly Detection . . . 13
1.2.4 Scaling and Automating Rule-Based Detection Methods . . . 14
1.2.5 Limitations of Rule-Based Detection Methods . . . 15
1.3 ML- and DL-based Detection Methods . . . 15
1.3.1 ML- and DL-based NIDS Typical Detection Tasks . . . 15
1.3.2 ML- and DL-based NIDS Pipelines . . . 16
1.3.3 Traffic Observation Granularity . . . 17
1.3.4 Configuration of ML/DL-based NIDS . . . 20
1.3.5 Limitations of Conventionally Trained ML/DL Models . . . 22

1.4 Adaptation Methods for ML/DL-Based Detection Methods 22
1.4.1 Drift Detection Mechanisms 23
1.4.2 Online and Continual-Learning Methods 25
1.4.3 Active and Semi-Supervised Label Acquisition 28
1.4.4 Transfer Learning and Domain Adaptation 30
1.4.5 Meta-Learning Frameworks 33
1.4.6 Threshold and Score Calibration 35
1.4.7 Synthesis of Adaptation Methods for ML/DL-based NIDS 37
1.5 Conclusion 39

2 A Meta-Learning Framework for Automated NIDS Configuration 41
2.1 Introduction 41
2.2 NIDS Configuration Problem Formulation 42
2.2.1 ML/DL NIDS Lifecycle and Notation 42
2.2.2 Hyperparameter Optimization Methods for NIDS 46
2.2.3 Limitations of Conventional Hyperparameter Search for NIDS 48
2.3 Meta-learning-based Auto-configuration 48
2.3.1 Threat Model 48
2.3.2 Overview 49
2.3.3 Offline Phase: Meta-modeling and Objective 50
2.3.4 Online Phase: On-the-fly Configuration 53
2.4 Conclusion 53

3 Evaluation of the Auto-Configuration Framework 55
3.1 Introduction 55
3.2 Experimental Setup 56
3.2.1 NIDS Design and Implementation 56
3.2.2 Datasets 58
3.2.3 Nested Cross-Validation for Oracle Configurations 60
3.2.4 Evaluation Metrics for Classification and Score-based Detection . . . 61
3.3 Empirical Evaluation of the Meta-Learning Framework for NIDS Classification 63
3.3.1 Detection Performance 63
3.3.2 Resource usage at Deployment Stage 65
3.3.3 Meta-dataset size 67
3.4 Empirical Evaluation of the Meta-Learning Framework for AD NIDS Configuration 67
3.4.1 Robustness 68

3.4.2 Anomaly Detection . . . 72
3.4.3 Configuration Generation Time . . . 73
3.4.4 Meta-learning Relative Performance . . . 73
3.5 Conclusion . . . 74

Partie II Test-Set Assessment and Traffic Augmentation for ML-Based Network Intrusion 77

4 Data Augmentation for NIDS 79
4.1 Introduction . . . 79
4.2 Motivation for Data Augmentation in NIDS . . . 80
4.2.1 Structural and Design Limitations . . . 80
4.2.2 Content-Level Pitfalls and Artifacts . . . 81
4.3 Data Augmentation for NIDS . . . 83
4.3.1 Performance-Oriented Data Augmentation Techniques . . . 83
4.3.2 Realism-Oriented Traffic Augmentation . . . 85
4.3.3 Test-Set Augmentation for NIDS Robustness Testing . . . 87
4.4 Conclusion . . . 93

5 TATA: An Approach for Assessing Benchmark NIDS Test-Sets and Targeted Augmentation 95
5.1 Introduction . . . 95
5.2 TATA Design . . . 98
5.2.1 Overview . . . 98
5.2.2 Optimized Latent Representation for Training Set . . . 100
5.2.3 Test Set Quality Assessment . . . 100
5.2.4 Test Set Targeted Augmentation . . . 104
5.3 Conclusion . . . 105

6 Evaluation of TATA 107
6.1 Introduction . . . 107
6.2 TATA Evaluation Environment and Implementation Details . . . 108
6.2.1 Datasets . . . 108
6.2.2 NIDS Models . . . 108
6.2.3 Contrastive Autoencoder . . . 109
6.3 Preliminary Phase: Evaluating the Contrastive Autoencoder . . . 109
6.4 Phase 1: Test Set Assessment Metrics in Practice . . . 110

6.4.1 Diversity 110
6.4.2 Proximity 110
6.4.3 Scarcity 112
6.5 Phase 2: Targeted Augmentation Evaluation 112
6.6 Datasets Analysis 116
6.7 Discussion of Results 117
6.8 Conclusion 119

Conclusion **121**

List of Figures **129**

List of Tables **133**

Annexes **135**

A Première Annexe **137**

Bibliography **147**

Résumé **171**

Abstract **172**

Introduction

1 Context

In today's interconnected world, organizations and public administrations depend on computer networks for almost every essential task. From high-value financial transactions and real-time remote collaboration to the protection of large volumes of personal data, network infrastructures are now a core component of modern society.

This dependence is visible in everyday practices. Across Europe, more than half of enterprises (52.9%) already held their meetings online in 2024 [1]. Public administrations show a similar pattern. In France, citizens completed 573 million administrative procedures entirely online in 2023, corresponding to about 82% of all requests, and 43 million people now use the FranceConnect digital-identity platform to access more than 1,800 public services [2].

The extensive scale and widespread presence of networked services expand the network attack surface. France's national cybersecurity agency, ANSSI, handled 4,386 security events in 2024, a 15% increase compared with 2023. Among these, 1,361 were confirmed intrusions into victim systems, including 144 ransomware incidents in which systems were fully encrypted and a ransom was demanded [3]. The impact is not only technical or financial; it also has direct human consequences. In France's health sector, 581 cyber incidents were reported in 2023, and in 35% of them, patient care was directly affected; one incident posed a proven threat to patient safety and 68 others were classified potentially dangerous [4].

Economic costs are equally significant. IBM's *Cost of a Data Breach Report 2024* estimates the global average cost of a data breach across all industries at 4.88 million US dollars. Costs are substantially higher in healthcare, where the average breach cost reached 9.77 million US dollars in 2024, the highest among all sectors, a position held since 2011 [5].

To mitigate escalating security risks, organizations frequently establish dedicated units, such as Security Operations Centers (SOCs), to monitor their infrastructures continuously. This monitoring framework often uses centralized monitoring technologies, including Security Information and Event Management (SIEM) systems, to aggregate heterogeneous telemetry from application

[1] Eurostat, *Digital economy and society statistics – enterprises*, data extracted February 2025. Available at `https://ec.europa.eu/eurostat`.

[2] Direction interministérielle du numérique (DINUM), *Observatoire de la qualité des démarches en ligne*, Comité interministériel de la transformation publique, April 2023. Summary and datasets at `https://observatoire.numerique.gouv.fr`.

[3] ANSSI, *Panorama de la cybermenace 2024*, February 2025, pp. 4–5 and 12–13. PDF: `https://www.cert.ssi.gouv.fr/uploads/CERTFR-2025-CTI-003.pdf`.

[4] Agence du Numérique en Santé (ANS), *Observatoire des incidents de sécurité des systèmes d'information pour les secteurs santé et médico-social 2023*, communiqué du 22 mai 2024; PDF `https://cyberveille.esante.gouv.fr/sites/default/files/media/document/2024-05/observatoire-incidents-cybersecurite-sante-2023.pdf`.

[5] IBM Security and Ponemon Institute, *Cost of a Data Breach Report 2024*, July 2024, Executive Summary.

logs, endpoint sensors, and network probes. Within this architecture, intrusion detection serves as a critical analytical layer, converting raw data into concrete indicators. This process not only provides early warning functions but also facilitates investigation and response.

Among these telemetry sources, network traffic offers a critical perspective. It captures the actual exchanges between systems, including communications that are missing from standard logs or originate from unmonitored devices. Network-based Intrusion Detection Systems (NIDS) leverage this access by analyzing traffic at strategic locations, such as the network perimeter or the boundaries of sensitive segments like data centers, to identify malicious patterns and potential intrusions. The critical operational role of these systems is reflected in growing investment in these technologies. Market analysis estimates that global investment in NIDS reached approximately US $4.5 billion in 2024, with projections exceeding US $8 billion by 2033 (a CAGR of 7.4%) [6].

For many years, most operational network detectors were built around explicit rules [Axe00, Roe99, Pax99]. These systems use signatures that define known attack patterns or protocol violations to scan traffic efficiently, triggering an alert whenever a match is identified. While this approach remains relevant for its deterministic behavior and interpretability for known threats [Axe00], it faces significant challenges with evolving risks; new attack variants quickly evade static signature databases [SP10a]. Encrypted payloads further reduce the visibility of content-based inspection, making signature matching harder in practice [SLPR15]. To address these inherent limitations, the field has increasingly investigated machine learning (ML) and deep learning (DL) methods [SP10a, BG16, KGVK19]. Unlike signature-based approaches, ML-based NIDS employ data-driven models to learn detection rules from past observations and generalize beyond known signatures [SP10a, BG16]. They exploit diverse input representations ranging from packet- or flow-level features to complex communication graphs [YZFH17b, FOS23].

Leveraging these data-driven approaches is widely considered a key step toward the development of adaptive NIDS [SIZ+24], systems capable of autonomously adapting to both novel threats and changes in legitimate traffic patterns. This shift is also visible in industrial deployments. This widespread adoption is quantified in the *SANS 2024 Detection & Response Survey* [7]: in a sample of almost 400 practitioners drawn from banking/finance, government, education, and cybersecurity, 51.2% report using ML algorithms for threat detection, while 38.9% rank Artificial Intelligence (AI)/ML among their current threat detection methods. In parallel, the scientific community is directing significant resources toward advancing these adaptive capabilities. For instance, France's PEPR Cybersécurité programme allocates €65 million over six years, with its main *SuperviZ* project focused explicitly on next-generation NIDS and security supervision [8].

Accordingly, over the years, the ML-based NIDS research community has proposed a wide range of methods to improve both the detection performance and the practical deployment of such systems. Early studies showed that supervised learning methods [AYL+11], applied to traffic features (i.e., measurable characteristics extracted from network communications such as packet size, flow duration, or connection counts) could already provide effective intrusion detection

[6] Verified Market Reports, *Network Intrusion Detection System (NIDS) Market*, February 2025. Coverage spans North America, Europe, Asia–Pacific, Latin America, and the Middle East & Africa, and includes hardware, software, and cloud-delivered NIDS revenue.

[7] J. Lemon, *SANS 2024 Detection and Response Survey: Transforming Cybersecurity Operations: AI, Automation, and Integration in Detection and Response*, SANS Institute, November 2024. Available at `https://www.sans.org/white-papers/sans-2024-detection-response-survey` (accessed 29 Apr 2025).

[8] CEA, *7 projets retenus dans le cadre du PEPR cyber*, press release, 22 June 2022. Available at `https://www.cea.fr/presse/Pages/actualites-communiques/ntic/pepr-cyber-presentation.aspx` (accessed 29 Apr 2025). See also: ANR, *PEPR Cybersécurité: projets ciblés*, 12 Apr 2023, and the project description page *SuperviZ* on the PEPR Cybersécurité website (accessed 29 Apr 2025).

while reducing the effort required to manually design expert rules. Later, deep learning (DL) approaches [YZFH17a] were introduced to learn richer representations either directly from network traffic or from minimal feature sets, further reducing the need for feature engineering. Beyond model design itself, the literature also explored several complementary directions to address different limitations of ML-based NIDS, including lightweight and online detection [MDES18], better pre-deployment evaluation protocols [ALS23], and robustness against adversarial manipulation [AMB21].

Within this broader research effort, a particularly important line of work focuses on the intrinsic weaknesses of these data-driven systems. These weaknesses arise directly from the way the models are trained: they extract patterns and establish decision boundaries from finite collections of past network traffic observations, under the assumption that the statistical properties of future traffic will remain consistent with these historical distributions [SP10a]. In practice, however, this assumption is rarely met. Operational networks are dynamic environments where traffic evolves due to protocol updates, new application deployments, and shifting usage patterns (for example, the adoption of a new encrypted protocol can change observable feature distributions) [SIZ+24]. Furthermore, attackers may adapt their techniques after failed attempts, which can further shift the data distribution seen by the detector [SIZ+24]. As a result, the literature has proposed a set of adaptive learning strategies that enable models to adjust to evolving network environments [SIZ+24].

In parallel with the development of these adaptive mechanisms, another fundamental research direction concerns their empirical validation. In this context, public datasets and benchmarks have become central evaluation resources, as they allow the standardization of experimental settings and the reproducible comparison of methods. Prominent examples include CIC-IDS2017 (IDS2017) and CSE-CIC-IDS2018 (IDS2018) [SHLG18a]. In this direction, existing literature has examined validation best practices, the design of evaluation frameworks that better reflect operational constraints, the use of cross-dataset assessments to probe broader deployment conditions, and the characterization of public datasets for selecting appropriate benchmarks [AQP+22, ALS23, APC22, RWS+19a]. Complementary works have focused on the quality of the datasets themselves [FEAD24a], establishing the critical finding that these collections serve only as temporally constrained approximations of the conditions encountered in live networks.

This thesis is situated at the intersection of these evolving research directions. It addresses the problem of building ML-based NIDS that can adapt to changing environments, and of rigorously evaluating these systems prior to real-world deployment to effectively assess their robustness.

1.1 Challenges and Problem Statement

A primary challenge for ML-based NIDS is maintaining robustness as operational traffic evolves and attackers adapt. To address this, a large body of work has proposed mechanisms to make these systems more resilient over time. As surveyed in the first part of this thesis, these approaches include drift detection, online and continual learning, active and semi-supervised label acquisition, transfer learning, and meta-learning for intrusion detection. In practice, they attempt to restore performance by monitoring NIDS outputs, updating weights incrementally, or transferring knowledge between different environments. In addition, threshold calibration methods adjust the decision cutoff of score-based detectors when score distributions shift.

However, across many of these families, the proposed adaptation strategies fundamentally overlook the configuration of the NIDS. The configuration denotes the set of choices made before deployment that determine how the NIDS is built and tuned, such as the input representation

(packet, flow, trace, or graph), the model type, and key settings like hyperparameters and, for anomaly detectors, the decision threshold. For example, two NIDS may use the same learning method, but behave very differently if one uses flow features with a conservative threshold while the other uses packet features with an aggressive threshold.

When traffic evolves, a configuration that was optimal at deployment can become suboptimal, leading to excessive false alarms or missed attacks. A conventional approach is to rerun hyperparameter optimization (HPO), using grid search, random search, or Bayesian optimization, to find a new configuration for the new environment. In practice, this is difficult to repeat whenever the network changes because it requires many costly training and evaluation runs and often depends on labeled data that are not immediately available in live networks.

A second major challenge lies in how we assess the performance and robustness of ML-based NIDS. Current practice largely relies on static train/validation/test splits constructed once from a benchmark dataset, which are then used both to tune models (the validation split) and to report final performance metrics (the test split). This methodology is based on the assumption that the test set accurately represents the complexity of future network environments. In reality, however, standard benchmarks frequently fail to capture the full diversity of operational traffic.

One way to address this limitation is to stress-test ML-based NIDS by augmenting the test set with challenging network traffic while preserving its semantic nature, ensuring that attacks remain attacks and benign samples remain benign.

In the context of NIDS, recent research has begun to explore synthetic network traffic generation with two main goals: performance-oriented augmentation, which mainly addresses class imbalance and the scarcity of certain attack types to improve training; and realism-oriented augmentation, which aims to closely replicate the structural and temporal characteristics of operational networks. The objective is to prevent models from learning the superficial irregularities often found in experimental datasets. However, the literature still lacks a general, model-agnostic method to assess how effectively a test set challenges a NIDS, and to augment those data in a controlled, protocol-respecting manner.

1.2 Objectives and Research Questions

The primary objective of this thesis is to enhance the robustness of ML-based NIDS. This goal is addressed through two complementary dimensions:

Objective O1: Configuration robustness

Enabling the dynamic adaptation of the NIDS's configuration (including hyperparameters and decision thresholds) in response to shifting network contexts.

Objective O2: Assessment robustness

Analyzing and augmenting the test sets used to evaluate ML-based NIDS, so that pre-deployment assessment more reliably reflects how the NIDS will behave in different operational environments.

Concretely, the work is organized around the following research questions.

Research Question 1:

How can we make the configuration of ML-based NIDS adaptive, instead of repeatedly running costly hyperparameter optimization from scratch?

We investigate methods that can leverage knowledge from prior configurations, so that when a NIDS is deployed on a new network or encounters evolving traffic distributions, it can rapidly achieve a near-optimal configuration without the computational overhead of traditional optimization. Additionally, the availability of training data influences the choice of detection model. Consequently, a key requirement is that the method handles both supervised classifiers and unsupervised anomaly detectors, allowing it to remain effective despite the scarcity of labeled attack data in real-world settings.

Research Question 2:

How can we characterize and measure the quality of NIDS benchmark test sets in a way that is model-agnostic?

We aim to define a set of quantitative metrics capable of characterizing the complexity and coverage of a test set relative to its corresponding training data. To ensure broad applicability, these indicators must capture the intrinsic geometry of the traffic distribution. Crucially, this assessment must remain model-agnostic, relying solely on the data structure rather than the internal states or specific architecture of the detector.

Research Question 3:

How can we systematically augment NIDS test sets with realistic network traffic that targets under-covered regions of the decision space and reveals weaknesses of trained detectors?

The goal is to design a targeted augmentation framework that generates realistic, protocol-compliant network traffic to specifically fill identified coverage gaps of a test set. Unlike purely synthetic feature manipulation, this approach seeks to produce valid traffic that systematically exposes the NIDS's weaknesses. The resulting feedback loop is guided by the quality metrics defined in RQ2, ensuring that new samples effectively stress-test the NIDS while maintaining semantic integrity.

2 Thesis Contributions

This thesis makes the following contributions.

• **C1: Survey and gap analysis on robustness in ML-based NIDS.** We systematize the literature on ML-based NIDS, with a focus on mechanisms that aim to maintain performance under changing network conditions. We show that many approaches adapt model parameters or decision thresholds, but typically keep the detector configuration fixed (e.g., hyperparameters and architectural choices). This analysis motivates the need for configuration-aware adaptation strategies. This contribution is mainly presented in Chapter 1.

• **C2: A meta-learning framework for automated configuration of ML-based NIDS.** We formalize the configuration of ML-based NIDS as a meta-learning problem. The framework learns a mapping from operational contexts (dataset and traffic characteristics) to near-optimal hyperparameters and, when needed, decision thresholds. The goal is to avoid rerunning computationally expensive HPO from scratch whenever the context changes, including the practical case where the training dataset is attack-free.

Auto-tuning of Hyper-parameters for Detecting Network Intrusions via Meta-learning
Omar Anser, Jérôme François, and Isabelle Chrisment.
IEEE/IFIP Network Operations and Management Symposium (NOMS), 2023.
DOI: https://doi.org/10.1109/NOMS56928.2023.10154381

Automated Machine Learning Configuration to Learn Intrusion Detectors on Attack-Free Datasets
Omar Anser, Jérôme François, and Isabelle Chrisment.
IEEE Conference on Local Computer Networks (LCN), 2024.
DOI: https://doi.org/10.1109/LCN60385.2024.10639690

• **C3: Empirical study on configuration robustness of unsupervised IDS.** We propose a benchmark protocol on the day-partitioned IDS2017 and IDS2018 datasets to measure how performance degrades when an anomaly-detection IDS keeps a fixed hyperparameter configuration under context drift. Compared to a Bayesian optimization oracle, practical low-budget strategies and simple configuration transfer schemes can lose roughly one-third of their ROC–AUC score. This contribution is mainly presented in Chapter 3.

An Empirical Study on Configuration Robustness of Unsupervised IDS
Omar Anser, Jérôme François, and Isabelle Chrisment.
2026 IEEE/IFIP Network Operations and Management Symposium (NOMS).
HAL: https://hal.science/view/index/docid/5656209

• **C4: TATA, a model-agnostic framework for test-set assessment and targeted augmentation.** We propose a two-phase approach for robustness evaluation of ML-based NIDS. First, we assess test-set quality using model-agnostic metrics computed in a learned representation space (diversity, proximity, and scarcity). Second, we use these metrics to guide targeted traffic generation through a reinforcement learning agent interacting with a testbed, producing protocol-compliant traffic that fills identified coverage gaps. This contribution is mainly presented in Chapters 5 and 6.

TATA: Benchmark NIDS Test Sets Assessment and Targeted Augmentation
Omar Anser, Jérôme François, Isabelle Chrisment, and Daishi Kondo.
European Symposium on Research in Computer Security (ESORICS), 2025.
DOI: https://doi.org/10.1007/978-3-032-07884-1_2

3 Thesis outline

The manuscript is organized into six chapters, excluding the introduction and conclusion. It is structured to mirror the two robustness dimensions defined by the research objectives: configuration robustness and assessment robustness. Figure 1 summarizes the logical flow, linking chapters to research questions and contributions.

Part I: Adaptable Configuration for ML-based NIDS

Part I focuses on configuration robustness. Chapter 1 surveys adaptation mechanisms and highlights the configuration gap. Chapter 2 introduces a meta-learning framework to infer near-optimal configurations for new contexts. Chapter 3 evaluates this approach empirically and studies configuration robustness for unsupervised IDS.

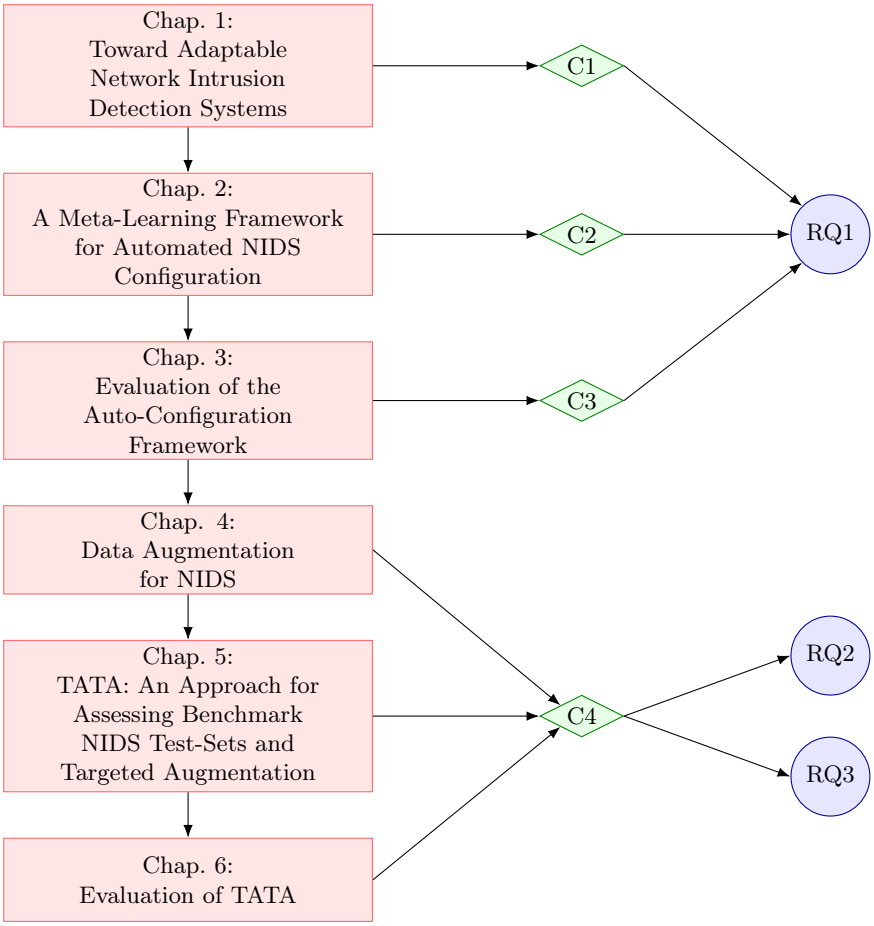


Figure 1: Thesis organization: research questions, contributions, and chapters.

Part II: Test-set Assessment and Targeted Traffic Augmentation
Part II focuses on assessment robustness. Chapter 4 reviews data augmentation approaches for NIDS and clarifies the difference between augmentation for training and augmentation for robustness evaluation. Chapter 5 introduces TATA's model-agnostic quality metrics for test sets. Chapter 6 validates targeted, protocol-compliant augmentation with a reinforcement learning agent interacting with a testbed.

4 Publications

The work conducted during this thesis has led to scientific contributions disseminated through peer-reviewed international conferences.

4.1 International conferences and symposia

- O. Anser, J. François, and I. Chrisment. *Auto-tuning of Hyper-parameters for Detecting Network Intrusions via Meta-learning.* 2023 IEEE/IFIP Network Operations and Management Symposium (NOMS), AnNet Workshop. DOI: `https://doi.org/10.1109/NOMS56928.2023.10154381`.
- O. Anser, J. François, and I. Chrisment. *Automated Machine Learning Configuration to Learn Intrusion Detectors on Attack-Free Datasets.* 2024 IEEE 49th Conference on Local Computer Networks (LCN). DOI: `https://doi.org/10.1109/LCN60385.2024.10639690`.
- O. Anser, J. François, I. Chrisment, and D. Kondo. *TATA: Benchmark NIDS Test Sets Assessment and Targeted Augmentation.* 30th European Symposium on Research in Computer Security (ESORICS 2025). DOI: `https://doi.org/10.1007/978-3-032-07884-1_2`.
- O. Anser, J. François, and I. Chrisment. *An Empirical Study on Configuration Robustness of Unsupervised IDS.* 2026 IEEE/IFIP Network Operations and Management Symposium (NOMS). HAL: `https://hal.science/view/index/docid/5656209`.

Part I

Adaptability in Machine-Learning-Based Network Intrusion Detection Systems

Chapter 1

Toward Adaptable Network Intrusion Detection Systems

Contents

1.1 Introduction 11
1.2 Rule-Based (Signature-Based) Detection Methods 12
1.2.1 Misuse Detection 13
1.2.2 Specification-Based Detection 13
1.2.3 Threshold-Based Anomaly Detection 13
1.2.4 Scaling and Automating Rule-Based Detection Methods 14
1.2.5 Limitations of Rule-Based Detection Methods 15
1.3 ML- and DL-based Detection Methods 15
1.3.1 ML- and DL-based NIDS Typical Detection Tasks 15
1.3.2 ML- and DL-based NIDS Pipelines 16
1.3.3 Traffic Observation Granularity 17
1.3.4 Configuration of ML/DL-based NIDS 20
1.3.5 Limitations of Conventionally Trained ML/DL Models 22
1.4 Adaptation Methods for ML/DL-Based Detection Methods . . 22
1.4.1 Drift Detection Mechanisms 23
1.4.2 Online and Continual-Learning Methods 25
1.4.3 Active and Semi-Supervised Label Acquisition 28
1.4.4 Transfer Learning and Domain Adaptation 30
1.4.5 Meta-Learning Frameworks 33
1.4.6 Threshold and Score Calibration 35
1.4.7 Synthesis of Adaptation Methods for ML/DL-based NIDS 37
1.5 Conclusion 39

1.1 Introduction

A Network Intrusion Detection System (NIDS) is one of the principal mechanisms for detecting malicious activity across an entire computer network. The U.S. National Institute of Standards

and Technology (NIST) defines a NIDS as "*software that performs packet sniffing and network traffic analysis to identify suspicious activity and record relevant information.*"[9] A NIDS is typically connected via a passive network TAP, which physically replicates signals at the link layer; a Switch-Port Analyzer (SPAN), which logically mirrors packets at the network switch; or cloud traffic mirroring, a cloud-native mechanism functionally analogous to a physical TAP or SPAN but explicitly built for virtualized environments.

Detection mechanisms in NIDS have historically been based on explicit expert knowledge, encoded as sets of rules that describe, for example, known malicious behaviors or protocol violations [Roe99]. These rule-based systems match observed traffic against predefined signatures, enabling precise and interpretable detection of attacks whose characteristics are known in advance [SM07]. However, the increasing scale, heterogeneity, and volume of network environments, combined with the growing sophistication of attack techniques, have exposed the limitations of such static representations [SP10b]. Furthermore, the widespread use of encryption reduces visibility into packet payloads, which limits rule-based detection that depends on content inspection.

This evolution of the network landscape motivated the introduction of ML- and DL-based approaches, which make detection more adaptable. In particular, ML- and DL-based NIDS learn generalized patterns from historical network traffic data and construct a model that enables the detection of threats based on behavioral characteristics rather than requiring an exact match to a predefined signature [BG16, CJ22]. In other words, detection moves from case-by-case rule encoding to generalizable traffic-pattern learning. Crucially, once detection is framed as a learnable model, it becomes possible to incorporate explicit adaptation mechanisms that monitor deployment conditions and update the detector over time, for example, through online learning [BCR+18], providing a further step toward practical adaptability.

At the same time, learning-based detection introduces another practical dimension: configuration. Beyond what the model learns from traffic data, ML/DL-based NIDS rely on design choices that must be set explicitly and are not learned automatically. These choices can strongly shape detection performance and also influence how a detector operates once deployed. For example, the choice of traffic window length determines what the NIDS can observe, while other training settings can shift the balance between detecting rare attacks and keeping false alarms low. This means that adaptability is not only a question of whether a system can update its knowledge over time, but also of how the learning process has been configured in the first place.

In what follows, we first review rule-based (signature-based) detection methods in Section 1.2. Next, Section 1.3 surveys ML- and DL-based NIDS, examining their architectural components, common configuration practices, and the limitations inherent in their conventional static deployment. Addressing these limitations, Section 1.4 explores adaptation methods designed to maintain detection performance over time. Finally, we provide a synthesis of these approaches in Section 1.4.7 and conclude the chapter in Section 1.5.

1.2 Rule-Based (Signature-Based) Detection Methods

Rule-based (or signature-based) network intrusion detection relies on explicit rules to distinguish malicious activity from legitimate traffic. In practice, these rules take three main forms: (i) misuse detection, which enumerates byte patterns or protocol states that characterize known attacks; (ii) specification-based detection, which strictly defines how a protocol must behave and flags any deviation; and (iii) threshold-based anomaly detection, which defines a fixed baseline

[9] National Institute of Standards and Technology, *Special Publication 800–86: Guide to Integrating Forensic Techniques into Incident Response*, Glossary, 2006.

of normal behavior (e.g., packet rates) and raises an alert whenever traffic exceeds those limits. All three approaches rely mostly on manually encoded policies, although some advanced systems demonstrate the ability to automatically learn rules directly from network traffic [SY23, KK04, NKS05]. This section first provides an overview of these three rule-based approaches, then discusses methods for scaling and automating them, and concludes by examining their inherent limitations.

1.2.1 Misuse Detection

Misuse-detection systems are commonly built upon open-source engines such as Snort [Roe99], Zeek (formerly *Bro*) [Pax99], and Suricata [Ope]. Each engine includes a large, mostly community-maintained signature set that serves as a default baseline. In practice, network security teams adapt and extend this baseline to fit their own network protocols, assets, and threat models. Regardless of how the rules are provisioned (whether shipped with the engine, fetched periodically from community or commercial rule repositories, or centrally pushed to multiple sensors by a management server), a rule in this category typically specifies two elements. First, it defines a matching condition, built as a conjunction of header fields (protocol, source and destination IP addresses and ports, and the direction operator) together with rule options that encode state or semantics (for example, flow keywords, per-flow memory flags such as flowbits, application-layer field checks, and payload string or signature patterns). Second, it specifies an action to execute when the condition is satisfied, such as raising an alert or dropping the packet.

1.2.2 Specification-Based Detection

Specification-based systems adopt the opposite approach to pattern matching by codifying how a protocol or program is expected to behave and raising an alarm on any illegal state transition. Sekar et al. [SBDB02] built compact finite-state automata from system-call traces and flagged executions that deviated from the defined specification. Extending the same principle to the network layer, Vigna et al. [VRKK03] introduced WebSTAT, a stateful NIDS that enforces valid HTTP request-response sequences, while Berthier et al. [BS11] applied specification-based monitoring to the strictly defined C12.22 protocol used in smart-grid Advanced Metering Infrastructures.

1.2.3 Threshold-Based Anomaly Detection

Threshold-based anomaly detection systems monitor high-level indicators (traffic volume, protocol distributions, session frequencies) and raise an alert when these metrics exceed a fixed or learned baseline. This approach enables the identification of zero-day attacks even within encrypted traffic [CBK09]. A basic implementation uses static limits (e.g., a surge in TCP SYN packets that signals a SYN flood). More adaptive schemes apply statistical smoothing, such as an exponentially weighted moving average (EWMA), to emphasize recent data while retaining historical context [YVC03, VD04]. Others apply a χ^2 test to compare the histogram of recent traffic against a benign baseline [YC01], or track the Shannon entropy of 5-tuple headers (src/dst IP, src/dst port, protocol), which drops during a focused port scan and rises when a worm randomly scans many hosts [WP05]. For multi-minute windows, principal-component analysis (PCA) [Jol02] learns a low-dimensional basis that captures the dominant correlations of benign traffic. Each new flow-level or window-level feature vector is then reconstructed from this basis, and the reconstruction error (the residual part not explained by the basis) is used as an anomaly score (unusually large residuals indicate anomalies) [RSRD07a]. Because these

detectors rely on aggregated metadata rather than payload content, they can identify emerging threats that evade conventional rules, as when a behavior-based sensor flagged the 2023 Pikabot trojan within minutes, well before Snort or Suricata signatures were published [Led24].

1.2.4 Scaling and Automating Rule-Based Detection Methods

In operational environments, misuse detection is the most widely deployed rule-based paradigm, largely due to its effectiveness against known threats [BMZ14, WJM22]. Consequently, research on scaling and automation has focused especially on two recurring challenges: the performance of signature-matching at scale and the manual effort involved in creating signatures.

To address the challenge of performance at scale, research focused on optimizing the pattern-matching stage, the step that scans reassembled byte streams against tens of thousands of signatures. Intel's open-source Hyperscan library [WHC+19] first applies fast string filters that discard traffic lacking required substrings and then verifies the remaining candidates with a compact finite-state automaton. When the authors ported Snort to Hyperscan, they observed a single-core throughput jump from 113 Mb/s to 986 Mb/s. Both Suricata and Snort 3 now support this optimization. The same approach has also been implemented in dedicated hardware: Pigasus [ZSA+20] off-loads the first-stage filters to a SmartNIC FPGA that discards roughly 95% of packets before they reach the host. The remaining stream is processed by just five CPU cores, still using Hyperscan, so a single server sustains 100 Gb/s while drawing about 38 times less power than a CPU-only design.

In contrast to systems that rely on manually written rules, GIPS [SY23] shows that signatures can instead be learned on the fly from live traffic. GIPS hashes every packet into a compact bitmap, clusters packets whose bitmaps repeatedly collide, and then mines each cluster for groups of co-occurring substrings (e.g., `root` and `admin`), rather than relying on a single frequent token that may be common in benign traffic. Once an attacker replays the same exploit a few times, GIPS automatically generates and deploys a blocking rule without analyst input. As illustrated in Figure 1.1, GIPS learns groups of co-occurring substrings rather than a single frequent token, which helps it ignore common, non-discriminative substrings (such as generic HTTP tokens) and produce more specific signatures for the exploit. In evaluations on synthetic, IoT-23 [GPE20], CIC-IDS-2017 [Sha18], and ISP traces, GIPS sustained line-rate processing with approximately 128 kB of state while achieving precision and recall values up to 0.99. Because the method relies on plaintext tokens, it is ineffective when payloads remain end-to-end encrypted.

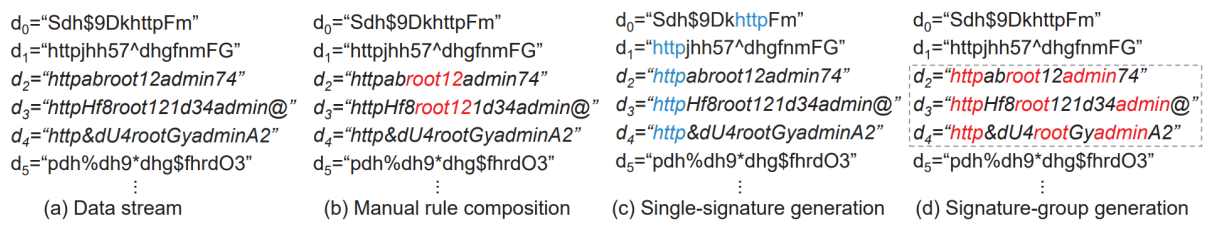


Figure 1.1: Example of GIPS signature-group generation on a packet stream. (a) Data stream that contains repeated exploit payloads. (b) Manual rule composition, where an analyst inspects payloads and chooses a signature string. (c) Automatic single-signature generation, which tends to select frequent but non-discriminative substrings such as "http". (d) GIPS learns a signature group of co-occurring substrings (for example, "root" and "admin") that more precisely characterizes the exploit. Adapted from [SY23].

1.2.5 Limitations of Rule-Based Detection Methods

Despite their maturity, rule-based NIDS still have inherent limitations. Misuse detectors match only byte sequences already encoded in their rule sets, so new or slightly altered attacks, such as attacks padded with spaces or URL-encoded variants, can evade detection. Moreover, maintaining large Snort/Suricata rule sets and keeping them tested and deduplicated still requires significant analyst effort [BM01, Bar24].

Specification-based systems rely on handwritten protocol models that quickly become outdated. This fragility arises not only because new standards emerge, but also because real-world implementations often subtly diverge from formal specifications, creating many implementation-specific protocol variants. Consequently, additions like HTTP/2, QUIC, or TLS require costly rewrites; otherwise, attacks may evade detection or benign traffic may trigger false alarms [Ko00].

Threshold-based anomaly detectors avoid payload inspection but depend on fixed or slowly updated limits: gradual low-and-slow attacks remain undetected, shifts in legitimate traffic patterns (e.g., due to a new application deployment) raise alerts, and once flows are encrypted, content-based statistics such as payload entropy or PCA-style subspace features no longer provide useful information, because ciphertext hides structure and makes benign and malicious payloads appear similarly random [YVC03].

Ultimately, these methods all share a core weakness: they require constant manual intervention and cannot adapt automatically to novel threats or changing network contexts. This deficiency motivates a data-driven alternative: ML-based detectors infer complex, multidimensional rules directly from observed traffic, detecting behavior that was never encoded by hand, retaining effectiveness when payloads are opaque to traditional signatures, and forming a foundation that can be enhanced with techniques for continuous adaptation.

1.3 ML- and DL-based Detection Methods

An ML- and DL-based NIDS is a data-driven system that converts *traffic observations* (also termed samples, instances, data points, or examples in ML/DL terminology) into threat scores using a *detection model* built (also termed *trained*) on historical data. This model architecture typically integrates two components organized into sequential stages: *feature extraction* and *decision-making*. During training, the system optimizes these components, either separately or end-to-end. This optimization enables (i) the feature extraction component to transform raw network observations into compact, structured feature vectors, which serve as numerical representations of their relevant characteristics, and (ii) the decision-making component to ingest these vectors and, in typical NIDS applications, perform one of two main tasks: *classification* or *anomaly detection*. This section first details these detection tasks and the architectural pipelines that support them, followed by a categorization of traffic observation granularity. We then examine common configuration practices found in the literature and conclude by discussing the inherent limitations of conventionally trained ML/DL models.

1.3.1 ML- and DL-based NIDS Typical Detection Tasks

Classification

Classification assigns each network observation to a specific *class* (e.g. Benign, DDoS, or Port Scan). It is appropriate when labeled training examples exist for a representative set of traffic categories (it is therefore a *supervised learning* task). This approach serves as a data-driven counterpart to rule-based misuse detection; rather than matching static, handwritten signatures,

it learns a decision boundary (or a set of boundaries) that can generalize to near-miss variants of known attacks. Its performance is commonly measured by metrics such as *accuracy* (the overall percentage of correct predictions), *precision* (the proportion of predicted attacks that were actual attacks), *recall* (the proportion of actual attacks that the system correctly identified), and *F1-score* (a single metric that balances the trade-off between precision and recall).

Anomaly Detection

Anomaly detection serves as a data-driven alternative to fixed, manually chosen thresholds on simple traffic statistics, such as connection counts, byte rates, or numbers of failed logins. It first learns a decision boundary of normal behavior from network observations, then assigns an *anomaly score* to each new observation reflecting its deviation from that baseline. Alerts are raised by comparing the score to a calibrated *threshold*. This approach is particularly useful when labeled attack samples are scarce or unavailable (it is therefore an *unsupervised* or *one-class* learning task). When ground-truth labels are available for evaluation periods, performance is reported with threshold-dependent metrics (Precision, Recall, F1) or threshold-agnostic curves (ROC-AUC, PR-AUC); otherwise, practitioners rely on operational signals (e.g., alert volume, analyst confirmations) to monitor effectiveness.

1.3.2 ML- and DL-based NIDS Pipelines

Implementing these detection tasks in practice requires a processing pipeline that defines how raw network observations are transformed into model inputs and how detection decisions are produced. The structure of this pipeline differs between traditional ML-based systems and DL-based systems.

ML Pipeline

In a conventional ML pipeline, the feature extractor is explicitly *hand-coded* by a domain expert, who performs feature engineering to produce a comprehensive, structured, largely protocol-agnostic feature vector (e.g., flow duration, total packets, and protocol type). The decision-making component in this case is usually implemented by a lightweight ML algorithm. For classification, frequently used methods are Support-Vector Machines (SVM), which separate classes with a maximum-margin hyperplane [CV95], Decision Trees (DT) that recursively partition the feature space into homogeneous leaves [Qui93], and Random Forests (RF) that aggregate many decorrelated trees to reduce variance [Bre01a]. For anomaly detection, widely adopted techniques include the one-class SVM [SPST$^+$01], Isolation Forest (IF), which isolates rare observations through random partitioning [LTZ08a], and Local Outlier Factor (LOF), which identifies samples whose neighborhood density is unusually low [BKNS00].

DL Pipeline

In a DL pipeline, manual feature engineering is typically reduced to a minimal set of coarse descriptors for each network observation. These features form the input to a learnable feature extractor that is trained jointly with the decision-making component. Through this optimization, the extractor identifies hierarchical representations and fine-grained signals of normal and malicious traffic that are challenging to capture with fixed, hand-crafted features. Since the learned component performs the majority of representation learning, the initial feature set can be significantly less detailed than in a conventional ML setup, which in turn mitigates the need for

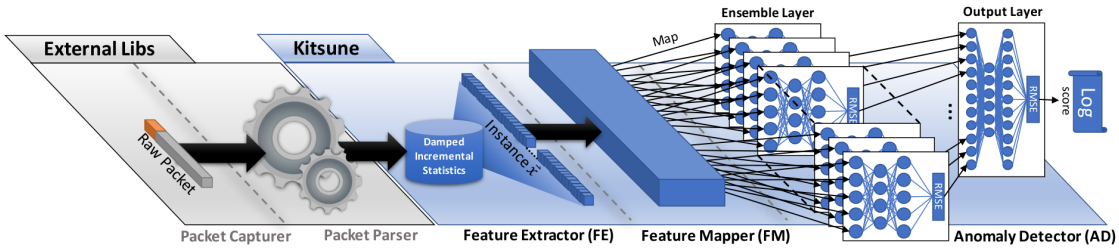


Figure 1.2: Example of a deep-learning-based NIDS pipeline (KitNET in Kitsune): an input feature vector is mapped to an ensemble of autoencoders, and reconstruction errors are aggregated into a final anomaly score. Adapted from [MDES18].

extensive, domain-specific feature engineering [MDES18]. Figure 1.2 illustrates a representative DL-based detection pipeline.

DL Pipeline Architectures

When both the feature extractor and decision-making components are learned jointly, the detection model is typically implemented as a neural network (NN). In the terminology commonly used in the ML/DL community, the feature extraction part is often called the *backbone*, while the decision-making part is called the *head*. Such networks stack multiple layers of linear transformations (with trainable parameters, often referred to as weights) and non-linear activation functions. During training, these weights are adjusted so that the backbone learns a useful feature representation and the head learns a decision rule (for example, a classifier) that maps these features to threat scores directly from data. Six canonical designs dominate NIDS research: Multi-Layer Perceptrons (MLPs) [RHW86], Autoencoders [VLBM08], Convolutional and Recurrent Neural Networks (CNNs, RNNs) [LBBH98, HS97], Graph Neural Networks (GNNs) [KW17], and Transformers [VSP+17]. While all these designs share the fundamental goal of learning a set of (sub-)optimal weights, each architecture employs a unique underlying principle to structure and share these parameters. MLPs provide a straightforward dense-layer approach, while CNNs exploit local spatial or byte-level patterns, and RNNs capture sequential traffic dynamics. Autoencoders learn to encode–decode normal behaviors, making reconstruction error an outlier indicator, whereas GNNs operate on host-level or network-wide graphs. Transformers rely on self-attention to capture long-range dependencies in network traffic data. In practice, MLPs, CNNs, RNNs, GNNs, and Transformers are usually trained to classify traffic classes, whereas Autoencoders are most often run to produce anomaly scores [MDES18].

1.3.3 Traffic Observation Granularity

A NIDS pipeline's effectiveness depends not only on the design of its feature extraction and decision-making components, but also on the *granularity* of the observations processed. In practice, network observations can be encoded at several levels: individual *packets* (e.g., byte sequences and header fields), aggregated *flows*, longer *time series* of packets or flows, or *graphs* that capture host-to-host interactions. These levels are not mutually exclusive; sophisticated NIDS may integrate multiple granularities simultaneously. Each granularity reveals certain attack signals while hiding others. For example, inspecting raw packets can find a specific exploit string, while aggregating data into flows can reveal a slow port scan that is invisible at the packet level. As a result, the choice of representation carries its own strengths, weaknesses, and the appropriate feature extraction and decision-making components.

Packet-Level Detection

At the finest granularity, each network observation is a single IP packet captured in a trace. Packet-level NIDS therefore rely on readily available header fields and, when visible, payload content. A fixed extractor may use basic IP/TCP attributes (e.g., length, time-to-live, ports, and TCP flags) and simple payload summaries such as byte histograms over the first bytes of the payload [WS04]. In the literature, representative approaches include payload-statistics anomaly detection (PAYL [WS04], POSEIDON [BZEH06], Anagram [WPS06]), supervised packet classification using hand-crafted features with classical ML models [LSM99, BM16], and DL methods that learn packet representations automatically for anomaly detection or classification. For instance, Kitsune [MDES18] applies an ensemble of autoencoders to compact, packet-derived statistics (e.g., header and simple timing features), while packet-level Transformers [CASA25] tokenize byte or field sequences and use self-attention to capture dependencies within a packet for detection. This granularity provides the earliest view of traffic, avoids flow reconstruction (making it suitable for high-speed or resource-constrained settings), and is robust to packet reordering. However, a single packet carries limited context and payload visibility is unavailable for encrypted traffic, so detection may rely on weak signals that can be obscured through padding, fragmentation, or timing manipulation.

Flow-Level Detection

At the flow granularity, the continuous packet stream is aggregated into summary records based on a shared *flow key*, typically a subset of header fields ranging from coarse (e.g., source IP) to fine-grained (e.g., the 5-tuple connection). This choice of key defines the analytical scope, allowing detection to focus on anything from broad host behaviors to specific connection-level interactions. A flow record is generated when the flow ends or reaches a configured timeout, producing a compact feature vector of statistical metrics (such as duration, volume, and directional counters) rather than raw packet content [ACT13, Can17].

In the literature, many flow-based NIDS solutions apply classical ML to hand-crafted flow statistics, sometimes enriched with contextual side information. For example, Anderson and McGrew [AM16] combine standard flow-record features with unencrypted TLS handshake and certificate fields and add related DNS and HTTP context, which helps detect malicious encrypted flows. BotMiner [GPZL08] builds on flow-level communication statistics by clustering hosts with similar connection patterns and then correlates these clusters with activity patterns to identify coordinated botnet behavior. Other works perform unsupervised detection on time windows of aggregated flow statistics (rather than on individual flows), for instance using clustering to highlight anomalous windows [MLC] or one-class models that compare consecutive windows to a learned baseline [WFSE]. More recent work learns representations directly from short packet sequences associated with a flow: LUCID uses a lightweight CNN on packet-level sequences for DDoS detection [DCMSH$^+$20], while Flow-MAE learns a compact flow embedding by masking parts of a short packet burst and training a model to reconstruct the missing parts [HLWX23].

Flow-level captures temporal and behavioral patterns that single packets do not. The trade-offs are latency, because records are exported upon flow termination or on active/idle timeouts, and reduced granularity relative to per-packet data. Short or low-rate activity can be obscured if only per-flow averages are used, and adversaries can distribute activity across many short flows to dilute per-flow indicators. These limitations are amplified with VPNs and other tunnels: unless measurement occurs at the tunnel endpoints, observed flows describe only the outer tunnel. Destination- and service-specific features disappear, and directional counters reflect an aggregate

of unrelated inner sessions rather than a single client-server exchange.

Time-Series Detection

Unlike static flow summaries that aggregate activity into single records, time-series detection analyzes network traffic as ordered sequences of timestamped observations, preserving the temporal evolution of events. This granularity can be characterized by three components: the *scope*, which specifies the entity being tracked (e.g., a connection, a host, or the entire network); the *observation unit*, which defines the signal measured at each time step (from packet metadata to flow events or aggregate counters); and the *segmentation policy*, which slices the stream into discrete units (e.g., fixed time windows, packet prefixes, or protocol state boundaries). In practice, the pipeline also adopts a consistent policy for late, missing, or irregular observations. By maintaining sequential order, this approach captures dynamic behaviors such as burst patterns, inter-arrival variations, and state transitions that are otherwise lost in static aggregations.

Time-series-based NIDS have been explored at multiple scopes and with different observation units. First, **early-decision** methods analyze short packet prefixes at the connection scope to classify traffic from only the first few packets [BTO06, CDGS07, KLL+22]. Second, at broader scopes, many approaches monitor aggregate traffic signals over **fixed or sliding time windows** to detect abrupt changes in volume or distribution that often occur with large-scale anomalies such as flooding activity [TMH11, WZS02, CHK07, Mal05]. Another line of work models network-wide traffic time series with **a low-dimensional subspace**. For example, PCA can be used to learn the main, recurring structure of normal traffic, and anomalies are then indicated by the residual variation that cannot be well explained by this subspace [LCD04, LBC+06]. Finally, more recent **learning-based** NIDSs have targeted temporal dependencies explicitly by learning representations from ordered sequences. Examples include applying DL to host-level time series [NLD+19] or to sequences of flow records [MLL+24]. Overall, these works share the same principle: keeping the temporal order exposes behaviors (bursts, periodicity, and state changes) that static summaries tend to obscure.

Time series inspection preserves packet order, enabling the detector to identify burst patterns, scans, and abnormal protocol behavior that are lost once packets are merged into flow summaries. Prefix-based models can raise an alarm after only the first T packets, whereas fixed windows of length Δt support line-rate monitoring without keeping state for each flow. These benefits have costs: prefix methods must buffer metadata for every active connection and miss attacks that begin after the first T packets, while windowing imposes at least a Δt delay, can hide brief or low-rate attacks within benign traffic, and loses direct host attribution. Both views require careful tuning of timeouts and window sizes.

Graph-Based Detection

At the graph granularity, a NIDS represents network activity as a relational structure that emphasizes *who communicates with whom* over a given time scope. Traffic is encoded as a communication graph whose nodes represent entities such as hosts, services, or users, and whose edges represent observed interactions (e.g., flows or aggregated connections) between these entities. Both nodes and edges can carry feature vectors that summarize behavior (e.g., traffic volumes, protocol and port information, or connection persistence), and the resulting graph can be analyzed either as a single snapshot for a fixed time window or as a time-evolving sequence of graphs. This representation enables the NIDS to exploit both graph structure (connectivity patterns, neighborhoods, communities) and temporal changes (appearance or disappearance of

links and roles) that are difficult to capture with isolated packet, flow, or windowed statistics.

In the literature, graph-based NIDS exploit communication structure in several ways. Early approaches derived hand-crafted graph features from per-window snapshots; for example, BotTrack [FWSE11] builds host communication graphs and uses centrality signals to identify clustered bots. Other approaches learn embeddings on evolving graphs, such as NetWalk [YCA+18], which flags nodes or links whose dynamic representations deviate from normal patterns. More recently, GNN-based methods have been used to learn representations directly from graph data, including NetVigil [HWS+24], which scores anomalous edges, and ARGANIDS [VFMC23], which utilizes graph autoencoders.

Graph-level inspection reveals structural anomalies (unusual communication pairs, dense botnet clusters, or rare connectivity patterns) that finer-grained views often miss. However, compared to packet-, flow-, or time-series-based pipelines, graph-based approaches are typically more resource-intensive and temporally coarser, because they require explicit graph construction and storage at the level of nodes and edges [WZG+25, NMH+10, KSJ+23]. In practice, scaling issues often emerge in memory and runtime when graph size grows with the number of active entities and interactions [WZG+25]. Detection quality also depends on carefully tuned parameters (window width, clustering radius), and seed-based schemes succeed only when at least one known malicious node appears in each malicious cluster.

Other Granularities

Beyond packet-, flow-, time-series-, and graph-based representations, some security methods operate at alternative or hybrid granularities. *AMIDES* [UHH+24] addresses the critical problem of rule evasion, where attackers make minor changes to commands to bypass static SIEM signatures. Its novel approach is to learn a similarity model by training directly on an organization's SIEM ruleset as the "malicious" class and a history of non-alerting events as the "benign" class. In operation, AMIDES flags events that do not trigger a specific rule but are determined by the model to be highly similar to one, providing context by suggesting which rules were likely evaded. Evaluated on a large enterprise dataset with over 500 handcrafted evasions, AMIDES successfully detected 70% of them with zero false positives. Qu et al. [QML+23] propose a multi-granularity hierarchical classifier, evaluated across five tasks, including intrusion detection, which processes traffic in three stages: it vectorizes packet metadata (sizes, timing), uses an encoder to aggregate packets into flow vectors, and then uses another encoder to aggregate flow vectors into a single trace vector (i.e., a final, single numerical summary that represents the entire traffic event). By retaining fine packet timing, within-flow dynamics, and cross-flow context that single-granularity models discard, the approach reduces information loss and improves classification on encrypted, variable-length traffic.

1.3.4 Configuration of ML/DL-based NIDS

Core Concepts

In addition to what a learning-based detector learns from historical traffic data during training, ML/DL-based NIDS depend on a set of design choices that are fixed explicitly by the practitioner. We refer to these choices as the system's *configuration*. In this context, configuration mainly includes hyperparameters that control the learning process and the model setup. For example, an RF classifier requires setting the number of trees and their maximum depth, while an IF depends on the number of trees and the subsampling size used to build each tree. For DL models such as an LSTM-based detector, configuration includes architectural choices (e.g., number of

layers and hidden size) and training settings (e.g., learning rate and batch size). Unlike the model's learned weights, these settings are not inferred automatically from traffic data and must be selected before training or updating.

Configuration can be performed manually or automatically. Manual configuration selects hyperparameters using domain heuristics, values reused from prior implementations, default settings, and trial-and-error adjustments. Automatic configuration relies on hyperparameter optimization (HPO) procedures that explore candidate settings according to a defined search rule. Common approaches include exhaustive *grid search* and *random search* [BB12], which evaluate candidate configurations from a predefined set or sampled ranges. More sample-efficient methods include *Bayesian optimization*, which uses a surrogate model to guide the search toward promising regions of the configuration space [SLA12]. Other families of automated strategies include *evolutionary algorithms* for hyperparameter search [RAHL19], and *population-based training*, which combines parallel training with periodic exploitation of strong models and exploration through hyperparameter perturbations [JDO+17].

Configuration in Practice for Conventionally Trained ML/DL NIDS

Learning-based NIDS rely on an offline configuration step to set their hyperparameters. In this section, we examine how configuration is handled in contemporary ML/DL-based NIDS research. For this analysis, we reviewed 37 peer-reviewed papers published between 2015 and 2025 across 11 top-tier security venues, including *USENIX Security*, *IEEE Symposium on Security and Privacy (S&P)*, *ACM Conference on Computer and Communications Security (CCS)*, *Network and Distributed System Security Symposium (NDSS)*, *IEEE EuroS&P*, *RAID*, *ESORICS*, *AsiaCCS*, *SAC*, *IEEE INFOCOM*, and *ACM KDD*. Of these studies, 32 relied on heuristic or manual parameter selection, while 5 used automatic methods (all five using grid search). The full bibliography appears in Appendix A.

Collectively, the corpus spans a broad spectrum of use cases. Roughly half of the papers (19/37) analyze enterprise or campus-backbone traffic; twelve focus on industrial or in-vehicle control networks (*e.g.*, CANtropy [SC24]), and the remaining six address IoT edge deployments such as ARGUS [RSW+23]. With respect to technique, one-third (13) still adopt classical statistical or rule-based detectors (*e.g.*, Haetae [NP15]), whereas deep-learning pipelines dominate the rest (24 papers), from autoencoder ensembles (Kitsune [MDES18]) to masked-graph models (MAGIC [JWGZ24]). A notable subgroup of nine studies employs system- or data-plane provenance graphs for high-fidelity attack reconstruction (*e.g.*, FLASH [uRZRX24] and KAIROS [CLL+24]). Despite this technical diversity, the overwhelming reliance on hand-tuned parameters persists, underscoring a field in which rigorous, scalable configuration search is still the exception rather than the rule.

A closer inspection of the 32 manually configured studies shows that 14 undertake a limited sensitivity analysis of their detectors. Typically, the authors sweep a narrow range for a single hyperparameter, such as the anomaly-score threshold or time-window length, and plot the resulting change in F_1, ROC-AUC, or false-positive rate. This per-parameter sweep reassures the reader that the chosen default is not arbitrarily fragile. Yet in almost every case the experiment varies only one hyperparameter at a time while holding all others fixed; interactions between parameters remain unexplored, and no attempt is made to search the joint space of settings. As a result, these studies demonstrate local robustness but stop short of establishing sound configuration across the joint hyperparameter space.

Notably, none of the papers surveyed relies on Bayesian optimization, evolutionary algorithms, or population-based training; grid search remains the sole automated technique we ob-

served. We attribute this gap to several interrelated factors. First, grid search is trivial to implement and deterministic, making results easy for reviewers to reproduce without specialized software tooling. Second, most NIDS evaluations still run on modest datasets or limited-scale testbeds, so the additional computational cost of more advanced optimizers yields little marginal benefit over an exhaustive search of a few discrete values. Third, the configuration logic in published NIDS codebases is often tightly coupled with the main application, lacking the modularity required to integrate external optimization frameworks like `Optuna` or `RayTune`. Finally, the community's emphasis on *explainability and operational simplicity* favors techniques whose search trajectories can be enumerated, a property that grid search offers by construction.

1.3.5 Limitations of Conventionally Trained ML/DL Models

While the conventionally trained ML/DL NIDS surveyed in the previous section represent a step toward greater adaptability by moving beyond rigid signature-based detection, their reliance on offline training limits their effectiveness in dynamic settings. Trained once on a fixed dataset and then deployed statically, these systems are poorly suited to evolving network environments. Their effectiveness relies on a fragile assumption: that the data distribution at deployment will match the one seen during training. This assumption is consistently violated in practice due to *concept drift*, driven by factors such as protocol evolution, new applications, and attacker adaptation [GŽB+14, LLD+19, SHG+15].

This challenge affects both main learning paradigms. Supervised classifiers are particularly vulnerable, as they depend on a constant supply of timely labels and are inherently biased toward detecting only known attack classes [SP10b]. Similarly, anomaly detectors, while not requiring attack labels, are sensitive to changes in the normal behavior baseline, often raising false alarms in response to benign shifts or failing to detect gradual, "low-and-slow" attacks whose patterns evolve slowly over time [RSRD07b, LCD04].

In short, while these NIDS solutions are a necessary foundation, they are not enough for building a truly adaptable NIDS. To overcome these limitations, they must be coupled with explicit adaptation mechanisms designed to handle the challenges of dynamic, real-world network traffic.

1.4 Adaptation Methods for ML/DL-Based Detection Methods

To address the limitations of conventionally trained ML/DL-based NIDS, the literature proposes a range of explicit adaptation mechanisms designed to handle the challenges of dynamic network traffic. This section surveys the six major families of these methods:

- **Drift Detection:** Methods in this family act as an early-warning mechanism that monitors incoming traffic for distributional changes (i.e., concept drift) over time.
- **Online and Continual Learning Methods:** These methods enable the NIDS to learn incrementally from a continuous stream of new samples, adapting its parameters to temporal changes without requiring a complete and costly retraining.
- **Active and Semi-Supervised Label Acquisition:** These methods address the critical bottleneck of scarce and costly expert labels in operational environments. Instead of relying on extensive supervision, they employ strategies that maximize the information extracted from the available mix of limited human annotations and abundant unlabeled network traffic.

- **Transfer Learning and Domain Adaptation:** These approaches aim to improve adaptation by reusing prior experience from a source domain or task (for example, an NIDS trained on HTTP traffic from one enterprise network). They adapt the model to a new target environment so that learned features and decision rules are effectively transferred rather than learned from scratch.
- **Meta-Learning Frameworks:** These methods explicitly learn an adaptation strategy. For instance, one strategy is to learn an optimal parameter initialization for the NIDS model, which allows it to adapt quickly to new network environments or attack types with minimal data.
- **Threshold and Score Calibration:** This family contains post-hoc mechanisms that adjust the decision boundary or the interpretation of the model outputs without changing the internal weights.

1.4.1 Drift Detection Mechanisms

Drift-detection mechanisms act as the first line of defense in a dynamic traffic landscape. Their primary function is to raise an alert indicating that the network context has changed (e.g., due to new applications, protocols, or attack types), signaling the need for refinement of the detection model, such as retraining or threshold resets. Existing methods in the literature for drift detection in NIDS can be grouped along three main design axes: the monitored signal (data representation, model outputs, or prediction confidence), the change measure (distance-based or statistical tests), and the deployment location (host-based systems or in-network data-plane logic), as summarized in Table 1.1. Concretely, each row describes a single drift-detection method by answering the three questions that define the axes above: *what* signal is monitored, *how* changes are measured, and *where* the detection logic is deployed.

Table 1.1: Categorization of Drift-Detection Mechanisms

Method (Reference)	**Monitored Component (What)**	**Measurement Technique (How)**	**Deployment Location (Where)**
CADE [YXCG21]	Data representation (latent flow embedding space)	Euclidean Distance & MAD Test	Host-based
OWAD [HWL23]	Model outputs (calibrated normal-confidence scores)	Permutation Test & Kullback–Leibler Divergence	Host-based
TRANSCEND [JSD+17]	Prediction confidence/credibility (conformal p-values)	Thresholding on Learned p-value Statistics	Host-based
Fast Learning Enabled by In-Network Drift Detection [XMT+24]	Streaming traffic statistics computed in the P4 pipeline.	Statistical Test (EWMA)	Network Data Plane

- *CADE* [YXCG21] addresses the challenge of detecting drift in high-dimensional data by first learning an optimized representation of traffic. It employs a contrastive autoencoder trained with existing labels to map data into a low-dimensional latent space, encouraging same-class samples to cluster and different classes to separate. For each new flow, CADE computes its Euclidean distance to every class centroid and applies a per-class Median Absolute Deviation (MAD) threshold; a flow that exceeds the threshold for all classes is flagged as drift.

- *OWAD* [HWL23] monitors the model's output distribution, focusing on benign traffic, using the workflow shown in Figure 1.3. The drift detection process itself involves two primary stages. First (Step 1), it calibrates the model's raw outputs into probability-like normal-confidence scores, making them suitable for distributional testing (so that score shifts reflect real changes rather than arbitrary scaling or overconfidence). Second (Step 2), it builds frequency histograms of these calibrated scores for a control window (Before Shift (Old)) and a treatment window (After Shift (New)). A non-parametric permutation test using Kullback–Leibler divergence as the test statistic is then applied to these histograms to formally detect a normality shift. While the full OWAD framework also includes subsequent steps for (Step 3) explaining the detected shift and (Step 4) adapting the model, these functions are complementary to the core drift detection mechanism.

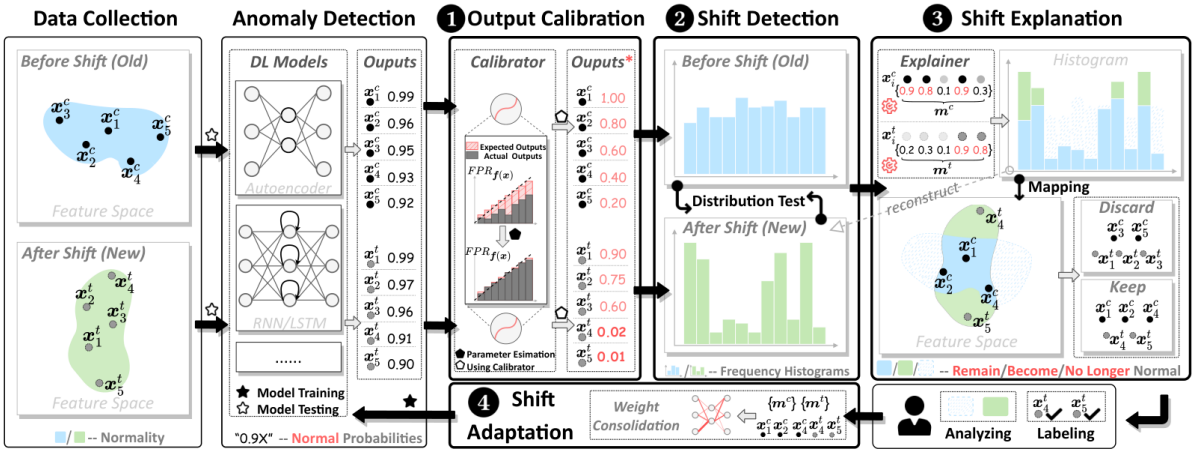


Figure 1.3: Overview of the OWAD workflow, illustrating the four stages of drift detection and model refinement: (1) Calibration, (2) Shift detection, (3) Explanation, and (4) Adaptation. Adapted from [HWL23]

- *TRANSCEND* [JSD+17] tracks the model's confidence in its own predictions. It implements this by wrapping any classifier in a conformal-evaluation layer. Instead of using the classifier's raw probability output, which can be misleadingly high for new, out-of-distribution data, this layer computes statistical p-values. These p-values serve as a statistical measure of normality, representing the proportion of training samples that were at least as dissimilar (or non-conforming) as the new sample. TRANSCEND then learns class-specific thresholds for these p-values from the training set to satisfy a user-chosen performance goal. Any prediction whose p-value falls below these learned thresholds is flagged as untrustworthy, thereby exposing concept drift without relying on fresh labels.

- *Fast Learning Enabled by In-Network Drift Detection* [XMT+24] emphasizes *where* drift is detected by embedding the drift detector in the switch data plane. Concretely, it implements an in-network exponentially weighted moving-average (EWMA) change detector over streaming traffic features computed in the P4 pipeline; when the EWMA statistic crosses a predefined control limit, the BMv2 switch raises an alarm. Upon an alarm, the control plane collects updated traffic and retrains the downstream NIDS model. In the evaluated prototype, this model is a decision-tree classifier, chosen for its fast retraining and lightweight inference, and is then redeployed.

1.4.2 Online and Continual-Learning Methods

Online and continual-learning methods provide an ML-based NIDS with the ability to revise its own parameters whenever new traffic patterns emerge. Unlike drift detectors, which raise an alarm that the distribution has shifted, these techniques update the feature extractor, the decision-making component, or both, thereby realigning the decision boundary with the current data distribution. In an *online* setting, the model adapts after every flow or mini-batch, whereas in a *continual* framework it learns sequential tasks over time, assimilating new knowledge while retaining what it learned earlier.

Online Learning Approaches

Online methods achieve continuous adaptation primarily through autonomous retraining on pseudo-labeled incoming data or by dynamically managing an ensemble of learners.

- *Autonomous Retraining.* AOC–IDS [ZCW24] targets online deployment where traffic and attacks evolve, and the IDS must update itself with minimal human input. It first learns flow representations with an autoencoder trained using a contrastive objective that pulls normal samples together and pushes abnormal ones away. At inference time, each new representation is compared to a reference normal profile (computed from an initial clean normal set) using a similarity score. The system then fits the similarity scores with two Gaussian distributions, interpreted as "normal" and "abnormal", and uses the resulting posterior probabilities as pseudo-labels. These pseudo-labels are accumulated over time and periodically used to update the detection module and refresh the Gaussian parameters. The rationale is to avoid manual threshold tuning and to enable continual adaptation by reusing the IDS's own high-confidence decisions. A second example, *Online Self-Supervised Intrusion Detection* (SSID) [LNL24] follows an online setting where no offline dataset or manual labeling is assumed. It maintains a trust coefficient that reflects whether the NIDS is currently reliable enough to learn from its own outputs. This trust combines (i) a representativeness factor that measures how different current traffic is from previously learned traffic (via normalized KL divergence on simple packet properties such as inter-transmission time and packet length), and (ii) a generalization factor that captures whether the model has enough effective training evidence and has recently achieved low error. When this trust falls below a threshold, SSID collects a small batch of packets and updates the NIDS by treating a packet as malicious or benign with a probability that is proportionally scaled by the model's score and the current trust level, and otherwise ignoring it. This mechanism reduces confirmation bias by learning less when trust is low, and it limits the risk of learning from attack-contaminated data by discarding the batch whenever an attack is detected during collection.

- *Ensemble-Based Adaptation.* ENIDrift [Wan22] uses an adaptive ensemble workflow, illustrated in Figure 1.4. It is based on a dual process to balance high-speed detection and drift adaptation. For real-time detection, every raw packet is embedded by the lightweight iP2V encoder (Part 1). This vector is immediately fed to the 'Ensemble detection' module (Part 2), where a pool of unsupervised sub-classifiers (e.g., AEs or PCAs) produces a fast, performance-weighted prediction (weights reflect recent accuracy on labeled chunks). For drift adaptation, this vector is also copied to the 'Learning new distribution' loop (Part 3), which operates in the background. Here, traffic is collected into chunks. After labels are released by an oracle, the system calculates a 'generation index' (G-idx) (Part 3(1))

that combines prediction variance and error to assess stability and accuracy, reducing the risk of training on unstable or attack-contaminated chunks. If the data chunk satisfies this index, the system trains a new sub-classifier. This new model is then integrated into the ensemble, and the weights of all existing sub-classifiers are dynamically adjusted based on their recent accuracy, effectively muting obsolete models while strengthening accurate ones.

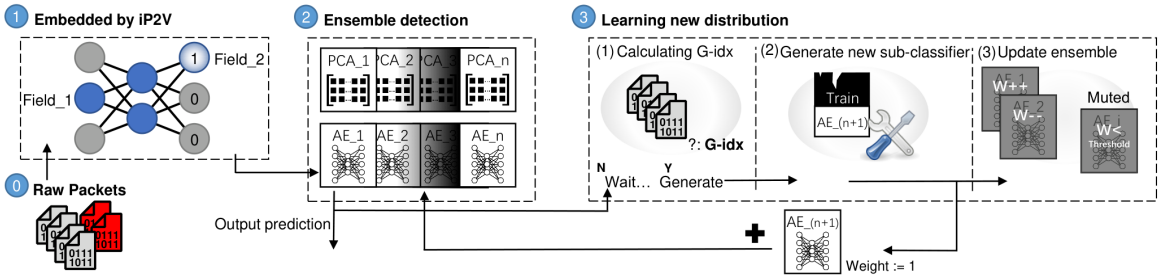


Figure 1.4: Overview of the ENIDrift workflow, illustrating its three primary stages: (1) iP2V embedding, (2) Ensemble detection, and (3) Learning new distribution. Adapted from [Wan22].

Continual Learning Frameworks

Continual learning frameworks face the dual challenge of assimilating new knowledge while avoiding catastrophic forgetting of previously learned patterns.

- *Replay and Memory Management.* A common way to limit catastrophic forgetting is to keep a small replay buffer of past samples and to decide carefully what to store and what to overwrite. The Extended Class-Balancing Reservoir Sampling (ECBRS) policy in [RP23] focuses on the practical issue of severe class imbalance. It maintains simple per-class counts and derives a target buffer quota γ_c for each class c, so that rare attack classes are guaranteed a minimum presence in memory. When the buffer is full and a new sample arrives from a class that is still below its quota, ECBRS preferentially overwrites a sample from a globally frequent and currently over-represented class (typically benign), while ensuring no class falls below γ_c. When the incoming class has already reached its quota, ECBRS switches to a standard class-wise reservoir rule: it replaces an existing sample of the same class only with a probability that decreases as more samples of that class have been observed. The rationale is to prevent frequent traffic from monopolizing the buffer, while also avoiding excessive replacement within any single class. Strategic Selection and Forgetting (SSF) [KL25] addresses a different problem: the buffer can become outdated when the traffic distribution drifts. It first uses a Kolmogorov–Smirnov test to check whether recent traffic differs from what the buffer represents. If drift is detected, SSF ranks stored samples by how representative they remain for the current distribution and ranks new samples by how well they describe what has changed. It then keeps representative samples and discards those that are no longer useful, so that limited memory is spent on current, informative traffic. The rationale is to keep replay aligned with the present operating conditions, while still coping with class imbalance.

- *Representation Learning and Regularization.* Beyond simple replay, other approaches combine memory with advanced representation learning or gradient-based methods. ReCDA [XLW24] adapts a feature extractor and the classifier of an NIDS in two stages, as illustrated in

Figure 1.5. First, the Representation Enhancement Stage learns robust features without needing new labels. It employs Drift-aware Perturbation to create augmented samples by mixing features from historical (D^h) and recent (D^r) flows, simulating the drift process. These perturbed flows and their originals are fed into a shared Feature Extractor (encoder). Through Representation Alignment, a contrastive loss ($\mathcal{L}_{ra}$) encourages the encoder to produce similar representations for original and perturbed pairs, thus learning features that are both aware of the drift and invariant to it. Second, the Classifier Tuning Stage injects classification knowledge. An Instructive Sampling module selects a small, informative subset of the already-labeled historical flows (D^h). These selected samples are then used to fine-tune the Feature Extractor and train the Classifier using a standard cross-entropy loss ($\mathcal{L}_{ce}$) combined with a consistency regularization term ($\mathcal{L}_{cr}$). This regularization ensures the feature extractor retains the robust properties learned in the first stage, preventing catastrophic forgetting and adapting the classifier without requiring any new annotations. This two-stage routine can be triggered by an external detector or run periodically to maintain performance against concept drift.

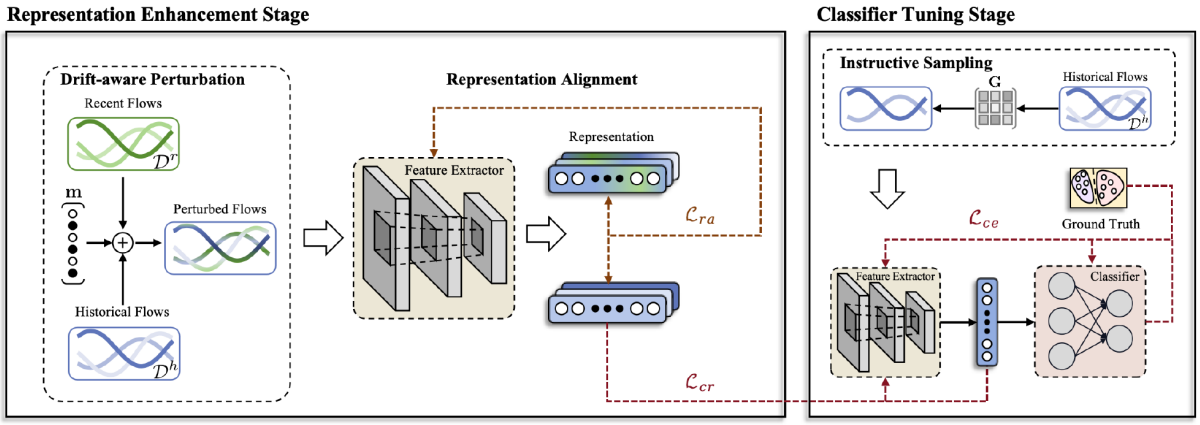


Figure 1.5: Overview of ReCDA. **Stage 1 — Representation Enhancement:** apply *Drift-aware Perturbation* (feature-wise mixing of a historical flow D^h with a recent flow D^r) and *Representation Alignment* (contrastive loss) to learn features that are drift-aware yet invariant. **Stage 2 — Classifier Tuning:** use *Instructive Sampling* to select a small labeled subset from D^h, and update the encoder and classifier with cross-entropy plus a *Representation-Consistency Regularizer*; no new annotations are required. Adapted from [XLW24].

- *Gradient-Based Optimization.* SPIDER [LPG24] enables continual learning for NIDS using only unlabeled data. It enhances privacy by storing only unlabeled samples from the most recent training stage in a buffer and relies on Gradient Projection Memory (GPM) to retain knowledge from earlier stages. GPM stores a small set of basis vectors (directions in the NN's activation space) that capture the essential features learned at each prior stage. When learning a new stage, the model computes the gradient of its current training loss (the direction its parameters would move to reduce error). SPIDER then filters this update, removing components that would interfere with those stored feature directions; in effect, it adjusts the model only along directions not represented in the GPM memory. This allows learning on the new stage while minimizing interference with previous knowledge.

- *Open-world novelty detection.* Beyond adapting to drift among known patterns, some continual-learning NIDS also address the open-world setting, where entirely new attack types may appear over time and must be flagged as novel rather than matched to known

classes. For instance, CND-IDS [MR25] continually trains an autoencoder-based feature extractor using unlabeled traffic and a small clean subset of normal flows. After each continual training stage, it fits PCA on the encoded normal features and scores each new sample by its PCA reconstruction error; the rationale is that PCA captures the low-dimensional structure of normal behavior, so normal samples reconstruct well, whereas novel attacks tend to deviate from this structure and therefore produce larger reconstruction errors.

Summary

Collectively, these methods illustrate key strategies for building an adaptive NIDS. The design choices range from fully automated self-training (AOC-IDS, SSID) to continual-learning approaches that require little or no analyst input (SPIDER). A crucial aspect is memory management, with techniques spanning class-balancing reservoirs (ECBRS) to drift-aware selection and forgetting (SSF). Some continual-learning systems also address the open-world setting by flagging previously unseen attacks as novel (CND-IDS).

1.4.3 Active and Semi-Supervised Label Acquisition

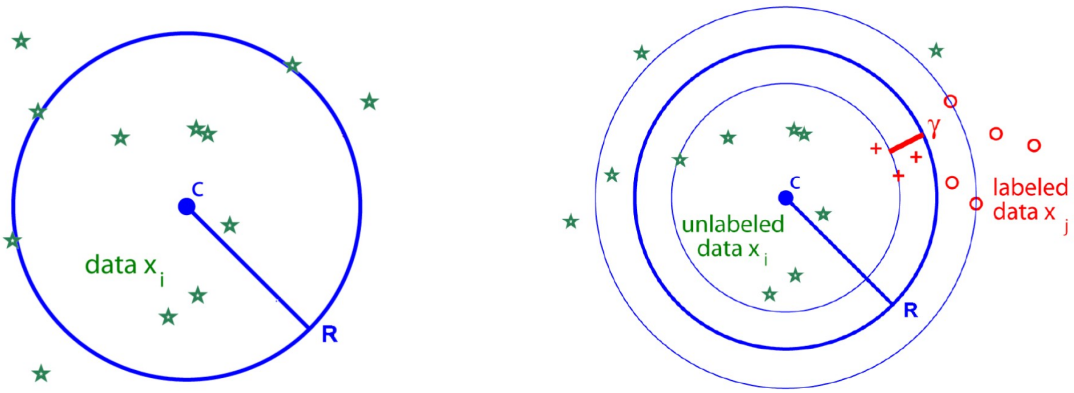
Active and semi-supervised label acquisition methods address the challenge of scarce and costly annotations by selecting which samples to label. These approaches can be broadly categorized based on the source of the new labels. Active Learning (AL) focuses on strategically querying a human analyst for labels of the most informative samples to maximize model improvement. In contrast, Semi-Supervised Learning (SSL) aims to generate high-confidence pseudo-labels for new, unlabeled data, thereby minimizing or eliminating direct human involvement.

Active Learning Query and Budget Strategies

In active learning, the core challenge is to define a query strategy and budget policy that maximize performance gain per annotated sample.

- *Uncertainty and Geometry-Based Sampling.* The most established techniques are based on uncertainty sampling, where the model queries the samples about which it is least confident. An early example, Active–SVDD [GKRB09], integrates AL with one-class Support Vector Data Description (SVDD). A standard SVDD fits a minimal hypersphere that encloses the normal data, flagging points outside as anomalies (Figure 1.6(a)). Active–SVDD goes beyond querying only near-boundary points by combining two criteria. The first is a margin score (λ_1) that measures proximity to the SVDD boundary (closer means more uncertain). The second is a k-NN neighborhood score (λ_2), computed on a k-nearest-neighbor graph to favor candidates that are surrounded by similar unlabeled samples but not close to the already-labeled benign region. The final query score is a convex combination $\tau\lambda_1+(1-\tau)\lambda_2$, so selected points are both uncertain and representative of new regions, which reshapes the boundary once labels are incorporated (Figure 1.6(b)).

- *Drift-Aware Representative Sampling.* Under benign distribution shift, Mateen [AM24] avoids exhaustive manual labeling by triggering annotation only when drift is detected, then asking an analyst to label only a small representative coreset. The framework first uses a Kolmogorov–Smirnov test on the DAE's reconstruction errors to detect when the benign distribution has drifted. Upon detection, it builds a small, representative coreset for manual labeling by first partitioning the data by reconstruction error. It then computes



(a) SVDD baseline. Fit a minimal hypersphere (center c, radius R) to normal data; points outside are anomalies.

(b) Integrated update. Incorporate analyst labels (benign in red, attacks in blue) together with unlabeled data (green) to jointly refit c, R, and a margin γ, reshaping the boundary.

Figure 1.6: **Active–SVDD overview.** Active querying scores unlabeled points with a convex combination of a margin score (distance to the SVDD boundary) and a k-NN neighborhood score that favors points surrounded by similar unlabeled samples but not close to the labeled benign region; the final query score is $\tau\lambda_1 + (1-\tau)\lambda_2$. Adapted from Görnitz [GKRB09].

two scores for each sample: Informativeness ($\hat{I}$) quantifies how many similar samples a point represents (based on the distance of their concatenated reconstruction and feature error vectors), while Uniqueness ($\hat{U}$) is computed as the cumulative distance of a sample's feature error vector to all others in its partition. The samples with the highest combined weighted score ($\lambda_0\hat{U} + \lambda_1\hat{I}$) are selected.

- *Policy and Budget Optimization.* More advanced frameworks optimize the query strategy itself. Deep-Q Active Learning [WHW24] frames query selection as a reinforcement-learning task, training a controller to learn an optimal sampling policy. In contrast, Human-in-the-Loop IDS [KDZ24] introduces an explicit cost model, with policies designed to shorten detection time while respecting a fixed analyst budget.

Semi-Supervised and Hybrid Mechanisms

To further reduce human workload, semi-supervised methods generate automatic pseudo-labels for unlabeled traffic, while hybrid approaches combine limited manual labels with additional mechanisms such as augmentation.

- *Semi-supervised pseudo-labeling.* INSOMNIA [PASR21] employs a dual-model, co-training-style design where a primary DNN classifier is paired with an independent Nearest-Centroid (NC) oracle. When network traffic drifts, the system identifies the DNN's least-confident predictions. Instead of querying a human, these uncertain samples are pseudo-labeled as either benign or attack by the NC oracle, which assigns the label of the closest class centroid learned from previously labeled data. The DNN is then fine-tuned on this pseudo-labeled set, enabling adaptation to new traffic patterns without direct analyst intervention. The rationale is that the NC oracle provides a simple, independent reference that can guide updates on hard samples and reduces the risk of the DNN reinforcing its own errors during self-training under drift.

- *Hybrid active learning with augmentation.* NetGuard [GLZ25] addresses concept drift together with class imbalance by combining budgeted active learning with synthetic data augmentation. When performance degrades on a new unlabeled batch, it fits Gaussian Mixture Models (GMMs) to the labeled source data and the unlabeled target batch, and ranks target samples using an informativeness score based on their relative likelihood under the two GMMs. The top-ranked samples are then sent to a human annotator (or oracle) within a fixed budget for labeling. To mitigate imbalance, NetGuard augments minority attack classes by generating additional synthetic samples using tabular generative models (TVAE, a VAE-based generator, and RealTabFormer, a Transformer-based generator), and retrains the detector on the expanded dataset.

Summary

Overall, label-efficient adaptation methods reduce the reliance on scarce human annotations by either selecting a small set of high-value samples for manual labeling (active learning) or by extracting supervision from unlabeled traffic through pseudo-labels and related mechanisms (semi-supervised learning). Hybrid designs combine a limited labeling budget with augmentation to maintain robustness under drift and class imbalance while keeping analyst effort manageable.

1.4.4 Transfer Learning and Domain Adaptation

Transfer learning and domain adaptation techniques enable a NIDS trained on one deployment environment to operate effectively on another without requiring a brand-new, fully labeled dataset. These methods align the feature space of the source domain (the original training environment) with that of the target domain (the new deployment environment).

Label-Free Diagnosis and Calibration

Before performing any form of adaptation, it is often necessary to verify that the deployment data remain compatible with the training distribution, using, for example, label-free bias diagnostics and calibration techniques.

- *Label-Free Bias Diagnosis.* Thirumuruganathan *et al.* [TDK+24] address performance decay from sampling bias, which is the distributional mismatch between labeled training data (D_T) and unlabeled deployment data (D_U). They propose two complementary methods to detect this bias. First, domain discrimination trains a lightweight classifier to predict whether a sample comes from D_T or D_U; if the classifier achieves high accuracy, the two datasets are easy to separate, which indicates likely sampling bias. Second, they propose a k-nearest-neighbor (k-NN) distance test (Fig. 1.7). An encoder is trained on the training data using Supervised Contrastive Learning (SupCon) to produce embeddings, which are then normalized. The core intuition is to check whether deployment embeddings fall into the same local neighborhoods as training embeddings. If they do not, deployment points tend to be farther from their nearest neighbors in the training set than training points are from one another, leading to a systematic shift between the two distance distributions and thus flagging sampling bias. The authors introduce both detectors because they provide complementary trade-offs: domain discrimination is simple and fast, while the k-NN test is non-parametric and sensitive to geometric mismatch in the embedding space.

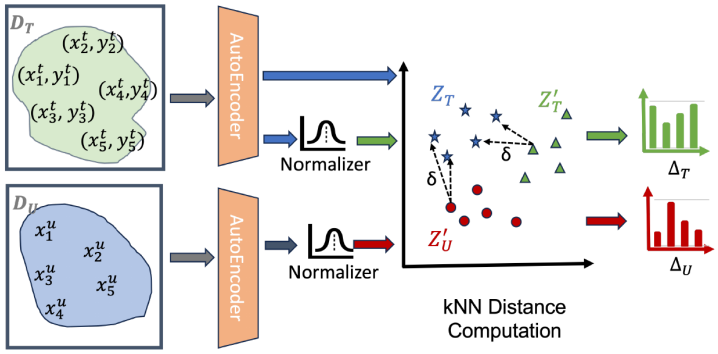


Figure 1.7: Overview of the k-NN distance diagnostic pipeline for detecting sampling bias between labeled training data (D_T) and unlabeled deployment data (D_U). Adapted from [TDK+24].

- *Output Loss Normalization.* While the previous method diagnoses distribution shift in representation space, Yehezkel *et al.* [YES21] instead create invariance in the reconstruction-loss space to transfer anomaly detection across domains. They use a split architecture with (i) one local autoencoder per domain, trained on local traffic assumed to be mostly benign, and (ii) a shared Global Detector. Each local autoencoder outputs, for every input window, a *loss vector* (feature-wise reconstruction errors). Because loss magnitudes vary across networks (different volumes and patterns), the authors normalize these loss vectors locally (e.g., Min-Max or norm scaling) so that losses from different domains become comparable. The Global Detector is then a supervised decision model trained on normalized loss vectors collected from multiple domains, and its role is to detect whether the current window is normal or anomalous (threat). In other words, the local autoencoder produces the anomaly signal (loss), and the Global Detector consumes the normalized signal to output the final anomaly decision, enabling one shared detector to operate across heterogeneous deployments without retraining a separate classifier per domain.

Few-Shot Adaptation

When a small number of labeled samples from the target domain are available, more sophisticated adaptation techniques become possible. These often aim to align the feature representations of the source and target domains.

- *Adversarial Alignment.* Singla *et al.* [SVB20] formulate domain adaptation for NIDS in which labeled data are available in a source domain and only limited labeled data are available in a target domain. Their GAN-based framework (Fig. 1.8) learns a representation that supports intrusion classification while reducing domain-specific cues, so that a detector trained with source labels generalizes better to the target setting. Concretely, the framework jointly trains (i) a generator (G) that projects source and target records into a shared, domain-invariant latent space and (ii) two competing heads fed by this latent space: a classifier to distinguish attack vs. benign and a discriminator (D) to distinguish source vs. target. By alternately minimizing the standard classification loss and a domain-confusion loss, the generator learns features that are simultaneously attack-discriminative and domain-invariant; the trained G is then used directly as the detector. Because the same mechanism operates even when the two datasets expose different feature schemas, the method naturally covers heterogeneous migrations.

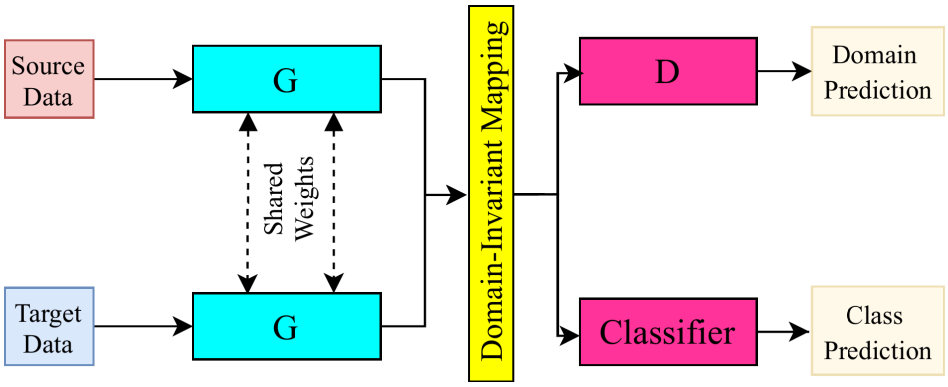


Figure 1.8: Architecture of the adversarial domain adaptation framework for NIDS, illustrating the joint training of the shared generator (G), domain discriminator (D), and intrusion classifier. Adapted from [SVB20].

- *Explicit Feature Mapping.* Most heterogeneous domain adaptation methods align source and target data in a latent space, which often improves transfer but makes features hard to interpret because latent dimensions do not have a clear meaning. Yu *et al.* [YLJ+25] propose NAEF to keep the transfer process in an explicit and human-readable feature space. Their key idea is to build one augmented feature vector that contains three parts: (i) features shared by both domains, (ii) features that exist only in the source domain, and (iii) features that exist only in the target domain. Because a source sample does not contain target-specific feature values (and vice versa), NAEF learns two small nonlinear mapping models that predict domain-specific features from the shared ones, one for each domain. It then uses these mappings to fill in the missing part, so that both source and target samples are represented in the same augmented explicit space where each dimension keeps its original, named meaning (e.g., flow duration, packet size statistics). Finally, NAEF trains an ensemble of Transformer-Encoder classifiers on these unified explicit vectors; the attention mechanism helps identify which explicit features contribute most to the intrusion decision, improving interpretability.

Unsupervised Domain Adaptation

The most advanced techniques aim for true zero-shot transfer, adapting to a new domain with no target labels. In practice, unsupervised domain adaptation for NIDS is often implemented through one of three strategies: (i) self-training with pseudo-labels, often combined with contrastive objectives, (ii) adversarial domain alignment (possibly with multiple source domains), or (iii) large-scale self-supervised pre-training that learns transferable features before any labels are used.

- *Contrastive Self-Training for Bias Mitigation.* When sampling bias is detected, Thirumuruganathan *et al.* [TDK+24] also propose mitigation algorithms that learn a more domain-invariant representation. The core intuition is that the original feature space is poorly aligned: samples from the labeled source domain and the unlabeled target domain may form different regions even when they represent the same underlying behavior, which harms generalization. The goal is therefore to learn a new latent space that respects two standard assumptions: (i) a *continuity* assumption, where nearby samples should share the same label, and (ii) a *cluster* assumption, where class boundaries should lie in low-density gaps between clusters rather than cutting through dense groups. The proposed mitigation

strategy uses contrastive learning to build such a space by pulling together samples that are likely to share the same class (e.g., benign) across both datasets and pushing them away from samples likely to belong to the other class (e.g., malicious), thereby improving transfer under sampling bias.

- *Multi-Source Adversarial Learning.* Another common unsupervised strategy is adversarial domain alignment, which trains an encoder to produce representations that remain predictive for intrusion classification on labeled source data while making unlabeled target samples hard to distinguish from source samples in the latent space. Layeghy *et al.* [LLP23] introduce *DI-NIDS*, which combines a multi-source Domain-Adversarial Neural Network (DANN) with a one-class SVM (OSVM). The DANN encoder is optimized with two simultaneous objectives: (i) predict attack vs. benign using labeled source data and (ii) confuse a domain classifier trained on representations from both the source and the unlabeled target domain. The resulting domain-invariant feature extractor is then coupled with an OCSVM, enabling detection in the new network without requiring any target labels.

- *Self-Supervised Pre-training.* Moving beyond adversarial alignment, Golchin *et al.* [KSK25] explore self-supervised contrastive pre-training. They discard labels during pre-training and teach a packet-sequence Transformer to discriminate between augmented views of the same flow (token replacement, order shuffling) and unrelated flows. The contrastive loss forces the encoder to capture protocol syntax and timing cues that persist across networks.

1.4.5 Meta-Learning Frameworks

Meta-learning frameworks go beyond aligning specific domains. Instead, they learn an adaptation strategy so that a NIDS can be updated with only a small amount of data when deployed in a new network context. During meta-training, the model is trained over many short *episodes*, where each episode corresponds to a different network context (e.g., a site, time period, or attack setting). By repeating training across diverse episodes, meta-learning optimizes the model so that, at deployment time, it can adapt rapidly to an unseen context with only a few update steps and limited supervision.

Metric-based Meta-Learning

One major family of meta-learning, known as metric-based learning, aims to learn an embedding space in which classification can be performed by comparing samples with a similarity measure. The "metric" can be (i) a *learned* comparison function implemented by a neural network (outputting a similarity or difference score), or (ii) a *fixed* distance such as Euclidean distance applied in an embedding space that is itself meta-trained to make this distance meaningful. In both cases, the key point is that the embedding and/or the comparison rule is trained across many few-shot episodes so that it transfers to new contexts with little target supervision.

- *Pairwise Comparison.* FC-Net [XSD20] learns a **data-driven similarity function** for comparing two traffic samples. The architecture has two parts: a feature extractor (F-Net) and a comparison network (C-Net). Given two flows x and x', F-Net embeds them into feature maps, and C-Net outputs a single delta score that acts as the method's metric: a small delta means the two flows are similar, and a large delta means they are different. This delta score is therefore the distance function used by the method, and it is learned directly from data rather than chosen a priori. During episodic meta-training, the model

repeatedly solves few-shot tasks by comparing each unseen example to a small labeled set of reference examples from each class. At inference time, a new flow is classified by computing its average delta score to the reference examples of each class and assigning the label with the smallest average delta (Fig. 1.9).

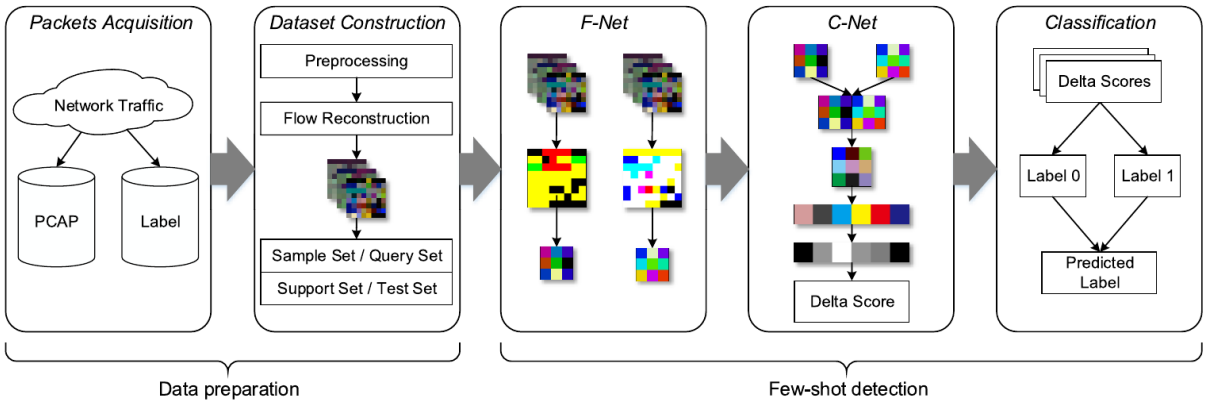


Figure 1.9: Overview of the FC-Net few-shot detection pipeline, illustrating the pairwise comparison process using F-Net for feature extraction and C-Net for similarity scoring. Adapted from [XSD20].

- *Prototype Learning.* PTN-IDS [NMW24] follows a *prototype-based* metric approach, meaning that each class is represented by a single reference point in a learned embedding space. Concretely, after a feature extractor maps traffic samples to embeddings, the method computes one prototype per class as the average embedding of a small labeled set of examples from that class. A new flow is then classified by embedding it and assigning the label of the closest class prototype, typically using Euclidean distance. The meta-learning happens during training: the model is trained over many small tasks constructed from different subsets of data, and after each task the extractor is updated to make samples cluster tightly around their correct class prototype and far from other class prototypes. Repeating this process across many tasks yields an embedding space in which these class averages remain reliable, allowing the method to generalize to new settings with only a few labeled examples. At deployment in a new target environment, the same feature extractor is reused to embed a small labeled reference set from each class. Class prototypes are computed as the average embeddings of these reference examples, and each new flow is classified by the nearest prototype, without any additional gradient-based fine-tuning (Fig. 1.10).

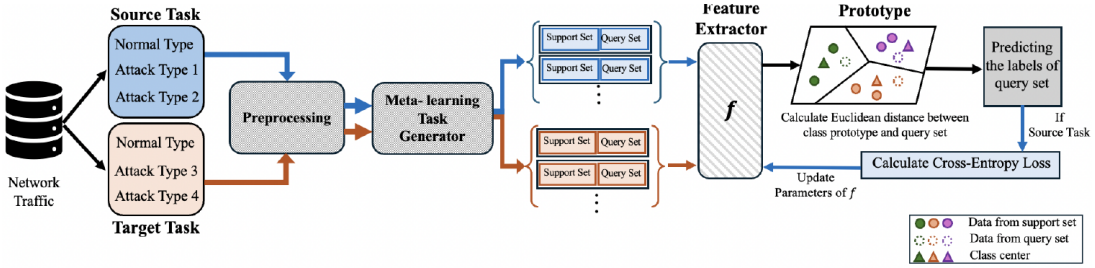


Figure 1.10: Overview of the PTN-IDS prototype-based few-shot detection pipeline, illustrating the meta-training process and prototype computation for deployment. Adapted from [NMW24].

Optimization-based and Hybrid Approaches

A second family, known as optimization-based meta-learning, instead learns a model initialization that can rapidly adapt to a new task with only a few gradient steps.

- *Model Initialization.* RETSINA [LWL+23] employs this strategy for unsupervised zero-day web attack detection. The key intuition is to first train a "universal initial model" on data from many different (auxiliary) web domains; this model is not trained to be good at any single domain, but rather to be highly adaptable to new, unseen domains. When a new target domain is introduced, RETSINA adapts this universal model using only a small amount of unlabeled requests from that domain (in experiments, about five minutes of traffic sufficed). This adaptation is achieved by fine-tuning the model to minimize the reconstruction error on the target domain's unlabeled data, requiring no attack labels.
- *Hybrid Self-Supervision.* To push label efficiency further, *FS3* [DAM23] first pre-trains an encoder with self-supervised masking on unlabeled traffic. This pre-training teaches the encoder generic traffic patterns from raw features, so the later contrastive fine-tuning can adapt the detector with only 5–10 labeled flows per class.

Summary

Taken together, these four studies illustrate a clear progression of meta-adaptation. They begin with metric-based few-shot learners such as FC-Net, which can approach fully supervised performance with few labeled flows per class. FS3 then introduces self-supervision to further reduce labeling needs and to better handle severe class imbalance. PTN-IDS next adopts a prototype-based formulation, enabling low-overhead inference and straightforward adaptation to novel classes. Finally, RETSINA targets the most constrained setting by providing a label-free bootstrap procedure for deployments where no annotations can be collected at deployment time.

1.4.6 Threshold and Score Calibration

Threshold- and score-calibration procedures adjust the decision boundary of an intrusion detector, rather than its learned weights, so that the stream of anomaly scores remains consistent with real-world risk and operational limits. By rescaling scores or repositioning the cut-off, they translate model output into a trade-off between false positives and false negatives. Because calibration acts only on the model's outputs, it can be applied instantly, often without any new labels or full retraining. The result is an NIDS that continues to alert at a manageable rate and maintains coverage long after its last training cycle.

Statistical and Analytical Approaches

The most direct methods treat threshold setting as a statistical problem, deriving it from the properties of the score distribution.

- *Probability-Based Thresholds.* Bridges *et al.* [BJR17] lay the foundation by giving an explicit, non-parametric formula linking a p-value threshold β and the expected alert rate. With N scored events, the expected number of alerts is at most βN; thus choosing $\beta = R/N$ targets, in expectation, at most R alerts. Equality holds when the p-values are continuous and properly calibrated (i.e., when the score model matches the data). For example, with about 4,645 scores per minute and a budget of one alert per minute, setting $\beta \approx 2.15 \times 10^{-4}$

yielded roughly 0.64 alerts per minute in their 337-minute stream. The result applies to rule-, ML-, and ensemble-based detectors. Therefore, a sustained deviation between the observed alert rate and the target budget serves as a robust indicator of model miscalibration or data drift, signaling a need to retrain or recalibrate the model before the budget can be trusted again.

- *Dynamic Statistical Thresholds.* Building on that analytic baseline, Chae *et al.* [CKD19] introduce a dynamic scheme with two thresholds, MaxTh and MinTh, recomputed per time window from the empirical distribution of a behavior variable (in their study, a connection count such as host- or port-degree). MaxTh is set using an outlier rule designed for heavy-tailed data (i.e., distributions where extreme-but-legitimate outliers, such as busy servers, are common) to identify the statistically rare, low-probability tail. MinTh is simply the mean of all values below MaxTh, serving as a cut-off for clearly normal behavior. Observations falling between MinTh and MaxTh are labeled suspicious and are not immediately flagged. Instead, they are judged by a reputation system that accumulates a trust score for each host over time. Behaviors are scored based on these thresholds (e.g., below MinTh is positive trust, above MaxTh is negative), and a Predictability-Trust variant uses a sliding window to focus on recent evidence, which specifically mitigates On-Off attacks by preventing attackers from quickly rebuilding trust during quiet periods.

- *Error-Based Thresholds.* Hundman *et al.* [HCL+18] apply a similar statistical thresholding philosophy, but they derive their score from the prediction error of an LSTM model, proposing a general, unsupervised, non-parametric thresholding method. The intuition is to find a threshold that is not predefined but is discovered from the statistical structure of the recent history of smoothed prediction errors for that stream (computed online from past errors, without labels). The method first smooths the prediction error series (e.g., using an EWMA) to remove small, insignificant spikes that are just noise. It then performs an unsupervised search for an optimal threshold by testing multiple candidates (e.g., 2.5, 3.0, 3.5. . . standard deviations above the mean). The best threshold is chosen by optimizing a score that balances two factors: (1) the benefit of removing the outliers (measured by the percentage drop in the remaining data's mean and standard deviation) and (2) a penalty for the number of anomalous values and sequences this would create. This penalty prevents the algorithm from being overly greedy and flagging too many minor events. Finally, a critical pruning step is applied to reduce false positives, as illustrated in Figure 1.11. After the main threshold is set, the system checks all resulting anomalous sequences. Any sequence that is not substantially larger than the next-most-anomalous sequence is re-classified as normal. For example, in Figure 1.11, *Anomaly 1* (0.01072) is pruned because its magnitude is too close to the next peak (0.00994), suggesting it is part of the data's normal noise. Conversely, *Anomaly 2* (0.01396) is kept because it is significantly larger than *Anomaly 1*.

Game-Theoretic and Cost-Based Approaches

Moving beyond pure statistics, some work incorporates economic and adversarial factors by framing threshold selection as a game.

Ghafouri *et al.* [GAL+16] reframe the problem as a defender–attacker game. While their model is for Cyber-Physical Systems (CPS), where damage is physical and time-varying (e.g., a water-network attack at peak vs. low demand), their formal, game-theoretic approach is broadly adaptable. They model the defender's loss as the sum of false-positive costs and the time-dependent damage an undetected attack can cause. They then derive a polynomial-time

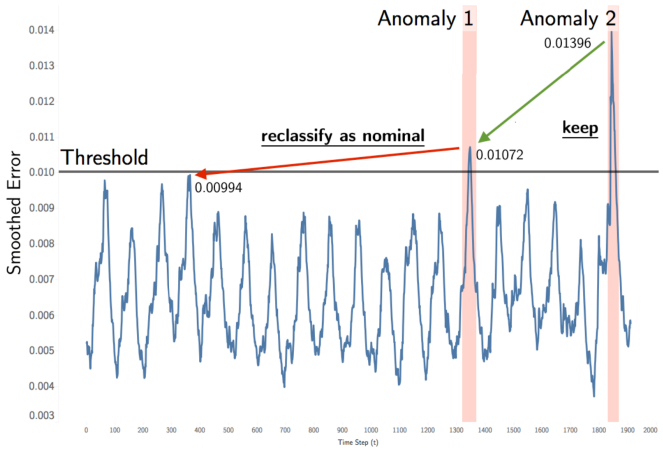


Figure 1.11: Overview of the non-parametric dynamic thresholding and pruning process. The plot illustrates how candidate anomalous segments of the smoothed prediction error are evaluated and filtered based on their relative peak magnitudes. Adapted from [HCL+18].

algorithm that computes an adaptive threshold policy, which strategically lowers the threshold (increasing sensitivity) during high-risk periods and raises it during low-risk times. This adaptive policy was shown to lower the defender's total loss compared to the best possible fixed threshold.

Reinforcement Learning Approaches

Some methods replace analytical formulas with learning agents that discover an optimal thresholding policy online.

Heartfield *et al.* [HLBP21] deploy MAGPIE, a non-stationary multi-armed bandit that, every five minutes, picks the contamination parameter of Isolation Forests using the silhouette score as an unsupervised reward. Yang *et al.* [YHS23] generalize this idea with ADT: a deep-Q network monitors the detector's recent outcomes, including how often it raises correct alerts, how often it triggers false alarms on normal data, and how often it misses true anomalies. Based on these running counts, the agent switches between a passive mode with a higher decision threshold (τ) to reduce false alarms, and an active mode with a lower τ to increase sensitivity. This adaptive control drives the F_1 score to about 0.95 on YAHOO[10] and up to 0.999 on SWAT[11]/WADI[12].

1.4.7 Synthesis of Adaptation Methods for ML/DL-based NIDS

As synthesized in Table 1.2, the landscape of adaptation techniques for ML-based NIDS is not monolithic. Instead, each one addresses a different facet of the challenge posed by dynamic network environments. These methods can be understood as a series of design choices, revealing a clear trade-off between the complexity of a solution, its label requirements, and the specific problem it solves.

At one end of the spectrum are reactive strategies like Drift Detection and Threshold Calibration. These act as a first line of defense, monitoring a deployed model to either provide early

[10] **Yahoo**: the Yahoo Webscope S5 time-series benchmark for real-world anomaly detection.

[11] **SWaT**: Secure Water Treatment, an operational water-treatment ICS testbed dataset widely used for cyber-physical intrusion detection.

[12] **WADI**: Water Distribution, an ICS testbed dataset capturing network and process anomalies in a water-distribution system.

warnings of change or adjust the decision boundary to manage alert volume, by primarily acting on the model's output scores rather than retraining its internal weights. In the middle are data-centric approaches that optimize the learning process itself: Active and Semi-Supervised Learning makes the use of scarce human expertise more efficient, while Transfer and Meta-Learning enable rapid model deployment to entirely new networks with minimal data. At the other end of the spectrum, Online and Continual Learning methods directly update the model's weights, allowing it to track evolving traffic patterns and learn new threats over time.

The key takeaway is that building an adaptable NIDS is not a matter of selecting the single best algorithm, but of composing a strategy from these methods. The table serves as a decision-making framework, allowing a practitioner to identify the right method based on the specific operational challenge (be it managing alert volume, adapting to evolving threats, or deploying to a new site) and the available budget for labels and computation.

Table 1.2: Synthesis of adaptation families for ML/DL-based network intrusion-detection systems.

Family	**Problem Solved / Use Case**	**Core Mechanism**	**Label Budget**	**Key References**
Drift Detection	Provides early warning of concept drift without costly retraining.	Flags statistical shifts in data or model outputs; model weights are static.	None	CADE'21, OWAD'23, TRANSCEND'17, Xavier *et al.*'24
Online & Continual Learning	Maintains accuracy as traffic and threats evolve over long periods.	Incrementally updates model weights; uses replay or regularization to prevent forgetting.	Zero to Low (Pseudo-labels)	AOC-IDS'24, ENIDrift'22, ReCDA'24, SPIDER'24, CND-IDS'25
Active & Semi-Supervised	Minimizes labeling effort to respect limited analyst time and budget.	Intelligently queries an oracle for high-value labels; uses pseudo-labels to fill gaps.	Minimal (Analyst-bounded)	Active-SVDD'09, Deep-Q AL'24, HITL-IDS'24, GAA'25, METALS'24
Transfer Learning & Domain Adaptation	Deploys a pre-trained model to a new network with a different data distribution.	Aligns source and target feature spaces to reuse existing knowledge.	Zero to Few (Target labels)	Singla *et al.*'20, NAEF'25, DI-NIDS'23, Portnoy *et al.*'21
Meta-Learning Frameworks	Enables ultra-fast adaptation to new environments or threats with very few examples.	Learns an optimal initializer or metric space for rapid few-shot fine-tuning.	Few to Zero	FC-Net'20, FS3'23, PTN-IDS'24, RETSINA'23
Threshold & Score Calibration	Keeps alert volume manageable without retraining the underlying model.	Adjusts the decision boundary or rescales anomaly scores.	None to Very Low	Bridges'17, Ghafouri'16, Hundman'18, MAGPIE'21, ADT'23

1.5 Conclusion

This chapter reviewed the main stages of NIDSs, from rule-based engines to ML/DL-based NIDS, and then to learning-based NIDS augmented with explicit adaptation mechanisms. These adaptation methods were introduced to address practical deployment gaps, such as non-stationary traffic, evolving attacks, limited labels, and domain shifts. Across the surveyed families, adaptation can target different components of the pipeline, including, for example, the training data and the model parameters. The synthesis in Section 1.4.7 highlighted that there is no single adaptation strategy that fits all contexts.

A key observation of this chapter is that learning-based NIDS are shaped not only by what they learn from historical traffic, but also by how they are configured. The review of configuration practices in conventionally trained ML/DL NIDS showed that this step is still handled mostly manually in the literature, with automated procedures remaining uncommon and often limited to simple offline grid search. In parallel, although adaptation methods can update weights, request labels, or recalibrate the decision threshold during deployment, they generally preserve a fixed configuration once the system is deployed. Threshold calibration is a notable exception, since many works adapt the final cut-off online. Beyond this case, the remaining hyperparameters and architectural choices are typically selected offline and then frozen.

This leads to a central gap: current adaptive NIDS rarely reconsider the configuration that governs their own updates. In other words, the literature largely adapts the model, but not the settings that control how adaptation is carried out. For example, an online learner may keep the same learning rate, replay-buffer size, and regularization strength long after traffic volume, noise, or class balance has changed. Similarly, an active learning system may fix its query budget and pseudo-label confidence threshold even when analyst availability or the uncertainty profile of the traffic evolves. As a result, each adaptation family remains vulnerable to gradual performance decay that existing adaptation layers may not correct.

At the same time, adjacent machine learning fields have demonstrated that automatic hyperparameter tuning can be done during training, using approaches such as gradient-based hyperparameter optimization, population-based training, or warm-started optimization services. This suggests that configuration adaptation is technically feasible, but it has not yet been integrated into the NIDS adaptation loop in a systematic way. In the next chapter, we address this missing link by proposing a meta-learning framework for adaptive configuration, aiming to retune key hyperparameters online in response to changing network contexts.

Chapter 2

A Meta-Learning Framework for Automated NIDS Configuration

Contents

2.1 Introduction 41
2.2 NIDS Configuration Problem Formulation 42
2.2.1 ML/DL NIDS Lifecycle and Notation 42
2.2.2 Hyperparameter Optimization Methods for NIDS 46
2.2.3 Limitations of Conventional Hyperparameter Search for NIDS . . . 48
2.3 Meta-learning-based Auto-configuration 48
2.3.1 Threat Model 48
2.3.2 Overview 49
2.3.3 Offline Phase: Meta-modeling and Objective 50
2.3.4 Online Phase: On-the-fly Configuration 53
2.4 Conclusion 53

2.1 Introduction

Conventional hyper-parameter optimization (HPO) addresses configuration as a single, statis, offline search. While effective for an initial deployment, this static approach becomes a bottleneck under context change: re-tuning from scratch is expensive.

This chapter introduces a *meta-learning framework for automated NIDS configuration*. The core principle of our approach is to reframe the problem: instead of treating HPO as a repetitive search to be solved independently for each new network context, we treat it as a supervised prediction task that learns from past experience. By leveraging knowledge from historical, offline tuning tasks, our framework can rapidly infer an effective configuration for a new, unseen network context in a single shot.

This paradigm offers two significant advantages. First, it drastically reduces the time and computational resources required for reconfiguration. Second, it resolves a fundamental limitation of HPO for anomaly detectors by enabling the joint inference of both hyperparameters and the decision threshold using only benign traffic. This chapter will first provide a formal mathematical definition of the NIDS learning lifecycle and the configuration problem. It will then

detail the proposed meta-learning framework, describing its offline meta-modeling phase and its lightweight online inference phase.

2.2 NIDS Configuration Problem Formulation

This section formalizes the ML/DL-based NIDS configuration problem. Moving from the high-level overview in Section 1.3, it starts by presenting the formal mathematical framework that defines the NIDS learning process, specifying the end-to-end NIDS lifecycle, and clarifying where configuration intervenes in the construction of the detection model. Building on this, the section provides a mathematical formalization of HPO, defining it as an optimization problem where the goal is to find the best-performing configuration within a vast search space that minimizes a given validation metric. It then provides a more detailed technical description of the primary configuration search procedures introduced in the previous chapter.

2.2.1 ML/DL NIDS Lifecycle and Notation

End-to-end NIDS Lifecycle

The common lifecycle of a detection model unfolds in four sequential stages: the *configuration-training*, *validation*, and *evaluation* carried out in the pre-deployment space, followed by *deployment* in the deployment space (production network at runtime).

The process begins with the configuration and training stage. Here, the detection model is first configured by setting its hyperparameters. It then ingests a *training set* of past observations (e.g., packets, flows, host graphs), which may be fully annotated with legitimate vs. malicious labels, only labeled as legitimate (no malicious labels are available), or partially annotated. Using this data, the model refines its internal parameters, which embed the detection logic, to minimize a training goal, such as reducing classification error.

Next, the validation stage assesses the detection model on a separate *validation set* to determine how effectively the initial configuration performs in practice. Any necessary adjustments are made iteratively in a feedback loop with the configuration-training stage. Once the detection model is finalized, the evaluation stage determines how well it handles completely unseen traffic using an independent *test set*.

Finally, once evaluated, the trained system is deployed to continuously monitor new observations and detect intrusions in real time. Figure 2.1 illustrates an example of this entire workflow. In the pre-deployment space, three annotated sets (training, validation, and evaluation) are used to configure and train a Random Forest [Bre01b] (RF) model, validate its performance (with iterative feedback to refine the configuration), and finally conduct a last evaluation before deployment. The rest of this section introduces these steps in detail.

Observation space and dataset representations

Let $\mathcal{X}$ denote the space of all possible network observations. A single observation $\mathbf{x} \in \mathcal{X}$ may take one of several forms depending on the chosen granularity: (i) the raw bytes of a single packet, (ii) a flow vector aggregating byte and packet counters between two endpoints, (iii) a short time-series $\{\mathbf{p}_t\}_{t=1}^{T}$ from the first T packets of a connection, or (iv) a communication graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ that records host interactions. In practice, large volumes of such observations are collected into a *dataset* $\mathcal{D}$, which may focus on only one format (e.g., flows) or include a mixture of forms (depending on the study or collection process).

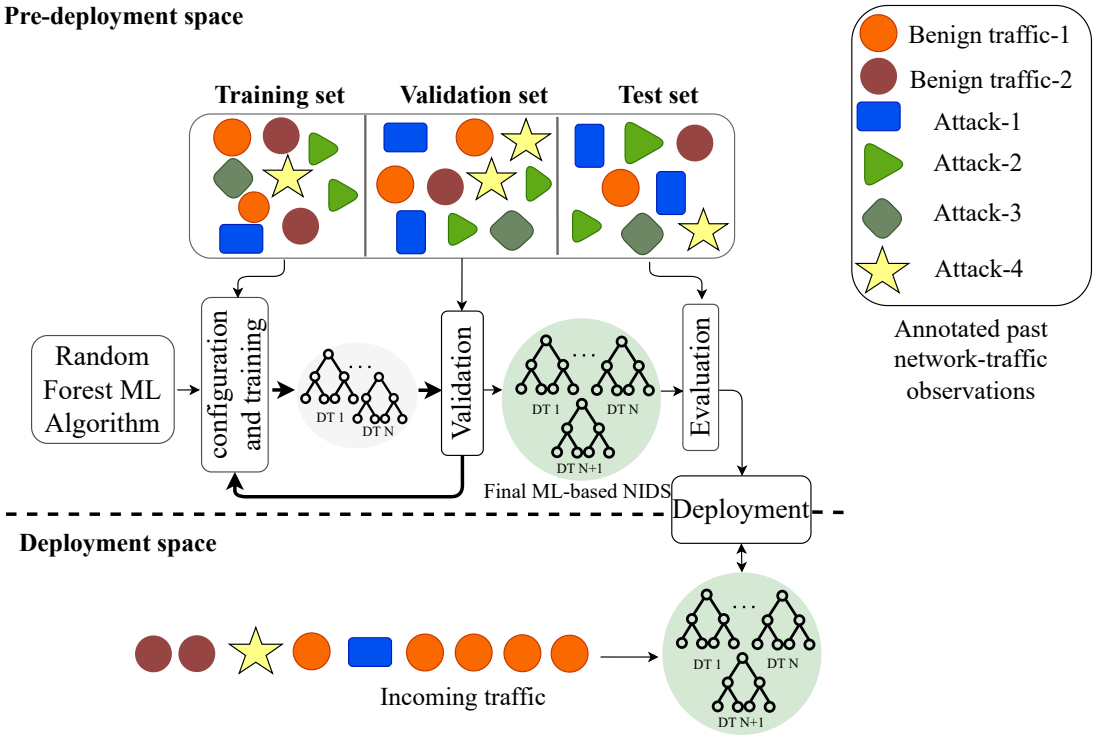


Figure 2.1: Illustration of the NIDS workflow from configuration and training through validation and final evaluation, culminating in deployment.

Labeling Schemes and Learning Paradigms

The nature of the labels within this dataset determines the learning paradigm. If each $\mathbf{x}_i$ is paired with a label $y_i \in \mathcal{Y}$, where $\mathcal{Y}$ is the set of possible classes, then $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}$, and the setup is *supervised*, with classification as the typical task. This can be binary classification, which distinguishes between two classes (e.g., Benign vs. Malicious); multi-class classification, which distinguishes among several classes (e.g., Benign, DDoS, Port Scan); or multi-label classification, where an observation may belong to more than one class at the same time (e.g., tagged as both Botnet and Spam). If, however, only legitimate traffic is labeled, we have an *unsupervised* (or one-class) approach, where $\mathcal{D} = \{\mathbf{x}_i\}$ (all supposed to be benign) [13], and the most common task is anomaly detection. Partial annotation naturally allows a *semi-supervised* approach, where a fraction of both legitimate and malicious observations are labeled, and the rest remain unlabeled.

Dataset splitting and validation protocols

After typical preprocessing (e.g., removing corrupted records, normalizing timestamps), $\mathcal{D}$ is split into $\mathcal{D}_{\text{train}}, \mathcal{D}_{\text{val}}$, and $\mathcal{D}_{\text{test}}$ for training, validation, and evaluation stages. In practice, this partitioning is often done randomly or via stratification to preserve label distribution, with common splits like 70/15/15 or 80/10/10, ensuring no overlap between subsets. Alternatively, k-fold cross-validation can be applied: first divide $\mathcal{D}$ into k equal-sized, non-overlapping folds (typical choices are $k = 5$ or $k = 10$); then repeat k rounds in which, at round i, the model

[13] In general machine-learning terminology, "unsupervised" implies *no* ground-truth labels whatsoever. However, in the NIDS domain, unsupervised is frequently interpreted as having *only legitimate* data labeled, with no malicious labels available [LGH+23].

is trained on the union of all folds except fold i and validated on fold i; finally average the k validation scores (and often report their standard deviation) to produce a performance estimate that is less sensitive to the choice of a single split.

Feature Extractor and Decision Function

These partitioned datasets are the foundation for the detection model's construction. This model can be defined as:

$$h^{(\boldsymbol{\lambda})} = g_{\boldsymbol{\theta}^{(\boldsymbol{\lambda})}} \circ f_{\boldsymbol{\phi}^{(\boldsymbol{\lambda})}} : \mathcal{X} \to \mathcal{Y}. \tag{2.1}$$

where the function:

$$f_{\boldsymbol{\phi}^{(\boldsymbol{\lambda})}} : \mathcal{X} \longrightarrow \mathbb{R}^d \tag{2.2}$$

is the feature extractor function, parameterized by $\boldsymbol{\phi}$, that maps each observation $\mathbf{x}$ to a fixed-length vector $\mathbf{z} \in \mathbb{R}^d$. This step converts the training set $\mathcal{D}_{\text{train}}$ into a uniform representation suitable for standard learning algorithms, while capturing the most relevant and discriminative characteristics from the raw, potentially high-dimensional observations. The function:

$$g_{\boldsymbol{\theta}^{(\boldsymbol{\lambda})}} : \mathbb{R}^d \longrightarrow \mathcal{Y} \tag{2.3}$$

is the decision function, parameterized by $\boldsymbol{\theta}$, that processes $\mathbf{z}$ and outputs the detection model's prediction $\hat{y}$. Depending on the task, $\hat{y}$ may be a discrete label (e.g., via a multiclass softmax with $\arg\max$) or a real-valued score $s^{(\boldsymbol{\lambda})}(\mathbf{x}) \in \mathbb{R}$ (e.g., a probability or anomaly score). In the score-based case, a discrete alert is issued by applying a pre-calibrated threshold τ:

$$\hat{y} = \mathbb{1}\big[s^{(\boldsymbol{\lambda})}(\mathbf{x}) \geq \tau\big]. \tag{2.4}$$

Hyperparameters and threshold conventions

Both f and g depend on the hyperparameter vector $\boldsymbol{\lambda}$, which specifies the design choices for the overall model h. Unlike the model parameters $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$, this vector is selected from a search space Λ beforehand and is not learned during the training process. For score-based detectors, the configuration defined by $\boldsymbol{\lambda}$ must also account the decision threshold τ. For this, two conventions are common. One treats τ as part of the configuration by defining an augmented hyperparameter vector $\boldsymbol{\lambda}' = (\boldsymbol{\lambda}, \tau)$ and searching jointly over the space $\Lambda' = \Lambda \times \mathcal{T}$, where $\mathcal{T} \subseteq \mathbb{R}$ is the admissible threshold set (e.g., $[0, 1]$ for probabilistic outputs, or a finite grid over the relevant score range). The other decouples τ from the main search: for each configuration $\boldsymbol{\lambda} \in \Lambda$, a separate calibration step selects a threshold $\tau(\boldsymbol{\lambda}) \in \mathcal{T}$ according to the chosen operating objective. Which convention to use depends on whether the operating point should be chosen within the search or kept adjustable after training for different operating targets.

Bi-level optimization view of configuration and training

During the configuration–training and validation stages, the construction of h is inherently a two-level optimization problem. At the outer level, a configuration $\boldsymbol{\lambda} \in \Lambda$ is selected. With this choice fixed, the inner level tunes h's parameters $\boldsymbol{\Theta} = (\boldsymbol{\phi}, \boldsymbol{\theta})$ on $\mathcal{D}_{\text{train}}$ (although in classical pipelines $\boldsymbol{\phi}$ remain fixed and only $\boldsymbol{\theta}$ is optimized).

Inner loop: training objective, loss choices, and regularization

This tuning is done by minimizing a training goal:

$$\boldsymbol{\Theta}^*(\boldsymbol{\lambda}) \;=\; \arg\min_{\boldsymbol{\Theta}} \Big[\tfrac{1}{|\mathcal{D}_{\text{train}}|} \sum_{(\mathbf{x},\, y)\in\mathcal{D}_{\text{train}}} \ell\Big(g_{\boldsymbol{\theta}}\big(f_{\boldsymbol{\phi}}(\mathbf{x})\big),\, y\Big) \;+\; \mathcal{R}\big(\boldsymbol{\Theta};\, \boldsymbol{\lambda}\big) \Big], \tag{2.5}$$

where ℓ is a loss function that measures the discrepancy between the final detection model's output $\hat{y}$ and the target y. ℓ is chosen to be compatible with both the model's output space $\hat{y}$ and the structure of the labels y.

For a classification task, ℓ might compute cross-entropy using a known label y. In that case, the detection model outputs a probability distribution over $\mathcal{Y}$, and the cross-entropy loss measures how closely those predicted probabilities align with the true label. In practice, NIDS classification is typically imbalanced (far more benign than malicious samples), so one may use *weighted cross-entropy* or *focal loss* to emphasize rare malicious classes and mitigate bias toward the majority benign class.

For an anomaly-detection task, however, the model is typically trained only on legitimate traffic, so the loss ℓ does not use a label y. Instead, it might measure how well the model reconstructs the input data, or it might quantify how far an observation lies from a learned boundary that encloses the training distribution (e.g., as in a One-Class SVM).

In either case, as shown in Equation 2.5, the training goal is to find the parameters $\boldsymbol{\Theta}$ that minimize the average loss plus a regularization penalty $\mathcal{R}$. This penalty, governed by $\boldsymbol{\lambda}$ (or $\boldsymbol{\lambda}'$ for score-based detectors, depending on the chosen convention), ensures the model controls its complexity, thereby mitigating overfitting and improving its ability to generalize to new traffic.

Outer loop: validation selection and calibration

Once the model has been trained for a given $\boldsymbol{\lambda}$ (or $\boldsymbol{\lambda}'$), the resulting detector is assessed on $\mathcal{D}_{\text{val}}$ using the same loss ℓ. This *select* $\boldsymbol{\lambda}$ *(or* $\boldsymbol{\lambda}'$*)* $\rightarrow$ *train* $\rightarrow$ *validate* cycle is repeated (either concurrently when hyperparameter values are manually enumerated in a static grid or random list, or sequentially when new candidates are generated automatically by an adaptive search algorithm) until the search budget is exhausted. $\boldsymbol{\lambda}$ (or $\boldsymbol{\lambda}'$) that minimizes the average validation loss is then retained:

$$J(\boldsymbol{\lambda}) \;=\; \frac{1}{|\mathcal{D}_{\text{val}}|} \sum_{(\mathbf{x},y)\in\mathcal{D}_{\text{val}}} \ell\big(h_{\boldsymbol{\Theta}^*(\boldsymbol{\lambda})}(\mathbf{x}),\, y\big), \tag{2.6}$$

$$\boldsymbol{\lambda}^* \;=\; \arg\min_{\boldsymbol{\lambda}\in\Lambda} J(\boldsymbol{\lambda}). \tag{2.7}$$

The final h is then constituted by the associated parameter set $\boldsymbol{\Theta}^*(\boldsymbol{\lambda}^*)$ (or $\boldsymbol{\Theta}^*(\boldsymbol{\lambda}'^*)$), which is expected to generalize best to unseen traffic. In the case of a score-based detector using the decoupled convention, the final decision threshold $\tau(\boldsymbol{\lambda}^*)$ is then calibrated by selecting the value that maximizes the chosen performance metric on the validation set.

Evaluation metrics and deployment-time operation

During evaluation, $\mathcal{D}_{\text{test}}$ is used to measure how the tuned detection model performs on completely new traffic. This unbiased assessment approximates real-world performance. typically measured using standard metrics, which vary based on the model's output.

For models that output a specific class label, metrics like Accuracy, Precision, Recall, and F1-score are calculated. For score-based detectors, evaluation can be approached in two ways:

performance can be reported using these same threshold-dependent metrics at the specific operating point τ chosen during validation, or it can be measured across all possible thresholds using summary statistics like the Area Under the Curve (AUC) of the Receiver Operating Characteristic (ROC) or Precision-Recall (PR) curves. Although these measures are most prominently reported on the final test set, they can also guide training and validation.

After this final evaluation confirms the model's effectiveness, the system is deployed. Once deployed, it continuously monitors network observations in real time, detecting intrusions based on the final parameters $\boldsymbol{\Theta}^*$ and the best $\boldsymbol{\lambda}^*$ (and τ^* in the case of score-based detection).

2.2.2 Hyperparameter Optimization Methods for NIDS

Objective, notation, and threshold handling

As established in the previous section, the configuration optimization process involves searching for an optimal hyperparameter configuration $\boldsymbol{\lambda}$ over a search space Λ (or an augmented $\boldsymbol{\lambda}'$ over Λ' for score-based detectors that includes τ) that minimizes a chosen validation metric. To carry out this search efficiently, the literature offers a variety of techniques, ranging from classical grid or random search to more advanced Bayesian and evolutionary approaches.

For simplicity in the following notation, we write $\boldsymbol{\lambda}$ for the configuration throughout. In score-based detectors using the joint convention, interpret $\boldsymbol{\lambda}$ and Λ as the augmented $\boldsymbol{\lambda}' = (\boldsymbol{\lambda}, \tau)$ and $\Lambda' = \Lambda \times \mathcal{T}$; under the decoupled convention, τ is calibrated per $\boldsymbol{\lambda}$ and is not part of $\boldsymbol{\lambda}$.

Grid search

A classical approach, grid search, exhaustively explores the Cartesian product of finite sets for each hyperparameter dimension:

$$\Lambda \;=\; \Lambda_1 \times \Lambda_2 \times \ldots \times \Lambda_K,$$

where each Λ_k is a discrete set of possible values for hyperparameter k. Every combination $\boldsymbol{\lambda}$ is evaluated on $\mathcal{D}_{\text{val}}$, and the one yielding the best validation performance is chosen. Although simple to implement, grid search grows exponentially with the number of hyperparameters. For instance, if each dimension is tried at n levels, the total evaluations become n^K, quickly becoming computationally infeasible for large K.

Random search

Random search alleviates some of the combinatorial explosions by sampling each hyperparameter from a predefined probability distribution (e.g., a log-uniform distribution for the learning rate). Suppose we fix a total budget of N evaluations. For the j-th trial, we draw a hyperparameter vector $\boldsymbol{\lambda}^{(j)}$ from a factorized joint distribution

$$p(\boldsymbol{\lambda}) \;=\; \prod_{k=1}^{K} p_k(\lambda_k),$$

where each λ_k is sampled independently of $p_k(\lambda_k)$. Formally,

$$\boldsymbol{\lambda}^{(j)} \;\sim\; p(\boldsymbol{\lambda}), \quad j = 1, \ldots, N.$$

After all N samples are drawn and evaluated on the validation metric, the best set $\boldsymbol{\lambda}^*$ is retained. Because random search focuses on exploring broader regions rather than systematically enumerating every candidate, it often outperforms grid search in high-dimensional spaces. In particular,

if certain hyperparameters have a minor effect on model performance, random sampling frees the search to concentrate on those parameters that truly matter. This factorized approach also allows each hyperparameter dimension to be drawn from a specialized distribution that captures its plausible range (e.g., log-uniform for learning rates, uniform for regularization factors), further improving the chances of finding a near-optimal configuration within fewer trials.

Bayesian optimization

For expensive or high-dimensional searches, Bayesian optimization (BO) treats hyperparameter tuning as a sequential decision problem in which the validation objective $J(\boldsymbol{\lambda})$ (Eq. (2.6)) is an expensive black-box function to minimize, since each evaluation requires a full train-and-validate run.

Unlike grid or random search, which treat evaluations of $J(\boldsymbol{\lambda})$ as independent trials, BO builds a surrogate model $p\big(J \mid \boldsymbol{\lambda}\big)$ (e.g., a Gaussian Process or a Tree-Structured Parzen Estimator) to approximate the objective surface. After $t-1$ evaluations, we have observations $\{(\boldsymbol{\lambda}_i,\ J(\boldsymbol{\lambda}_i))\}_{i=1}^{t-1}$. BO uses these to form a posterior predictive distribution over the objective at any candidate $\boldsymbol{\lambda}$, and selects the next configuration by maximizing an acquisition function α:

$$\boldsymbol{\lambda}_t \ = \ \arg\max_{\boldsymbol{\lambda}\in\Lambda} \alpha\Big(\boldsymbol{\lambda}\,;\, p\big(J \mid \boldsymbol{\lambda}_{1:t-1},\ J(\boldsymbol{\lambda}_{1:t-1})\big)\Big).$$

The acquisition function balances exploitation (testing configurations predicted to achieve low J) and exploration (testing regions where the surrogate is uncertain). By iteratively updating its surrogate, BO typically reaches near-optimal configurations with far fewer train-and-validate evaluations than naive methods. BO is often run sequentially, but batch or parallel variants can evaluate multiple candidates per iteration to reduce wall-clock time.

Population-based search

In contrast to Bayesian optimization, which follows a single surrogate-guided trajectory, population-based techniques maintain a collection (a population) of candidate hyperparameter vectors. Let $P_t = \{\boldsymbol{\lambda}_t^{(1)}, \ldots, \boldsymbol{\lambda}_t^{(m)}\}$ denote the population at iteration t. Each candidate $\boldsymbol{\lambda}_t^{(i)}$ is evaluated by training a model under this configuration and measuring the validation objective $J(\boldsymbol{\lambda}_t^{(i)})$ defined in Eq. (2.6) (e.g., average validation loss).

The selection step then retains or prioritizes candidates with the lowest objective values, for example by keeping the top-ranked configurations (elitism) or by sampling parents with probability biased toward better fitness. New candidates are produced by applying variation operators such as mutation (randomly perturb one or more hyperparameters) and crossover (combine parts of two parent vectors):

$$\boldsymbol{\lambda}_{t+1}^{(i)} \ \sim \ \mathrm{Mutate}\Big(\mathrm{Crossover}\big(\boldsymbol{\lambda}_t^{(p_1)},\, \boldsymbol{\lambda}_t^{(p_2)}\big)\Big) \quad \text{for } i = 1, \ldots, m,$$

where p_1 and p_2 index the selected parent solutions. This process repeats until a fixed number of iterations is reached or convergence is observed. Because all candidates in P_t can be evaluated independently, population-based methods naturally support parallel exploration. This can be useful when J has multiple local optima, or when the best configuration changes over time as network conditions evolve.

2.2.3 Limitations of Conventional Hyperparameter Search for NIDS

While these HPO techniques are effective for finding a high-performing initial configuration, they share a fundamental limitation in the context of adaptable systems: they are *static, one-shot processes* designed for an offline setting. The underlying assumption is that an optimal set of hyperparameters can be found once and then remain frozen for deployment. This search process is computationally expensive and time-consuming, as it requires training and validating many model variations.

This static nature directly causes the critical limitation identified in Chapter 1. As previously established, even state-of-the-art adaptive NIDS maintain fixed configurations during their update cycles. While these systems actively adjust their model weights to handle concept drift, the hyperparameters guiding this adaptation remain static. Traditional HPO methods enforce this constraint because they are computationally expensive and require evaluating multiple model variants.

The primary drawback of this static approach becomes apparent in dynamic network environments. A hyperparameter configuration that is optimal for a given dataset can quickly become suboptimal as traffic patterns evolve. Similarly, the final alert threshold chosen during validation can drift out of alignment with changing operational costs and attack base rates, a problem known as *threshold staleness*. When a NIDS requires reconfiguration in response to such changes, these conventional methods provide no mechanism to reuse knowledge from previous tuning efforts. The entire expensive search must be *re-run from scratch*, creating a significant operational bottleneck and impairing a system's ability to adapt its learning behavior dynamically.

2.3 Meta-learning-based Auto-configuration

The previous sections established a critical gap in the literature: conventional HPO methods are static, computationally expensive, and must be re-run from scratch, making them ill-suited for the dynamic adaptation of a NIDS's internal learning behavior. To address this challenge, we introduce a meta-learning-based auto-configuration framework. The core principle of this approach is to *learn from experience*: instead of solving the HPO problem in isolation for each network context, our method leverages knowledge from previous, offline tuning tasks to rapidly infer an effective configuration for a new, unseen environment. This paradigm offers two significant advantages. First, it drastically reduces the time and computational resources required for reconfiguration. Second, it resolves a fundamental limitation of HPO for anomaly detectors by enabling the joint inference of both hyperparameters and the decision threshold using only benign traffic.

We first state the threat model and operational assumptions, then describe the two-phase framework.

2.3.1 Threat Model

Our meta-learning framework is designed to operate under a specific set of assumptions about the environment and the nature of the threats it helps to mitigate. These assumptions define the operational context in which the proposed method is effective.

The primary threat addressed by our auto-configuration framework is not a specific attack class (e.g., DDoS or Port Scan), but rather the degradation of a NIDS's detection performance due to changes in the network context. Such changes can be caused by a variety of factors:

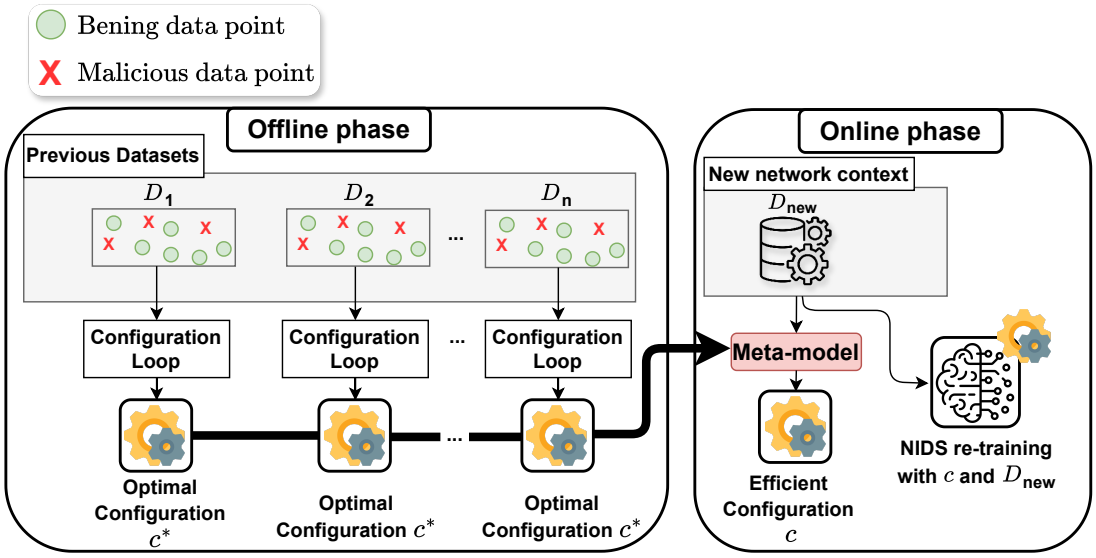


Figure 2.2: Two-phase meta-learning framework for NIDS autoconfiguration. The offline phase builds a meta-model from past tuning outcomes; the online phase uses a small sample from a new context to infer a configuration in one shot.

- **Adversarial Adaptation:** Attackers may introduce new attack variants or modify existing techniques, altering the statistical properties of malicious traffic.
- **Benign Network Evolution:** The deployment of new applications, changes in user behavior, or shifts in network topology can alter the characteristics of normal traffic, leading to an increase in false alarms if the NIDS is not re-calibrated.

Our framework assumes a threat environment where these context shifts occur, rendering a statically configured NIDS increasingly unreliable over time. The goal of the NIDS being configured is to detect specific attacks; the goal of our framework is to ensure the NIDS remains optimally configured to do so, even as the environment changes.

2.3.2 Overview

Our approach reframes the problem of autoconfiguration as a *supervised prediction over network contexts*. Instead of treating it as a repetitive search that must be started from scratch for each new network context, we treat it as a prediction task that can be solved in a single shot. The core of the method is to leverage knowledge from historical runs in the pre-deployment space, embed this knowledge into a predictive model, and reuse at configuration time for new contexts, thereby shortening time-to-deploy.

The NIDS configuration problem is therefore divided into two phases: an upstream (offline) phase that consolidates knowledge from past experience, and a downstream, operational (online) phase that uses this knowledge for fast configuration. Figure 2.2 illustrates the high-level idea of this two-stage approach, where optimal configurations determined offline are transferred to the online phase via a meta-model, which encapsulates this consolidated knowledge. The online phase is the active period during which the NIDS operates. Details of the meta-model's construction will be provided in Section 2.3.3.

Table 2.1: Meta-features used to build the context descriptor $\phi(\mathcal{D})$ (fast, model-agnostic). Items marked † require labels (supervised-only).

Simple meta-features	**Statistical meta-features**
Counts and ratios: number of instances; total number of attributes; number of numeric features; number of categorical features; number of binary attributes; ratios (attributes/instances, instances/attributes, categorical/numeric); number of distinct classes† ; relative class frequencies†; class-imbalance ratios† (optional). Structure and quality: number of distinct correlated attributes; number of attributes with at least one outlier; covariance eigenvalues.	**Per-feature summaries:** mean, median, trimmed mean, min, max, range; geometric and harmonic means. **Dispersion and shape:** variance, standard deviation, interquartile range (IQR), median absolute deviation (MAD), skewness, kurtosis; sparsity. **Correlation & multivariate tests:** absolute inter-attribute correlations; canonical correlations; number of canonical correlations between each attribute and the class†; Pillai's trace †, Lawley–Hotelling trace†, Wilks' lambda† ; dataset centroid norm ("gravity").

The entire workflow operates on three realistic assumptions. First, we assume that changes in network context are indirectly embedded in the generated data. For example, an increase in the number of hosts or modifications in network topology would not only change the volume but also impact the behavioral patterns of the observed network flows. Likewise, the introduction of a new application class will reshape flow packet-size distributions, and the TCP/UDP mix, even if total traffic volume remains similar.

Second, we assume access to a collection of datasets that have been characterized offline, including attack identification, which represents knowledge acquired from in-lab testing or post-deployment analysis. As shown in Figure 2.2 , we retain from each pre-deployment cycle only the final output of the configuration loop (Figure 2.1): the optimal hyper-parameter configuration $\boldsymbol{\lambda}^*$ (and, for score-based detectors, the operating threshold $\boldsymbol{\Theta}^*(\boldsymbol{\lambda}'^*)$). In other words, we take the result of the outer-level optimization carried out across the configuration-training and validation stages and use these outcomes as prior knowledge. This prior knowledge feeds the predictive model queried online, replacing the conventional loop and avoiding time-consuming optimization iterations.

Third, at deployment in a new network context, we assume that a small sample of recent traffic can be collected as a new dataset D_{new}. Its annotation status may vary (fully labeled, partially labeled, or unlabeled) depending on the detector and operational constraints. Using D_{new} together with the learned mapping, an effective configuration $(\boldsymbol{\lambda}, \tau)$ (including the operating threshold τ when the detector outputs scores) is inferred in one shot, and the detector is retrained once under these settings to adapt to the new context.

2.3.3 Offline Phase: Meta-modeling and Objective

Offline goal and intuition

To effectively transfer the optimal configurations, calculated during the offline phase, to the online phase, a meta-model is used. Figure 2.3 illustrates its construction.

Conceptually, the idea is to use the meta-model to integrate network contexts details during the transfer, as opposed to conducting an un-contextualized one. The aim is to transfer optimal configurations while taking into account the *characteristics* of both the source network contexts, where these configurations are originally effective, and the target network context, where efficient functioning of the NIDS is required. It is thus a more informed transfer by channeling multiple

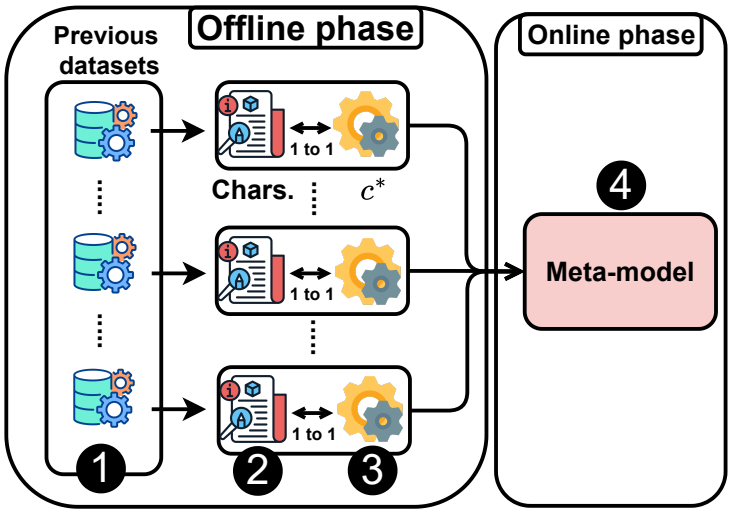


Figure 2.3: Meta-model construction in the offline phase. Each past context is summarized into meta-features, paired with the optimal configuration found offline, and used to train a supervised predictor.

optimal configurations alongside their respective network context summaries.

Offline construction pipeline

Practically, as depicted in Figure 2.3, the meta-model's construction begins by using each previous dataset in ① to determine the corresponding optimal configuration $\boldsymbol{\lambda}^\star$ (and, for score-based detectors, $\tau^\star$) to be used in ③ via the configuration loop. In parallel, each dataset is summarized into a compact descriptor $s = \phi(\mathcal{D})$ (step ②).

Context descriptor and meta-features

Rather than feeding raw, high-dimensional traffic to the meta-model, we compute a low-cost, detector-agnostic numerical fingerprint of the context. This follows standard practice in meta-learning and AutoML [Van19, HKV19, FKE+15], in which transfer across tasks is facilitated by task descriptors that capture stable, comparable properties of datasets.

Here, these descriptors encode (i) size and composition (counts and ratios), (ii) dispersion and shape (variances, skewness, kurtosis, IQR, MAD), and (iii) dependency structure (correlations, covariance eigenvalues, simple multivariate tests). The design is intentionally lightweight (fast to compute), model-agnostic (not tied to any detector family), and robust when labels are scarce. If a small set of label exists, we optionally append shallow supervision signals (e.g., imbalance ratios or a few confusion-matrix entries) to make the fingerprint sensitive to the intended operating point.

Table 2.1 provides the detailed list. The left column (Simple meta-features) contains coarse but informative descriptors of dataset structure, such as counts, ratios, and basic quality indicators. The right column (Statistical meta-features) summarizes the empirical distribution of features through location, dispersion, and shape statistics, and includes correlation-based descriptors to capture dependency structure. Items marked † require labels and are therefore only available for supervised settings. In label-scarce or unsupervised anomaly detection, the descriptor can be computed using only the label-free subset of Table 2.1. For example, consider a new

flow dataset collected from an enterprise network. Without any labels, $\phi(\mathcal{D})$ can still include the number of flows, the number of flow features, the fraction of sparse attributes, dispersion measures (IQR, MAD) that capture variability in flow sizes and durations, and correlation summaries that capture dependencies between flow counters. These statistics act as a compact signature of the new context, which enables the meta-model to condition the transfer on similarity: configurations that were optimal for past datasets with comparable size, variability, and dependency structure become more likely candidates for the new environment. If labels are later available, the descriptor can be extended with class-imbalance ratios to make the mapping more sensitive to supervised operating targets.

Meta-dataset and learning signal

Coupling each descriptor with the outcome of the outer configuration loop creates supervised training signal for the meta-model: a pair $(s, \boldsymbol{\lambda}^\star)$ (or a triplet $(s, (\boldsymbol{\lambda}^\star, \tau^\star))$ when thresholds apply) represents the finding that for a given context, a specific configuration was found to be optimal offline. Aggregating many such samples across heterogeneous contexts yields a *meta-dataset* that encodes patterns between statistical properties of the data and effective settings.

Training the meta-model on this meta-dataset (step ④) can be understood as learning a reusable optimizer. The cost of the expensive, iterative HPO process is transfered to the offline training phase. Consequently, at deployment time, this costly outer-loop search is replaced by a single, computationally inexpensive forward pass through the trained meta-model.

This approach is also flexible, as the meta-model can be designed to handle the structured and mixed-type output required for a full detector configuration, simultaneously predicting continuous hyperparameters, categorical choices, and the decision threshold where applicable.

Formal objective

Formally, our goal is to learn a meta-model, a function F_φ parameterized by φ, that maps the descriptor of a new dataset directly to a near-optimal hyperparameter configuration. This allows us to bypass the expensive, iterative search for $\boldsymbol{\lambda}^*$ defined in Equation 2.7. The meta-model takes a dataset's compact descriptor $s = \phi(\mathcal{D}) \in \mathbb{R}^p$ as input and outputs a predicted configuration:

$$\widehat{\boldsymbol{\lambda}} = F_\varphi(s). \tag{2.8}$$

In the case of score-based detectors where the threshold is included, the mapping becomes $F_\varphi : \mathbb{R}^p \to \Lambda'$ and the output is the augmented vector $(\widehat{\boldsymbol{\lambda}}, \widehat{\tau})$.

To train the meta-model F_φ, we construct a meta-dataset from a collection of historical pre-deployment episodes $\{(\mathcal{D}^{(e)})\}_{e=1}^{E}$. For each episode e, we compute its descriptor $s^{(e)} = \phi(\mathcal{D}^{(e)})$ and use a conventional HPO process to find its "oracle" or ground-truth optimal configuration, which we denote $\mathbf{y}^{(e)}$. This target $\mathbf{y}^{(e)}$ is either $\boldsymbol{\lambda}^{*(e)}$ or the augmented vector $(\boldsymbol{\lambda}^{*(e)}, \tau^{*(e)})$ for score-based detectors.

The parameters φ of the meta-model are then learned by fitting F_φ to reproduce these oracle configurations across all episodes, typically by minimizing a loss function:

$$\varphi^\star \in \arg\min_{\varphi} \sum_{e=1}^{E} \ell\left(F_\varphi\left(s^{(e)}\right), \mathbf{y}^{(e)}\right) + \Omega(\varphi), \tag{2.9}$$

where $\ell(\cdot, \cdot)$ is a loss function suitable for the mixed-type structure of the configuration vector (e.g., squared error for numeric hyperparameters and cross-entropy for categorical ones), and $\Omega(\varphi)$ is an optional regularizer on the meta-model's parameters.

2.3.4 Online Phase: On-the-fly Configuration

The online phase is designed for speed and efficiency, leveraging the knowledge consolidated in the pre-trained meta-model to bypass the costly HPO loop at deployment time. When the NIDS is introduced to a new network context, the process of finding an effective configuration is reduced to a simple, three-step workflow as shown in the "Online phase" of Figure 2.2.

First, a small sample of recent traffic from the new environment is collected to form a new dataset, D_{new}. The system then computes its compact descriptor, $s_{\text{new}} = \phi(\mathcal{D}_{\text{new}})$, using the same meta-feature mapping as the offline phase. Critically, this step is flexible regarding data annotation. While the framework can operate using only benign traffic (making it practical to deploy anomaly detectors without prior attack knowledge) it can also leverage attack labels if they are available in D_{new}. In such a supervised scenario, the descriptor can be enriched with more informative meta-features (e.g., class imbalance ratios) to guide the configuration process.

Second, this new descriptor s_{new} is fed as input to the pre-trained meta-model. In a single, computationally inexpensive forward pass, the meta-model infers the predicted configuration $\hat{\boldsymbol{\lambda}}$ (and $\hat{\tau}$ for score-based detectors) without performing any iterative search.

Finally, the inferred configuration is used to train the NIDS model a single time on the available data from D_{new}. After this one training run, the detector is adapted to the new context and is ready for deployment. This entire process replaces the iterative, trial-and-error search of conventional HPO with a single inference and a single training run, dramatically shortening the time-to-deploy for a new or reconfigured NIDS.

2.4 Conclusion

This chapter presented a meta-learning framework that reframes NIDS auto-configuration as learning from experience rather than re-tuning from scratch. We formalized the detection model and its configuration space, including the operating threshold for score-based detectors, and showed how configuration can be defined as a supervised prediction over network contexts. The resulting two-phase design separates an offline, knowledge-consolidation stage from a lightweight online stage. Offline, historical pre-deployment episodes are synthesized into pairs of compact context descriptors and their corresponding oracle configurations. Training a predictive meta-model on this collection of pairs resolves the heavy computational cost of hyperparameter optimization upfront. At deployment time, a new context is summarized once, the learned mapping is queried once to obtain the adapted configuration (with the threshold when applicable), and the base detector is retrained a single time under these settings. This one-shot procedure removes the outer HPO loop at deployment, reducing computation time while remaining detector-agnostic.

While the proposed framework offers a significant step towards automated NIDS adaptation, it is important to acknowledge its inherent limitations. The primary constraint is its dependence on historical data, commonly referred to as the cold-start problem. A new organization with no prior tuning history must first build this knowledge base using conventional, time-consuming HPO methods. Consequently, the performance and generalization capability of the meta-model are directly proportional to the size and diversity of the historical datasets used in the offline phase. Furthermore, the approach is highly sensitive to meta-feature quality. If the chosen statistical descriptors cannot adequately distinguish between varying network contexts, the meta-model will fail to make accurate predictions, making the careful engineering of these features critical to the system's success. Lastly, the scope of automation is strictly limited to configuring a given detection model. It does not automate other essential pipeline steps, such as initial algorithm selection or feature engineering, but rather functions as an HPO-specific solution

designed to integrate into a broader NIDS lifecycle.

In summary, this framework is designed to accelerate the reconfiguration of a NIDS in environments where historical data is available and where context changes are reflected in the statistical properties of the traffic. It is not a universal solution for all deployment scenarios but rather a powerful tool for organizations seeking to maintain the performance of their ML-based defenses in dynamic environments. The next chapter evaluates this framework empirically.

Chapter 3

Evaluation of the Auto-Configuration Framework

Contents

3.1 Introduction 55
3.2 Experimental Setup 56
3.2.1 NIDS Design and Implementation 56
3.2.2 Datasets 58
3.2.3 Nested Cross-Validation for Oracle Configurations 60
3.2.4 Evaluation Metrics for Classification and Score-based Detection . . 61
3.3 Empirical Evaluation of the Meta-Learning Framework for NIDS Classification 63
3.3.1 Detection Performance 63
3.3.2 Resource usage at Deployment Stage 65
3.3.3 Meta-dataset size 67
3.4 Empirical Evaluation of the Meta-Learning Framework for AD NIDS Configuration 67
3.4.1 Robustness 68
3.4.2 Anomaly Detection 72
3.4.3 Configuration Generation Time 73
3.4.4 Meta-learning Relative Performance 73
3.5 Conclusion 74

3.1 Introduction

The preceding chapters established a critical gap in the literature and proposed a novel meta-learning framework to address it. We argued that by reframing hyperparameter optimization as a supervised prediction task, our two-phase approach could dramatically reduce the time-to-deploy for a reconfigured NIDS and, crucially, enable the configuration of anomaly detectors on benign-only data.

This chapter now provides a comprehensive empirical evaluation to validate these claims. The primary goal is to assess the performance of our autoconfiguration framework under realistic conditions and quantify the trade-offs between our rapid, inference-based approach and the

computationally expensive but optimal configurations found by conventional Bayesian Optimization.

To achieve this, the chapter is structured as follows. First, we detail the experimental setup, describing the NIDS and Anomaly Detection (AD) NIDS models used, the datasets, the HPO process for establishing oracle configurations, and the evaluation metrics. Following this, we present the core results in two separate sections: one for the supervised NIDS Classification setting, where the goal is to infer optimal hyperparameters, and one for the AD NIDS Configuration setting, which involves the more complex task of jointly inferring both hyperparameters and the decision threshold. The chapter concludes with a discussion that synthesizes these findings and analyzes the overall effectiveness of the proposed framework.

3.2 Experimental Setup

3.2.1 NIDS Design and Implementation

In our empirical study, we instantiate the framework with two detector families to cover the two canonical decision outputs. (i) Supervised NIDS (classifier): the detector outputs a discrete label $\hat{y}$; the meta-model therefore predicts only the hyper-parameter vector $\boldsymbol{\lambda}$, and the operating point is implicit in the decision rule. (ii) Anomaly detection NIDS (score-based): the detector outputs a real-valued score $s^{(\boldsymbol{\lambda})}(\mathbf{x})$; the configuration must specify both the internal hyper-parameters and the alert threshold, i.e., $\boldsymbol{\lambda}' = (\boldsymbol{\lambda}, \tau)$. Evaluating both settings allows us to assess the proposed method when it infers $\boldsymbol{\lambda}$ (a classifier case) and when it jointly infers $(\boldsymbol{\lambda}, \tau)$ (an AD case), using the same flow-level feature space and preprocessing pipeline.

Network Intrusion Detection System (NIDS)

The supervised classifier evaluated in this study is a Random Forest (RF), an ensemble learning method that constructs a multitude of decision trees at training time. For classification, the final decision is made by a majority vote among all the individual trees. This approach is known for its robustness and high accuracy. For our experiments, we use the implementation provided by the `scikit-learn` library.

Based on prior work showing their significant impact on performance [BHA09, PWB19], two main hyperparameters are tuned for the Random Forest classifier:

- *max features*: This controls the number of features to consider when looking for the best split at each node in a tree.
- *min samples leaf*: This defines the minimum number of samples required to be at a leaf node, which controls the complexity of the trees and helps prevent overfitting.

The hyperparameter vector is denoted by $\boldsymbol{\lambda}$, and its search space Λ is the Cartesian product of the value ranges for these two hyperparameters. Because the Random Forest classifier outputs a discrete class label, a decision threshold τ is not required, and we therefore use the standard configuration $\boldsymbol{\lambda}$ over the search space Λ.

Anomaly Detection NIDS (AD NIDS)

The anomaly detector evaluated in this study is *Isolation Forest* (IF) [LTZ08b], a tree-based method that isolates abnormal points rather than modeling the density of normal behavior.

Each binary tree recursively selects a split variable and a split value at random; anomalies are therefore isolated close to the root, yielding shorter path lengths and larger anomaly scores. We rely on the high-performance `isotree` library [Cor23], which incorporates recent improvements for faster tree construction and clustered-outlier detection.

Seven hyperparameters are tuned because prior work shows they control most of IF's bias–variance trade-off [HCKB19, Cor21, CTBC22]:

- *Sample size* decides how many flows populate each tree; smaller samples speed training but may overlook rare benign patterns.
- *The number of trees* affects score stability: more trees reduce variance at higher computational cost.
- *Maximum depth* limits tree growth; shallow trees force early isolation and therefore raise anomaly scores for moderate deviations.
- *Split-feature count* governs the number of candidate variables considered per split, balancing exploration against over-fitting to dominant features.
- *Linear-combination probability* allows splits on random linear projections, capturing multivariate structure that axis-aligned splits would miss.
- *Gain threshold* sets the minimum information gain required to continue splitting, controlling tree granularity in noisy regions.
- *The number of projections tested* determines how many random combinations are evaluated before the best split is chosen.

The IF hyperparameter vector is denoted by $\boldsymbol{\lambda}$, and its search space Λ is the Cartesian product of the seven individual hyperparameter spaces. Because IF outputs a real-valued anomaly score, it requires an operating threshold τ to convert scores into binary decisions. In Isolation Forest, this score is derived from how quickly a sample is *isolated* by the ensemble of randomly built trees: for each tree, a sample traverses a sequence of random splits until it reaches a leaf, and the number of splits along this route is its path length. Samples that are isolated in fewer splits (shorter average path length across trees) are assigned higher anomaly scores, whereas samples that require more splits to isolate (longer paths) receive lower scores. Following the notation for score-based detectors, we therefore use the augmented configuration $\boldsymbol{\lambda}' = (\boldsymbol{\lambda}, \tau)$, with the search conducted over the augmented space $\Lambda' = \Lambda \times \mathcal{T}$.

Oracle configurations and baseline tuning

For the oracle configurations used offline and as baselines, we run Bayesian optimization (BO) over the detector's configuration space (RF: Λ; IF: $\Lambda' = \Lambda \times \mathcal{T}$ when the threshold is included). BO is initialized with five quasi-random seed points (e.g., Sobol/Latin hypercube), then fits a surrogate to the validation objective $\widetilde{J}(\boldsymbol{\lambda}) := J\big(h^{(\boldsymbol{\lambda})}, \mathcal{D}_{\text{val}}\big)$. Candidates are selected by maximizing Expected Improvement. The budget is 200 function evaluations (train–validate cycles), and we retain the best observed setting as the approximate optimum $\boldsymbol{\lambda}^*$ (or $(\boldsymbol{\lambda}^*, \tau^*)$ for score-based detectors).

3.2.2 Datasets

Our empirical study uses the CIC-IDS2017 (IDS2017) and CSE-CIC-IDS2018 (IDS2018) benchmarks [Sha18], two public, *flow-based* datasets that emulate enterprise traffic in a controlled testbed. Both are released as day-by-day captures comprising **14** daily contexts, with each record a bidirectional network flow described by NetFlow-style features. In addition to the original releases, we also rely on a revised version [LEL+22] that corrects labeling artifacts and removes inconsistencies reported in prior work (e.g., mislabels, duplicated or near-empty records), and that consolidates day-level files when appropriate.

The revised version is constructed directly on top of the original releases. It reprocesses the original packet captures and metadata with improved tooling, but does not generate any new traffic. The corrections address (i) mislabeled benign/attack flows and inconsistent attack timing; (ii) duplicated or near-empty artifact records; (iii) flow-construction errors (e.g., packet misordering, incorrect aggregation/splitting, protocol detection issues) that distorted flow attributes; and (iv) feature inconsistencies (NaNs/Infs, categorical coding mismatches) and documentation mismatches [LEL+22].

IDS2017 offers five daily traces; the first is purely benign and is therefore discarded, leaving four operational days that mix legitimate flows with fourteen attack families. IDS2018 was recorded eleven months later in the same laboratory, including seven networks and about 500 virtual machines, and contributes ten additional daily captures that introduce further attack variations. For clarity, we index the fourteen remaining daily captures as $D_0, \ldots, D_{13}$; the four IDS2017 days are $D_i,\ i \in [0, 3]$, while the ten IDS2018 days are $D_i,\ i \in [4, 13]$. Each record is a bidirectional, flow-based observation described by the same 83 NetFlow-style features. Following the original preprocessing pipeline, we retain the 14 attributes listed in Table 3.1, which earlier IDS studies [JM22, ZMM21] have found most discriminative.

Table 3.1: Features considered in each dataset

Feature Name	Description
Src Port	Source service port numbers (one-hot encoded).
Dst Port	Destination service port numbers (one-hot encoded).
Protocol	Network protocol of the flow (one-hot encoded).
Flow Duration	Duration of the flow.
Total Fwd/Bwd Packets	Packets in the forward/backward direction.
Total Length of Fwd/Bwd Packet	Size of packets in forward/backward direction.
Flow Bytes/s	Bytes transferred per second.
Flow Packets/s	Packets transferred per second.
Fwd/Bwd IAT Total	Time between packets in fwd/bwd direction.
Fwd/Bwd Packets/s	Forward/backward packets per second.

Figure 3.1 visualizes the daily composition of benign traffic and 31 distinct attack families. Rows correspond to the fourteen capture days $D_0 - D_{13}$; columns correspond to traffic classes ordered by their taxonomy ID. Color intensity encodes the percentage share of each class within the day, so darker cells mark higher prevalence. A label beginning with "Att" denotes an *attempted* (unsuccessful) attack—for example, failed SQL injections or half-open TCP handshakes. This distinction is specific to the revised version: in the original release, attempted traces were merged into the corresponding attack class and not labeled separately.

Several trends are visible. First, benign flows dominate every day, frequently exceeding 80 %

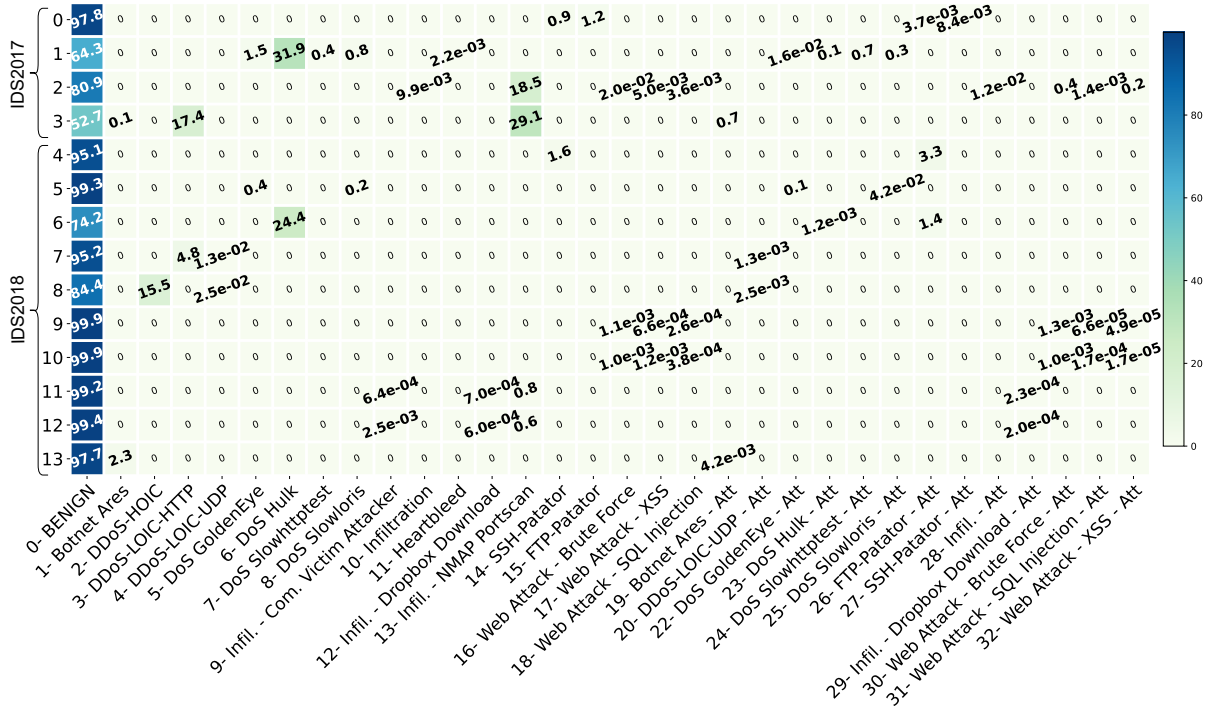


Figure 3.1: Daily traffic composition in IDS2017 (D0–D3) and IDS2018 (D4–D13). Rows denote days and columns denote traffic classes (benign and 31 attack families), ordered by taxonomy ID. Cell color shows the class share per day (% of flows) using a log-scaled color mapping; "Att." denotes unsuccessful attack attempts.

of the total volume, which mirrors the imbalance faced by real-world anomaly detectors. Second, high-volume classes such as DDoS and brute-force login (traffic IDs 4, 14, 15) still account for less than 2 % of daily flows, showing that high-volume attacks remain statistically infrequent. Third, the heat map indicates that *each day represents a distinct network context*: the mix of attacks and benign load varies enough that no two days exhibit identical distributions. The only near-repetitions occur in IDS2018, where Web attacks appear on both D_9 and D_{10}, and Infiltration traffic on D_{11} and D_{12}.

Figure 3.2 complements this view with a bar chart of bidirectional flow counts, confirming that traffic volume itself drifts substantially — from well below 200 000 flows on the quietest 2017 day to more than 600 000 on several 2018 days. Larger traffic volumes provide more normal flows, which in turn alter the effective subsampling rate and tree depth, two hyperparameters central to Isolation-Forest tuning.

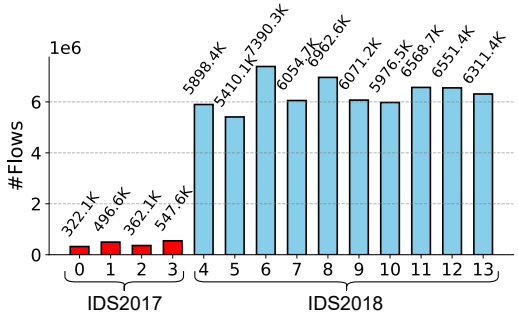


Figure 3.2: Flow Count Across Days

3.2.3 Nested Cross-Validation for Oracle Configurations

Evaluating a NIDS on the same data used to tune its hyperparameters leads to overly optimistic performance estimates. Nested cross-validation (CV) prevents this bias by dividing the process into two distinct stages: an inner loop dedicated entirely to configuration selection, and an outer loop reserved strictly for performance estimation. Because the tuning process never accesses the outer test data, the final evaluation provides a completely unbiased estimate of the system's true performance. Figure 3.3 summarizes this workflow.

For each dataset (episode) $\mathcal{D}$, we run a $K_{\text{out}}\times K_{\text{in}}$ nested CV to obtain an oracle configuration for the base detector $h^{(\boldsymbol{\lambda})}$. First, $\mathcal{D}$ is stratified into K_{out} outer folds; for each outer split t, we form $(\mathcal{D}_t^{\text{train}}, \mathcal{D}_t^{\text{test}})$ and reserve $\mathcal{D}_t^{\text{test}}$ exclusively for final scoring. On $\mathcal{D}_t^{\text{train}}$, we then perform K_{in} -fold CV to evaluate candidates $\boldsymbol{\lambda}\in\Lambda$ (or $\boldsymbol{\lambda}'\in\Lambda' = \Lambda\times\mathcal{T}$ for score-based detectors). The inner score for a candidate is

$$\overline{J}_{\text{in}}^{(t)}(\boldsymbol{\lambda}) \;=\; \frac{1}{K_{\text{in}}}\sum_{j=1}^{K_{\text{in}}} J\big(h^{(\boldsymbol{\lambda})} \text{ trained on } \mathcal{D}_{t,j}^{\text{tr}} \,\big|\, \mathcal{D}_{t,j}^{\text{val}}\big),$$

(with $\boldsymbol{\lambda}$ replaced by $\boldsymbol{\lambda}'$ in the score-based case), and we select the best-performing inner configuration $\widehat{\boldsymbol{\lambda}}_t \in \arg\max_{\boldsymbol{\lambda}\in\Lambda} \overline{J}_{\text{in}}^{(t)}(\boldsymbol{\lambda})$ (or the pair $(\widehat{\boldsymbol{\lambda}}_t, \widehat{\tau}_t)$ when τ is part of the configuration). Finally, we retrain $h^{(\widehat{\boldsymbol{\lambda}}_t)}$ on the full $\mathcal{D}_t^{\text{train}}$ and compute the outer score $J_{\text{out}}^{(t)}$ on $\mathcal{D}_t^{\text{test}}$ (applying $\widehat{\tau}_t$ when relevant). The outer score is used only for unbiased performance estimation and never for selecting the configuration.

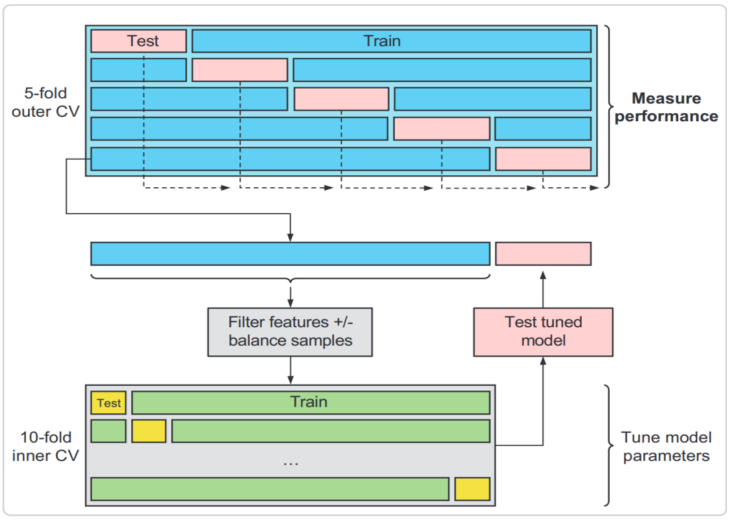


Figure 3.3: Nested cross-validation workflow: the inner loop selects the configuration on training data, while the outer loop evaluates generalization on held-out test folds. Adapted from [LSG+23].

After all outer folds, we derive the episode's oracle configuration by aggregating inner winners:

$$\boldsymbol{\lambda}^* \in \arg \max_{\boldsymbol{\lambda} \in \{\widehat{\boldsymbol{\lambda}}_1, \ldots, \widehat{\boldsymbol{\lambda}}_{K_{\text{out}}}\}} \frac{1}{K_{\text{out}}} \sum_{t=1}^{K_{\text{out}}} \overline{J}_{\text{in}}^{(t)}(\boldsymbol{\lambda}).$$

For score-based detectors, if τ is selected jointly, we analogously retain the pair $(\boldsymbol{\lambda}^*, \tau^*)$ that maximizes the mean inner score; if τ is decoupled, we calibrate $\tau^* = \tau(\boldsymbol{\lambda}^*)$ on $\mathcal{D}^{\text{train}}$ according to the chosen operating objective.

We use $K_{\text{out}} = 10$ and $K_{\text{in}} = 3$, with stratification when labels are present. The inner loop uses the Bayesian Optimization routine described previously, with a budget of 200 evaluations. J denotes the is chosen to be appropriate for the task (e.g., macro-F1 or MCC for classifiers; AUCPR or ROC-AUC for score-based detectors). The resulting pair $(s = \phi(\mathcal{D}), \mathbf{y})$ (with $\mathbf{y} = \boldsymbol{\lambda}^*$ or $(\boldsymbol{\lambda}^*, \tau^*)$) constitutes one meta-example for training the meta-model, while the set $\{J_{\text{out}}^{(t)}\}_{t=1}^{K_{\text{out}}}$ provides unbiased generalization estimates.

3.2.4 Evaluation Metrics for Classification and Score-based Detection

This section details common metrics used to evaluate either (i) a classification task (e.g. distinguishing *benign* from *malicious* observations), or (ii) an anomaly-detection task (e.g. assigning a continuous anomaly score and applying a threshold). Other tasks (e.g. regression) lie outside the scope of this thesis. Although classification and anomaly detection differ in how predictions $\hat{y}$ are generated (discrete vs. continuous), many of the underlying metrics—such as confusion matrix analysis, Precision/Recall, and ROC curves—are closely related.

In a classification setting, a practical way to summarize how well predictions $\hat{y}$ match ground-truth labels y is through a *confusion matrix* $\mathbf{C} = (c_{k,j})$, where k indexes the *true class* and j indexes the *predicted class*, both ranging over $\{1, \ldots, K\}$. In a NIDS context, one typically labels *malicious* traffic as the positive class and *benign* traffic as the negative class, reflecting the main detection goal: capturing malicious events (positives) and measuring how often benign traffic

is misclassified (negatives). Multi-class variants can further distinguish among multiple attack categories or subcategories of malicious behavior.

In the simplest *binary* case (*benign* vs. *malicious* with no further attack-type distinctions), $\mathbf{C}$ reduces to a 2×2 table:

	Predicted Malicious	Predicted Benign
Actual Malicious	True Positives (TP)	False Negatives (FN)
Actual Benign	False Positives (FP)	True Negatives (TN)

(3.1)

Here, TP is the number of malicious observations correctly identified as malicious, FP is how many benign observations were incorrectly labeled as malicious, FN is how many malicious observations were overlooked, and TN is how many benign observations were correctly recognized as benign.

When the classification problem extends to multiple categories (e.g. *benign* plus several attack types), the confusion matrix $\mathbf{C}$ is a $K \times K$ matrix. The diagonal entries c_{kk} then represent correct predictions for class k, whereas off-diagonal entries indicate various misclassifications. To generalize *true positives*, *false positives*, *false negatives*, and *true negatives* for each class k, one can temporarily treat k as the positive class and group all other classes into negative. Concretely,

$$\mathrm{TP}_k = c_{k,k}, \quad \mathrm{FP}_k = \sum_{\ell \neq k} c_{\ell,k}, \quad \mathrm{FN}_k = \sum_{\ell \neq k} c_{k,\ell}, \quad \mathrm{TN}_k = \sum_{\ell \neq k} \sum_{m \neq k} c_{\ell,m}. \tag{3.2}$$

Here, TP_k is how many instances of class k are correctly identified, FP_k captures how often class k is predicted when it is not actually present, FN_k is how many instances of class k are missed, and TN_k counts the observations that neither belong to k nor were predicted as k.

From these quantities, one can define common metrics. *Precision* (Prec_k) evaluates the fraction of predicted class-k items that indeed belong to k, while *Recall* (Rec_k) measures the fraction of class-k items that the model correctly identifies:

$$\mathrm{Prec}_k \;=\; \frac{\mathrm{TP}_k}{\mathrm{TP}_k + \mathrm{FP}_k}, \qquad \mathrm{Rec}_k \;=\; \frac{\mathrm{TP}_k}{\mathrm{TP}_k + \mathrm{FN}_k}. \tag{3.3}$$

Balancing these two rates, the *F-score* for class k (often the F_1-measure) is

$$F_{1,k} \;=\; \frac{2\,\big(\mathrm{Prec}_k\,\mathrm{Rec}_k\big)}{\mathrm{Prec}_k + \mathrm{Rec}_k}. \tag{3.4}$$

When more than two classes are involved ($K > 2$), it is often desirable to combine per-class scores into a single overall summary metric. The *macro-mean* metric

$$\overline{M} \;=\; \frac{1}{K} \sum_{k=1}^{K} M_k \tag{3.5}$$

treats all classes equally by averaging a per-class metric M_k (such as Precision, Recall, or F-score), regardless of how frequently each class appears. An alternative is the *micro-mean* value

$$\widetilde{M} \;=\; \frac{\sum_k \mathrm{TP}_k}{\sum_k \big(\mathrm{TP}_k + \mathrm{FP}_k\big)}, \tag{3.6}$$

which aggregates the total counts of true positives, false positives, and false negatives across all classes before computing an overall ratio; it thus gives greater weight to classes with more samples.

Another popular single-number measure is the *Matthews correlation coefficient* (MCC), which extends naturally to K classes via

$$\mathrm{MCC} = \frac{\sum_k \sum_\ell \sum_m c_{kk}\, c_{\ell m} - c_{k\ell}\, c_{mk}}{\sqrt{\left(\sum_{k,\ell} c_{k\ell}\right)\left(\sum_{k\neq\ell} c_{k\ell}\right)\left(\sum_{k,\ell} c_{\ell k}\right)\left(\sum_{k\neq\ell} c_{\ell k}\right)}}\,. \tag{3.7}$$

Because it factors in true and false classifications for all labels, MCC tends to be especially informative when class frequencies are imbalanced.

In the *anomaly-detection* setting, the detection model outputs a real-valued score rather than a hard label. One then applies a threshold τ to decide which observations are flagged as anomalous (positive) versus normal (negative). By fixing τ, the task effectively becomes a binary classification problem, allowing the standard classification metrics (Precision, Recall, F1, etc.) to be applied. By shifting τ across the full range of possible scores, each setting produces a different balance of true positives (malicious events correctly flagged) and false positives (benign events wrongly flagged). Plotting these balances for every τ yields *ROC* (receiver operating characteristic) curves—visualizing the true-positive rate (TPR) vs. false-positive rate (FPR)—or *precision–recall* (PR) curves. This approach offers insight into the model's discriminative power at each operating point, rather than committing to a single threshold in advance. The areas under these curves, AUROC or AUPRC, provide threshold-independent summaries of overall performance. Moreover, once a particular false-alarm rate or detection rate is deemed acceptable, one can select the corresponding τ to meet those practical requirements.

3.3 Empirical Evaluation of the Meta-Learning Framework for NIDS Classification

For evaluation purposes, a positive sample corresponds to an attack flow and a negative sample to a normal flow. We report four evaluation metrics: MCC, macro-mean F1-score, Precision, and Recall. We compare the performance of our auto-configuration approach (denoted as MtL) against two baselines: a default configuration provided by the ML toolchain, which does not depend on the data (in `scikit-learn`, *max_features* is set to 3^{14} and *min_samples_leaf* to 1), and an oracle configuration returned by Bayesian optimization (BO) (denoted as HPO).

Each day is treated as a separate dataset that requires its own configuration. To predict a configuration for a given day, we train a meta-model on the other days. In this classification setting, the meta-model is implemented as a Random Forest regressor (RFr): it takes simple dataset characteristics (meta-features) as input and predicts the two Random Forest hyperparameters. When running BO to produce oracle configurations, we maximize MCC, which is well suited to summarizing classification performance under strong class imbalance.

3.3.1 Detection Performance

The experimental results are reported in Table 3.2. In this table, the column labeled `MtL` denotes the performance of our proposed auto-configuration solution, which is based on meta-learning. These experiments use the original IDS2017 and IDS2018 datasets release. In that release, the 14 daily contexts are distributed across 17 CSV files because two calendar days were split at source: (i) `Thursday2017` appears as two files (`Thursday2017-Morning-WebAttacks` and `Thursday2017-Afternoon-Infilteration`); and (ii) `Friday2017` appears as three files (`Friday2017-Morning`

[14] The result of the floor function of the square root of the number of features

Table 3.2: Comparison performance results (best value in green, worst value in red, intermediate value in black)

	MCC			f1-score			precision			recall		
day id	Default	HPO	MtL	Default	HPO	MtL	Default	HPO	MtL	Default	HPO	MtL
2	0.715	**0.757**	0.747	0.702	**0.729**	0.717	0.87	**1.0**	**1.0**	**0.588**	0.574	0.559
3	0.558	**0.619**	0.596	0.525	**0.554**	0.524	0.792	**1.0**	**1.0**	**0.393**	0.383	0.355
8	0.333	**0.443**	0.424	**0.465**	0.441	0.457	0.6	**0.916**	0.831	**0.379**	0.29	0.315
9	0.081	**0.146**	0.145	**0.121**	0.069	0.069	0.22	**0.705**	0.698	**0.084**	0.036	0.036
12	0.801	0.801	0.801	0.783	0.783	0.783	1.0	1.0	1.0	0.644	0.644	0.644
13	0.771	**0.847**	0.368	0.769	**0.855**	0.23	0.833	0.847	**1.0**	0.714	**0.865**	0.141
14	0.767	**0.845**	0.837	0.77	**0.84**	0.834	**0.772**	0.74	0.745	0.767	**0.97**	0.946

`Friday2017-Afternoon-PortScan`, and `Friday2017-Afternoon-DDos`). We keep these author-provided splits and treat each file as a separate evaluation context (In the revised dataset version, these files are consolidated into a single CSV file per calendar day). Day identifiers and names follow Table 3.3; thus, all results in Table 3.2 reference those IDs. 10 out of 17 days are not reported because the detection performance obtained for these days using all the configuration techniques is higher than 0.99. Hence, during these days, MtL is as efficient as HPO or default.

On day 12, the configurations returned by the three methods give the same level of performance. Their values are 0.801 (MCC), 0.783 (f1-score), 1.0 (precision) and 0.644 (recall). Note that even if the inferred configuration differs (HPs values), it means that attack and normal traffic flows are classified similarly by the different RFs generated.

On the days 2, 3, 8, 9 and 14 and in comparison to the default, HPO improves the MCC with an average percentage increase of 27.79% (with values ranging from 5.54% for day 2 to 80.24% for day 9). This is an expected result since HPO's objective is to maximize this metric. MtL manages to improve it also in a similar order of magnitude with an average of 25.28% (with values ranging from 4.47% for day 2 to 79.01% for day 9). Although the precision was not taken into account in the HPO's objective function, it increases with an average of 62.03% (with an improvement of up to 220% on day 9). For MtL, the improvement is 58.69%. Regarding the recall, HPO and MtL bring a gain at day 14 where it increases from 0.767 to 0.97 for HPO and to 0.946 for MtL.

MtL is the least efficient on day 13. Actually, HPO manages to improve the NIDS's performance while MtL performs poorly on three of the four metrics. HPO increases the MCC's default configuration from 0.771 to 0.847, the f1-score from 0.769 to 0.855 and the recall from 0.714 to 0.865. HPO increases the precision from 0.833 to 0.847. MtL outperforms HPO on precision ($= 1$) but at the cost of a low recall. In that case the NIDS misses a lot of attacks but never raises a false positive alarm. As seen in Table 3.3, day 13 is the only day combining both a relatively low number of flows and a high imbalance between normal and attack traffic. This might have a major impact on the values of constructed meta-features and so on the learning phase of the meta-model.

Generally, HPO and MtL improve the performance on almost every day in comparison with the default configuration. Over the five relevant days (2, 3, 8, 9 and 14), MtL is at the same level of HPO performance. The difference is minimal with only 2.62% gain in MCC and 3.37% gain in precision for HPO compared to MtL. Thus, MtL manages to learn well from previous optimal configurations (based on the 17 other days) with a little degradation (less than 3.40%) in NIDS performance in comparison to HPO.

Table 3.3: List of the datasets

day id:Name	Attack	% Benign	% Attack	# flows
0:bot_ 02-02-2018	Botnet	82.37	17.63	819904
1:bruteforce-ftp-ssh_ 14-02-2018	Bruteforce	85.97	14.03	671126
2:bruteforce-web-xss-sql-injection_ 22-02-2018	Web attack	99.96	0.04	901548
3:bruteforce-web-xss-sql-injection_ 23-02-2018	Web attack	99.94	0.06	905177
4:ddos-loic-http-loic-udp_ 20-02-2018	DDoS	90.52	9.80	6073495
5:ddos-loic-udp_hoic_ 21-02-2018	DDoS	64.26	35.74	561405
6:dos-goldeneye-slowloris_ 15-02-2018	DoS	94.14	5.86	876946
7:dos-slowhttp-hulk_ 16-02-2018	DoS	75.45	24.55	591901
8:infiltration_ 01-03-2018	Infiltration	72.31	27.69	279818
9:infiltration_ 28-02-2018	Infiltration	90.31	9.69	347198
10:Friday2017-Afternoon-DDos	DDoS	57.38	42.62	223082
11:Friday2017-Afternoon-PortScan	Infiltration	57.57	42.43	213777
12:Friday2017-Morning-Bot	Botnet	98.94	1.06	184044
13:Thursday2017-Afternoon-Infilteration	Infiltration	99.98	0.02	252790
14:Thursday2017-Morning-WebAttacks	Web attack	98.69	1.31	164173
15:Tuesday2017-WorkingHours	Bruteforce	97.82	2.18	421626
16:Wednesday2017-workingHours	DoS	68.26	31.74	610445

3.3.2 Resource usage at Deployment Stage

In this section, we do not consider the time needed to learn the meta-model, since it can be done in advance. In our case, we have a new dataset every day, and after a number of uses of HPO (16 in this example), we except to be able to have the new day's configuration (the 17th) quickly.

The time MtL takes to infer a configuration is equivalent to the meta-model's inference time (in our case a RFr) but, unlike HPO, it requires first to extract the meta-features for the new dataset. We thus compare the time to generate a configuration for MtL and HPO by measuring the number of HPO iterations.

For MtL, we converted the time to extract the meta-features in an equivalent number of HPO iterations. In conducting our experiments, we found that the meta-model's inference time is between 7 and 9 secondes. For simplicity, we consider this time to be equal to one HPO iteration, which can be considered as the worst case since one iteration can be up to 10 seconds.

In practice, when using HPO, knowing the minimum number of iterations to achieve the maximum NIDS performance cannot be predicted. However, for a fair analysis, we consider here the case of the number of iterations performed by HPO to reach the same performance as MtL.

As shown in Figure 3.4, MtL is executed in quasi-constant number of iterations between 5 and 8 to generate a configuration. So, even for days 1, 5 and 8 where HPO is slightly better (about 3 to 5 iterations), the practical impact for the NIDS is negligible, *i.e* using MtL will take 30 or 50 seconds longer than HPO at most in this days.

In contrast, HPO takes significantly more iterations to find an equivalent configuration in terms of detection performance for the other days. This value highly varies depending on the day with extreme values above 100. It would be thus difficult to deduce the number of iterations in advance. Globally, MtL is faster with a factor of 9 in average over all days.

After MtL, the model still needs to be trained with the configuration and used on the test set to predict whether the flows are malicious or not. This is not the case for HPO because training

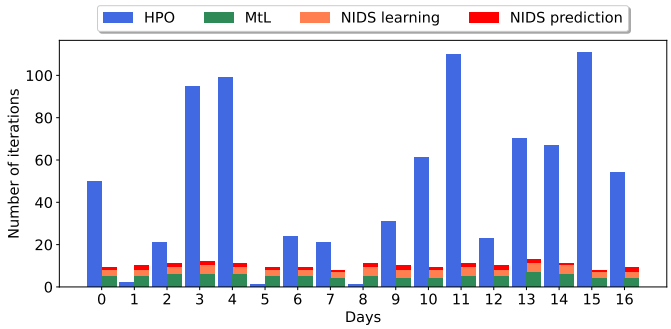


Figure 3.4: Configuration generation, learning and prediction time of the NIDS

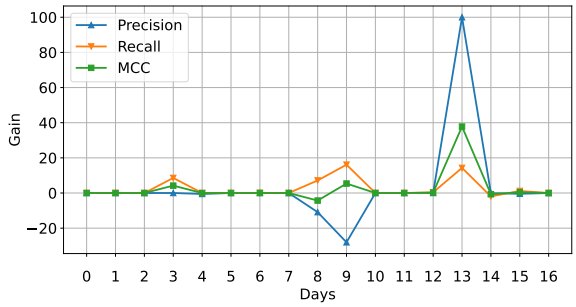


Figure 3.5: Relative performance of MtL calculated as a ratio between (MtL-HPO) and HPO (an infinite value due to division by zero is marked as 100% on day 13)

and prediction are done (mutiple times) in parallel with the search for optimal HPs. Figure 3.4 also includes the learning and prediction time of the MtL-based NIDS. The learning time is fairly constant and represents between 3 and 5 iterations. For the prediction time, it is between 1 and 2 iterations. Hence, in a normal scenario where the number of iterations of HPO is not known in advance, building a NIDS and predicting the nature of new flows takes 300 iterations for HPO while MtL takes between 9 and 15 iterations.

Figure 3.5 reports for each day the relative performance obtained by MtL by limiting the number of iterations of HPO to the time needed by MtL to infer a configuration. It is calculated as $(p(MtL) - p(HPO))/p(HPO)$ where p is a generic evaluation metric. For a large majority of days, differences are minor. On one hand, we notice a negative Precision gain of -10.89% for day 8 and -28.05% for day 9. On the other hand, MtL and HPO have a similar Precision in Table 3.2 because HPO continues to iterate to maximize MCC. Actually, the HPO technique we used is not multi-objective and so sacrifices the Precision in favor of a better MCC leading to results of Table 3.2. On day 3, MtL improves the Recall by 8.57% and the MCC by 4.19%. Hence During the 13th day, HPO obtained at iteration 7 a Precision close to 0 while MtL reached a value close to 1.

Regarding memory usage, the use of the whole-day dataset is necessary for the implementation of the HPO solution. The best case is day 11 with a size of 18.79 MB, and the worst case is day 4 with 837.33 MB. Since MtL only uses meta-features, less than 36KB is used for any day.

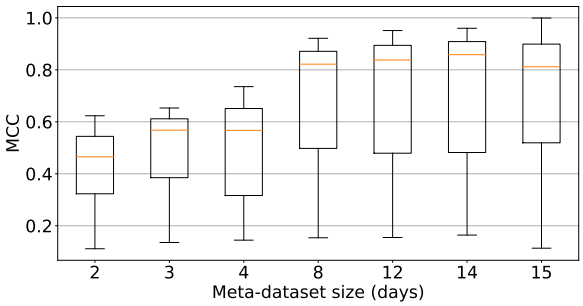


Figure 3.6: Detection performance according to the meta-dataset size (#days)

3.3.3 Meta-dataset size

A major assumption of meta-learning is the availability of a representative enough meta-dataset. Previously, we always considered 16 days of data. Although each day is different (in terms of network context and attack type), we consider in this section a more restrictive scenario with less prior experience, i.e., fewer days to build the meta-dataset. For each number of days and for each possible combination without repetition, a configuration is predicted for every single day not included in the combination. As an example, for the meta-dataset consisting of 2 days, we have 136 possible combinations, all applied on the remaining 15 days. This implies 2040 different configurations. The results are reported in Figure 3.6.

As expected, the more days used to infer a configuration, the better the NIDS performance. On very close numbers of days (3 and 4 for example or 14 and 15), there is no obvious difference in performance. The performance of the NIDS starts to stabilize with 8 days (with a mean equal to 0.82), where we can get almost similar performance to 12, 14 and 15 days. Hence, computational power can be also limited during the meta-modeling phase by using a partial meta-dataset. In other words, building the meta-target vector would no longer take 4800 iterations but 2400 for the same MtL performance.

3.4 Empirical Evaluation of the Meta-Learning Framework for AD NIDS Configuration

We evaluate our auto-configuration solution in terms of three key aspects: robustness, anomaly detection performance, and configuration generation time. In this context, robustness refers to the performance of the underlying anomaly detection model based on its hyperparameters ($\boldsymbol{\lambda}$) alone, independent of the final decision threshold τ. It is a measure of how well the model separates benign from malicious traffic across all possible thresholds. In contrast, anomaly detection performance measures the effectiveness at a specific, chosen threshold. We compare our method to the optimal configuration found using Bayesian Optimization (BO), an ideal but impractical scenario that requires attack traces, and to four other less time-consuming alternative methods. To configure each daily dataset D_i , our method uses data from the previous 13 days to build the meta-model, applying this process in a rotating manner to yield 14 independent AD IDS configurations.

3.4.1 Robustness

To clearly present our results, we first compare our solution with three alternative methods in a first section and follow with a final comparison. AD NIDS robustness is assessed thanks to the AUC ROC metric, denoted as p_{robust}.

3.4.1.1 Default, single overall and under-sampling based configurations

The first alternative is to use a default configuration, $\boldsymbol{\lambda}_{default}$, provided by the libraries implementing anomaly detection algorithms. This approach represents the most basic option available, requiring no technical intervention and offering immediate implementation. In our scenario, we rely on the default configuration offered by the isotree library.

The second alternative is a single overall configuration serving as a *one-fits-all* configuration. Applying BO to a set of fairly heterogeneous network contexts (a global dataset) aims at finding a configuration suitable to any network security dataset. The global dataset $\mathbf{D}$ is created as:

$$\mathbf{D} = \bigcup_{i=0}^{13} D_i \tag{1}$$

$\mathbf{D}$ includes all the 14 days of network traffic from the two datasets. The resulting optimal single overall configuration is $\boldsymbol{\lambda}^*_{\text{overall}} = \text{BO}(\mathbf{D})$, which can be computed offline.

The final option, under-sampling, involves a daily application of BO using less data, which addresses the time-consuming nature of this method. Each day i is thus sampled at 10% to significantly accelerate BO while ensuring a sufficient number of flow. Under-sampling is conducted individually for each traffic label (type), leading to the following configuration:

$$\boldsymbol{\lambda}^*_{\text{sample}_i} = \text{BO}(\text{sample}(D_i, 0.1)),\ \forall i \in [0, 13]$$

In Figure 3.7, each day i is evaluated by applying these derived configurations. We also include our meta-learning (MtL) results for comparison:

- The default configuration (orange): $p_{\text{robust}}(D_i, \boldsymbol{\lambda}_{\text{default}})$ is calculated.
- The single overall configuration (green): $\boldsymbol{\lambda}^*_{\text{overall}}$ is applied daily, and $p_{\text{robust}}(D_i, \boldsymbol{\lambda}^*_{\text{overall}})$ is calculated.
- The under-sampling configuration (blue): $p_{\text{robust}}(D_i, \boldsymbol{\lambda}^*_{\text{sample_}i})$ is computed by applying $\boldsymbol{\lambda}^*_{\text{sample_}i}$ on the corresponding day i.
- Our meta-learning(mtl)-based configuration (dark blue): $p_{\text{robust}}(D_i, \boldsymbol{\lambda}_{\text{mtl_}i})$ is calculated by applying the MtL configuration $\boldsymbol{\lambda}_{\text{mtl_}i}$ returned by the meta-model on day i.

These curves are benchmarked against the baseline, $p_{\text{robust}}(D_i, \boldsymbol{\lambda}^*_i)$, which represents the ideal but unrealistic scenario with a daily optimal configuration using BO, depicted in red in Figure 3.7.

The overall configuration applied on the full dataset yield to a ROC AUC of 0.95 and represented as a dashed line in the same Figure. At a first glance, this result seems excellent but this is due to a few days with many samples where the overall configuration is well suited. This is highlighted on days 0 and 4 where daily performance using $\boldsymbol{\lambda}^*_{\text{overall}}$ are even higher than 0.95 whereas most of other values (10 days) are generally close or significantly below 0.8. Actually, The average value across all days is just 0.77, with a notable standard deviation of 0.22, underlining the efficiency disparity of the overall configuration across all days.

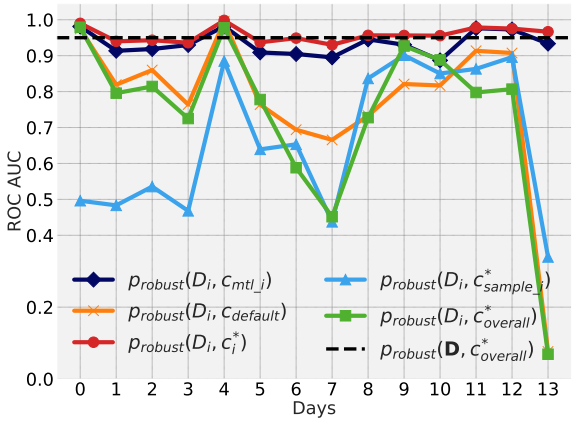


Figure 3.7: Comparison of different configurations: default, single overall, under-sampling, MtL, optimal

In contrast, a daily optimal configuration allows to reconfigure the AD IDS to maintain its performance optimal. It is highlighted with the AUC ROC always higher than 0.93 for $p_{\text{robust}}(D_i, \boldsymbol{\lambda}_i^*)$. Hence, while a configuration like $\boldsymbol{\lambda}^*_{\text{overall}}$ may perform well from an aggregated perspective, it doesn't necessarily ensure effectiveness in every context.

By comparison, the default configuration (in orange) only matches the optimal value on days 0 and 4, exhibiting significantly lower values on other days, especially day 13 with the AUC nearing 0.

Even the under-sampling technique does not provided satisfactory results as highlighted by the curve $p_{\text{robust}}(D_i, \boldsymbol{\lambda}^*_{\text{sample_}i})$, especially on days 0 to 3, 7, and 13, where the ROC AUC is notably below 0.6. Actually, less data is used and so the configuration might be less adapted depending on the heterogeneity of the day's dataset. While under-sampling could have been a simple but time-efficient alternative to full BO, it is not reliable.

Our technique ($p_{\text{robust}}(D_i, \boldsymbol{\lambda}_{\text{mtl_}i})$ in dark blue) is able to approximate the optimal (in red) with very low robustness degradation. The most notable difference is observed for day 10 where MTL robustness is 0.89 unlike the optimal BO value of 0.96. In general, For MtL, the ROC AUC values ranges from 0.89 to 0.98 and the average difference with the optimal is negligible (0.23%), clearly illustrating the effectiveness of our technique in approximating a near-optimal configuration.

On Day 13, detailed investigation showed that while alternative configurations performed well in certain folds (AUC ROC near 1 for some days), they underperformed in others (AUC ROC near 0). This variability implies Day 13's benign traffic is quite heterogeneous, making configuration more difficult and contributing to the day's poor obtained results for most alternative methods other than MtL.

3.4.1.2 Configuration Transferability

A final alternative that could be implemented without using BO is transferring configurations across days, which maintains simplicity from a computational standpoint.

For example, using daily optimized configurations from D0-D12, we can compute $p_{\text{robust}}(D_{13}, \boldsymbol{\lambda}_i^*)$ for each $\boldsymbol{\lambda}_0^*, \ldots, \boldsymbol{\lambda}_{12}^*$ to assess their direct transferability to the 13th dataset. Overall results are then obtained by rotating the test and compared against the baseline (i.e the daily optimal configurations). In this way, similar to the default and overall methods, reconfiguration does not

require a validation set with attacks.

Formally, this is defined as:

$$R_i = \{PPD(i,j) \,|\, \forall j \in [0,13], i \neq j\},\ \forall i \in [0,13]$$

where R_i denotes the distribution of values obtained from $PPD(i,j)$. The Performance Decrease Percentage (PPD) for day i is determined by comparing its optimal AD IDS performance against the performance achieved with day j's optimal configuration:

$$PPD(i,j) = \frac{p_{\text{robust}}(D_i, \boldsymbol{\lambda}_i^*) - p_{\text{robust}}(D_i, \boldsymbol{\lambda}_j^*)}{p_{\text{robust}}(D_i, \boldsymbol{\lambda}_i^*)} \times 100$$

The distribution R_i for each day is shown as a boxplot in Figure 3.8.

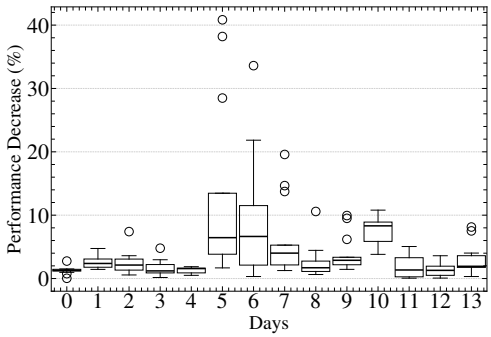


Figure 3.8: Daily optimal configurations transferability

Days 0-4, 8, 9, 11-13 show median PPD values under 5%, suggesting generally feasible transferability, despite some outliers nearly reaching or exceeding 10% degradation. In contrast, days 5-7, and 10 exhibit higher scattering, especially days 5 and 6 with values hitting 35-40%. Their wider box plots indicate large performance degradation.

Although day 7 has a compact box, it includes outliers of up to 20%, unlike day 10, which results in a higher median. Transferability is thus inefficient for these days. In conclusion, transferability is practical only on certain days, while on others, it's either impossible or requires predicting the exact day configuration to transfer. Compared to MtL, the degradations relative to the optimal are much larger and highly variable from one day to another. The worst degradation value for MtL is on day 10, with a negligible PPD of 0.072%. This highlights the challenges in sustaining attack detection performance through inter-day optimal configuration transfers, and invalidates meantime the possibility to have only a few good candidate configurations which allows to maintain AD IDS performance.

3.4.1.3 Per-attack Analysis

Despite previous findings on inconsistent transferability across days, we now investigate cases where the types of attacks to be detected can be anticipated. We acknowledge that this is an ideal scenario; one might expect that a configuration could be transferred easily across days with similar attacks, because only the network topology and the number of hosts differ between 2017 and 2018. The goal of this section is to verify this hypothesis.

The list $\mathcal{A}$ consists of 18 attacks common to IDS2017 and IDS2018. Each attack a_k in $\mathcal{A}$ ($\forall\, k \in [0,17]$) is individually represented in $\mathcal{D}_{a_k}$ for both the 2017 and 2018 versions and is combined with identical benign traffic, randomly sourced from IDS2017. In this section, we

present only the assessment of transferability of the optimal configurations from IDS2017 to IDS2018, but similar findings with the reverse scenario have been observed.

With BO, for each $\mathcal{D}_{a_k}^{j}$ ($j \in \{2017, 2018\}$) , we determine the respective $\boldsymbol{\lambda}_{a_k}^{\star 2017}$ and $\boldsymbol{\lambda}_{a_k}^{\star 2018}$ for all $k \in [0, 17]$, followed by computing:

- $p_{\text{robust}}(\mathcal{D}_{a_k}^{2018}, \boldsymbol{\lambda}_{a_k}^{\star 2018})$: the robustness when applying an *optimal* configuration, i.e., specific to the attack and the year (baseline),
- $p_{\text{robust}}(\mathcal{D}_{a_k}^{2018}, \boldsymbol{\lambda}_{a_k}^{\star 2017})$: the robustness when applying an *attack-based* configuration, i.e., specific to the attack but on the previous year.

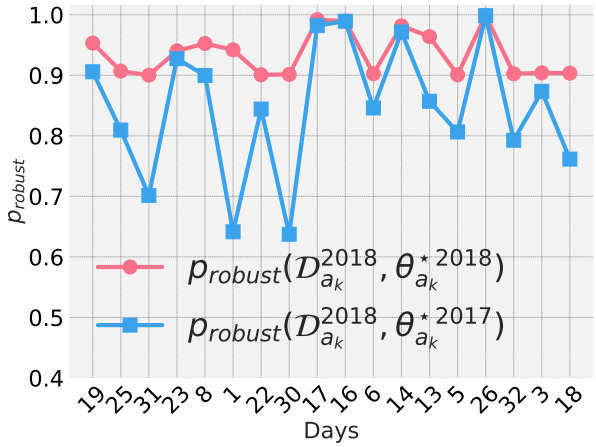


Figure 3.9: Transferability of optimal configurations across attacks.

In Fig. 3.9, attacks are indexed on the x-axis according to the IDs from Fig. 3.1. As highlighted, the robustness varies widely among attacks when the configuration is transferred from the previous year (blue). For example, attacks 23, 17, 16, 14, and 26 achieve close values to the baseline (red), minor discrepancies are seen in attacks 19, 8, 22, 6, and 3, but the variance is more significant in attacks 31, 1, and 30.

Even in ideal scenarios where the attack type is known in advance and only the network environment changes, reconfiguring the ML algorithm may still be necessary.

As a reminder, the number of flows is largely higher in IDS2018 than in IDS2017. A higher volume means many additional benign flows, and this affects the Isolation Forest in two ways. First, this reduces the effective subsampling rate, so each tree sees a smaller fraction of the available traffic. Second, deeper trees can be grown because more points are available to split, which increases computational cost and can shift the detector's sensitivity. Both effects imply that a configuration tuned on a light-traffic day may be poorly matched to a heavy-traffic day even when the mix of protocols remains similar. This is confirmed by our experiment.

3.4.1.4 Outcomes

We compared the robusstness of our configuration method, which doesn't require a validation set, with five others: optimal and undersampling—both not adhering to this criterion—and single overall, default, and transferability, all meeting it.

Initial findings from these experiments highlight the AD IDS's high sensitivity to its configuration; both default and overall configurations clearly illustrate the advantages of reconfiguring

and adapting the AD IDS to the day's specific context. Undersampling shows us that reducing the quantity of data and thus altering the distribution negatively affects the configuration of the AD IDS, even when using attacks.

The optimal approach achieves almost perfect ROC AUC scores by daily adapting the configuration using BO, but it requires attack samples. Our method manages to maintain AD IDS performance at comparable levels without requiring these samples. The other alternative methods that also avoid attacks often perform inconsistently and may lead to significant performance drops on specific days. Even though these methods may be tempting due to their simplicity, variability in results makes it challenging to choose the best one. For instance, on day 13, the transferability method is effective, unlike the overall and undersampling methods. Conversely, on day 5, transferability struggles while the overall method proves more effective. In contrast, our method achieves much greater stability across all days. Given the unsatisfactory results obtained with these methods, we will solely focus on comparing our method with the optimal approach in the following sections.

3.4.2 Anomaly Detection

In this section, we assess our method's anomaly detection performance by determining the classification performance at a specific threshold, denoted by p_{classif}. This evaluation examines our solution's ability to directly infer the threshold value. p_{classif} is evaluated using the F1-score and precision. While not presented, it is worth noticing that recall is always above 0.97.

For the optimal baseline scenario that assumes attack samples are known, τ_i^* is determined by maximizing the F1 score after BO has been applied to the HPs. As a reminder, τ is treated as an HP to be inferred with our solution.

In terms of the F1 score shown in Figure 3.10, the optimal configuration using BO ($p_{\text{classif}}(D_i, \boldsymbol{\lambda}_i^*, \tau_i^*)$) demonstrates high performance, with values ranging from 0.94 to 1. Although our method does not rely on a validation set with attack samples to infer τ, the F1 score is noticeably lower only on days 2 and 5-8. The worst result occurs on day 6, with 0.84 compared to 0.98 obtained with the optimal configuration.

Figure 3.11 highlights similar findings for precision, with MtL values generally lower than the optimal and exhibiting greater variability.

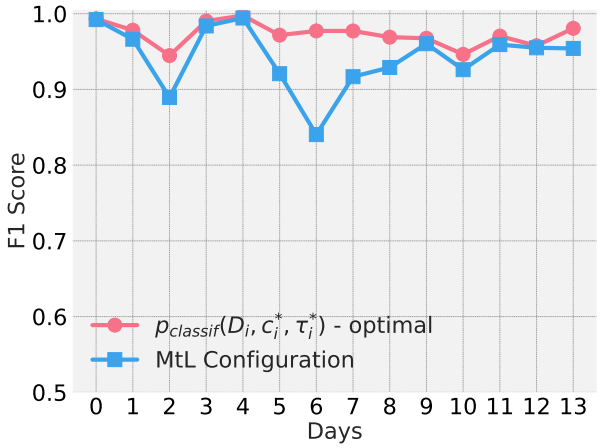


Figure 3.10: F1 score with a daily configuration

Generally, our approach is almost equivalent to BO for both the F1-score and precision over

8 days (out of 14). It is slightly less accurate on 2 days (days 8 and 13) and leads to noticeable degradation on only 4 days (2, 5, 6, and 7).

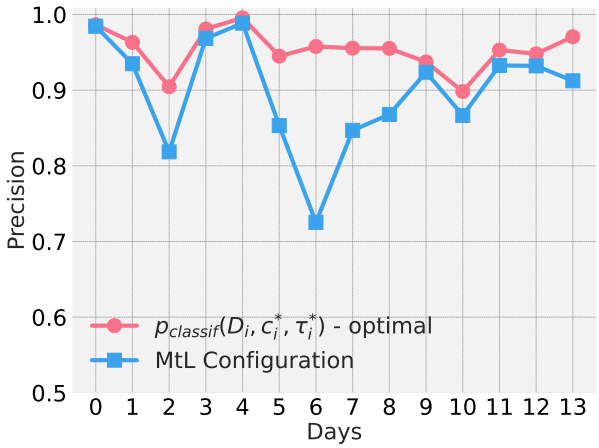


Figure 3.11: Precision with a daily configuration

Setting τ is challenging because the optimal threshold τ_i^* tends to vary each day. For instance, τ_0^* is 0.31, while τ_4^* is 0.10. This variability makes it difficult to develop an efficient ML model capable of predicting it without knowledge of the attack traffic characteristics. This suggests that even if the inferred τ_i (MtL) differs from τ_i^* (optimal), MtL compensates by optimizing all HPs jointly, a task challenging to perform manually. Hence, configuration is an interplay of all HPs, whereas most work considers setting the threshold and the HPs of the ML algorithm in separate phases.

3.4.3 Configuration Generation Time

We also assessed, on a daily basis, the time it takes for MtL to generate configurations compared to BO.

For MtL, the time for learning the meta-model is completed offline, and thus the generation time for MtL is the time taken to infer c_i, including the extraction of the dataset's characteristics. Once done, the AD IDS requires training with the chosen configuration and application on the test set to identify malicious flows. This process contrasts with BO, where training and prediction run concurrently. Hence, for a fair comparison, we also include the time taken by MtL for training and testing the model in the online phase.

In Figure 3.12, our solution always returns a configuration in less than 1200 seconds, with extremely fast results on days 9 and 10 (less than 1 second). Conversely, determining $\boldsymbol{\lambda}_i^*$ requires considerably more time, with durations varying from day to day and occasionally exceeding 100000 seconds. Overall, MtL is faster by about a factor of 160 across all days.

3.4.4 Meta-learning Relative Performance

Figure 3.13 highlights the daily relative performance (gain) of MtL for p_{robust} (ROC AUC) and $p_{classif}$ (F1 score) compared to the performance achieved by BO when the number of BO iterations is reduced to match MtL's configuration generation time. Even without considering attacks for configuration, MtL performs as well as BO. Specifically, the gain does not show any major negative values and is even better in certain cases. In terms of p_{robust}, we note that on

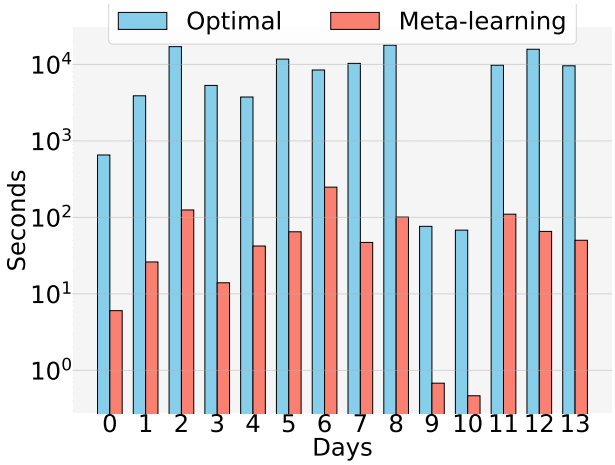


Figure 3.12: Configuration generation time.

5 out of 14 days, the gain is nearly or exceeds 0.25, while on 2 out of 14 days, it surpasses 0.5. For $p_{classif}$, although the gains are typically lower, it's worth noting that for 4 out of 14 days, they approach or surpass 0.25. This demonstrates the superiority of our approach over BO at an equivalent computational cost and without requiring attack samples.

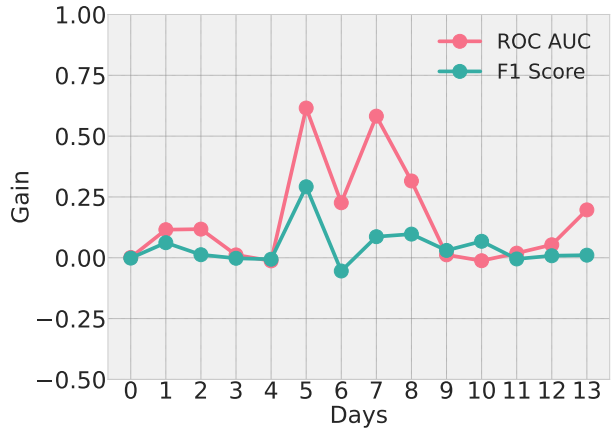


Figure 3.13: Performance gain of semi-supervised `MtL` computed as the difference between the performance of `MtL` and BO.

3.5 Conclusion

The empirical results presented in this chapter demonstrate that our meta-learning framework provides an effective solution to the challenge of NIDS autoconfiguration, successfully addressing the critical limitations of conventional HPO. Across both the supervised classification and unsupervised anomaly detection settings, the framework consistently infers configurations that achieve detection performance substantially close to the theoretical optimum found by a full Bayesian Optimization search. By effectively transferring knowledge from previous HPO tasks, the approach dramatically reduces the time and computational resources required for reconfiguration, achieving a speedup of up to two orders of magnitude. Furthermore, the framework's

ability to jointly infer hyperparameters and the decision threshold from benign-only data makes it a uniquely practical solution for deploying and maintaining anomaly detection systems in real-world environments where labeled attack data is scarce.

The study relies on two related testbeds (IDS2017/2018) and a flow-based feature space. While we evaluate both label- and score-based detectors, the core mechanism (learn a fingerprint, then predict a configuration) does not depend on flows, RF, or IF. Other granularities, such as packet-level or graph-based NIDS, could adopt this framework by using richer descriptors (e.g., temporal entropy, inter-arrival time statistics for packets; small-subgraph "motif" counts such as triangles or stars for graphs) and use the same mapping. Representation learning for descriptors (e.g., using compact autoencoders) is a promising extension when moving to deep sequence models. The reported gains also depend on the oracle's specific objective (MCC for classification; AUC-ROC/AUPRC/F1 for anomaly detection); different operational goals, such as prioritizing recall over precision, would change the performance trade-offs and likely require retraining the meta-model. Exploring these extensions constitutes promising future work.

Furthermore, the framework faces challenges regarding generalizability across disparate domains. Because the current meta-model is trained on data from specific environments, such as enterprise IT networks, its capacity to generalize to radically different architectures, including Industrial Control Systems or IoT networks, is not guaranteed. Transferring this predictive capability without retraining on historical data from those new domains constitutes an important open area for research. Exploring these extensions and boundaries is a critical next step for this methodology.

In sum, the results are best read not as MtL replaces HPO, but as MtL offsets the computational cost of HPO. It captures the reusable part of configuration knowledge in a detector-agnostic way, delivering a reliable one-shot configuration, reserving the need for a complete HPO usage only for network contexts whose meta-features fall outside the domain of the meta-model's training experience; quantifying this domain shift to automatically determine when to revert to a full HPO search is an important direction for future work.

The results confirm that our approach successfully addresses the critical limitations of conventional HPO. It dramatically reduces the time and computational resources required for reconfiguration, achieving a speedup of up to two orders of magnitude while maintaining detection performance close to the theoretical optimum. Furthermore, the framework's ability to jointly infer hyperparameters and the decision threshold from benign-only data makes it a uniquely practical solution for deploying and maintaining anomaly detection systems in real-world environments where labeled attack data is scarce.

Part II

Test-Set Assessment and Traffic Augmentation for ML-Based Network Intrusion

Chapter 4

Data Augmentation for NIDS

Contents

4.1 Introduction 79
4.2 Motivation for Data Augmentation in NIDS 80
4.2.1 Structural and Design Limitations 80
4.2.2 Content-Level Pitfalls and Artifacts 81
4.3 Data Augmentation for NIDS 83
4.3.1 Performance-Oriented Data Augmentation Techniques 83
4.3.2 Realism-Oriented Traffic Augmentation 85
4.3.3 Test-Set Augmentation for NIDS Robustness Testing 87
4.4 Conclusion 93

4.1 Introduction

The previous part established that for a ML-based NIDS to remain effective, it must be fundamentally adaptive. We identified a critical gap in current practice, arguing that adaptation must extend beyond only retraining to also consider the configuration of the hyperparameters as part of this adaptation. Building on these findings, we now turn to a second limitation in NIDS research: evaluation is usually performed on a fixed benchmark test set, which may not sufficiently challenge the detector or reflect the difficulty of real operational traffic.

ML-based NIDS are trained, compared, and selected largely on public benchmarks. These datasets make empirical research scalable and reproducible, yet they only approximate the conditions encountered in live networks [RWS+19c]. Subsequent analyses show that many benchmarks contain structural limitations (e.g., class imbalance, single-vantage captures, outdated protocol mixes) and data-level artifacts (e.g., mislabeling, duplication) [E+21, TBLG09]. Consequently, excellent scores on a standard split (i.e., the usual train, validation, and test partition drawn from the same dataset) do not always translate into reliable performance after deployment [CMB24]. This part asks how data augmentation can help bridge that gap.

We define augmentation broadly as any method that modifies existing traffic or creates new data to enrich the dataset. In practice, three goals emerge. First, *performance-oriented augmentation* seeks to expand and re-balance training data, primarily to address the severe class imbalance often found in NIDS datasets. This approach, which include statistical techniques like

SMOTE [CBHK02] or simple, rule-based transformations, aim to improve standard classification metrics (e.g., F1-score) and often operate at the feature level without a strict need to model packet-level detail. Second, *realism-preserving augmentation* moves beyond statistical balancing and aims to synthesize traffic that is semantically and structurally indistinguishable from live networks. This approach is critical for preventing models from learning non-transferable dataset artifacts. It operates by respecting protocol grammar, modeling stateful dependencies (such as TCP sessions), and capturing realistic timing patterns, often requiring complex generative models like GANs or VAEs. Third, *test-set augmentation for robustness* uses augmentation not for training, but for evaluation. This goal involves using challenging but label-preserving cases to systematically assess the model's robustness. This approach mainly targets the *robustness* of the model's decision boundary, finding inputs that cause misclassification and thereby exposing limitations that ordinary test sets may miss.

In this chapter, we first motivate the need for data augmentation in NIDS (§4.2) and review dataset quality, separating structural limitations (§4.2.1) from data-level pitfalls and artifacts (§4.2.2). We then survey augmentation methods (§4.3), covering three complementary goals: performance-oriented augmentation (§4.3.1), realism-preserving augmentation (§4.3.2), and test-set augmentation for robustness (§4.3.3), including the protocol-aware controls required for generating valid synthetic traffic. We conclude with a synthesis of the main limitations and open gaps (§4.4).

4.2 Motivation for Data Augmentation in NIDS

Public NIDS datasets have become the default reference points for ML and DL NIDS research. Their popularity derives from the convenience of ready-made traffic representations and permissive licenses, which enable hundreds of papers to run empirical studies with minimal data-engineering effort. These benchmarks include a variety of datasets, from older ones created in controlled laboratory environments (e.g., the DARPA/Lincoln Lab evaluations and KDD Cup '99 [LHF+00, SFL+99]) to more recent ones captured on test networks or internet backbones (e.g., UNSW-NB15 [MS15], IDS2018 and IDS2017 [SHLG18b], CTU-13 [GGSZ14], and UGR'16 [MFCMC+18]).

As these benchmarks became more widely used, the research community began to critically assess their quality and relevance. This revealed that their practical value is often compromised by significant flaws, a finding that motivated a growing interest in data augmentation as a solution. To establish the rationale for this focus, this section provides a review of the documented shortcomings in public NIDS datasets. This review is organized into two primary aspects of data quality: structural and design limitations and content-level pitfalls and artifacts.

4.2.1 Structural and Design Limitations

At a structural level, NIDS benchmarks are constrained by high-level design choices that limit how well they mirror real-world operational conditions. Severe class imbalance is common: in IDS2017, for example, there are 231,073 DoS-Hulk flows but only 11 Heartbleed flows, so rare, yet high-impact attacks are underrepresented and easy to miss in evaluation. Traffic mixes can be unrepresentative of today's network realities: In IDS2018, the authors of the dataset note that the benign profiles are dominated by web protocols and that 'the majority of traffic is HTTP and HTTPS'; importantly, a non-trivial portion is plaintext HTTP. By contrast, browser telemetry and Google's HTTPS Transparency Report [Goo25] indicate that the vast majority of page loads on today's web occur over HTTPS, so corpora with substantial plaintext HTTP are less

representative of modern conditions. Additionally, most datasets reflect a single vantage point: UGR' 16 aggregates NetFlow from one Spanish ISP, and MAWILab labels 15-minute captures from a single trans-Pacific backbone link, perspectives that miss multi-tenant and multi-site effects seen in deployment.

While specialized datasets for areas like IoT (e.g., IoT-23 [GPE20]) or ISP traffic (e.g., UGR'16) exist, a significant gap remains for more complex, heterogeneous environments. There is a notable lack of public datasets for cloud-scale or multi-tenant systems, a scarcity that broad surveys attribute to prohibitive privacy and legal constraints [RWS+19b]. Similarly, while benchmarks for Software-Defined Networks (SDN) exist, they are often narrowly focused; the InSDN dataset [ELKJ20], for example, targets specific control-plane attacks rather than general enterprise traffic. The same is true for Industrial Control Systems (ICS), where datasets like SWaT and WADI are available but plant-specific [MT16, APM17], failing to capture the diverse traffic between enterprise and industrial networks that practitioners must defend.

4.2.2 Content-Level Pitfalls and Artifacts

At the data level, the integrity of these benchmarks is further compromised by a range of internal flaws, errors, and artifacts. A series of separate reviews now shows that these datasets, though widely regarded as relevant benchmarks, embed artifacts that inflate reported performance and mask real detection weaknesses. Catillo *et al.* call this an *academic hype loop*: a cycle in which progress on synthetic benchmarks substitutes for genuine advances, even though the results rarely transfer to operational networks [CPV23]. Ring *et al.* reach a similar conclusion in their comprehensive survey of 30+ datasets [RWS+19c]. To clarify the specific mechanics of this problem, the following analysis categorizes these content-level flaws into three distinct areas: incorrect ground-truth labeling, widespread data artifacts (or "data smells"), and the use of outdated attack scenarios.

Incorrect Ground-Truth Labeling

Correct ground truth is essential for supervised learning, yet multiple studies reveal significant mis-labeling. Lanvin *et al.* find packet-reordering artifacts in *IDS2017* that invert source and destination fields and corrupt 5–8 %of labels; relabeling lowers recall on port-scan traffic by roughly 20 percentage points [LGH+22]. Engelen *et al.* link a further 6–8 %of label corruption in both *IDS2017* and *IDS2018* to undocumented filtering rules and faulty attack orchestration [ELL+22]. Because many studies train and evaluate on these mislabeled flows without verification, their reported precision and recall do not accurately reflect a detector's true effectiveness.

Widespread Data Artifacts and Smells

A second line of evidence concerns redundancy introduced by the flow-export tools themselves. Engelen *et al.* show that a bug in CICFlowMeter [Can17] can break a single TCP connection into many almost identical fragments, so that one session is recorded as dozens of duplicate flows [ELL+22]. Flood *et al.* generalize this observation across seven datasets, introducing the notion of *data-design smells* to describe six recurrent issues [FEAD24a]. They catalogue these smells as:

- **Poor Data Diversity:** Homogeneity from automated tooling with fixed configurations, which rewards overfitting and fails to test generalization.

- **Highly Dependent Features:** Features with high discriminative power that are unrelated to the attack mechanism, such as protocol or TTL values that inadvertently fingerprint operating systems.

- **Unclear Ground Truth:** Ambiguous labeling logic, where benign background services are incorrectly or inexplicably labeled as attacks.

- **Wrong Labels:** Inaccurate ground truth due to faulty attack orchestration or data processing errors.

- **Traffic Collapse:** A total lack of diversity from configuration errors, such as failed attacks against closed ports, which "collapses" traffic statistics into a trivial-to-detect pattern.

- **Artificial Diversity:** Spurious, unrealistic variability caused by network capture failures or high retransmission rates.

Their analysis shows these smells are pervasive across all seven datasets. The impact is significant: in *UNSW-NB15*, leaked features allow trivial rules to separate classes, and in *Bot-IoT*, benign traffic is so rare (less than 0.01%) that a 1% false-positive rate is operationally unusable. This demonstrates that models often memorize artifacts instead of learning real attack behavior.

Outdated Attack Scenarios

Even when labels are correct, the attack traces in widely cited datasets may no longer reflect current threat landscapes. Catillo *et al.* reproduce several DoS campaigns from *IDS2017* and show that they fail against well-secured servers, yet still yield perfect offline detection scores [CPV23]. Ring's survey reaches a related conclusion, noting that benchmark datasets quickly become obsolete as protocols and attack techniques evolve [RWS+19c]. Coupled with extreme class imbalance (e.g., DoS Hulk contributes 230 k flows in *IDS2017* whereas Heartbleed appears only eleven times) models can achieve very high macro scores while entirely missing rarer but high-impact exploits.

Taken together, the structural limitations and data-level pitfalls detailed above demonstrate that relying solely on existing NIDS benchmarks is an untenable strategy. This creates a significant dilemma: while the ideal solution would be to create and validate entirely new, large-scale datasets, this process is often impractical due to significant logistical, privacy, and financial challenges. In response to this dilemma, researchers have increasingly turned to data augmentation. It is critical to clarify, as highlighted by the 'data smell' analysis, that augmentation does not correct the inherent errors (such as 'Wrong Labels' or 'Traffic Collapse') embedded in faulty datasets. Instead, augmentation provides a pragmatic approach to mitigate the consequences of these limitations. For example, realism-oriented augmentation directly attempts to generate more credible, stateful traffic than the unrealistic or unrepresentative data points found in benchmarks. Similarly, performance-oriented techniques mitigate the severe structural issue of class imbalance, and robustness-testing synthetically explores the attack diversity that is often missing ('Poor Data Diversity'). The following section reviews these data augmentation techniques, organized according to their primary objective and application.

4.3 Data Augmentation for NIDS

Generically, data augmentation methods create new, synthetic training or evaluation samples, but their specific application in the literature varies significantly based on the intended outcome. Research in this area can be broadly categorized according to three principal objectives.

The first objective is performance-oriented. These methods augment the training data, typically by re-balancing class-imbalanced datasets or enlarging the pool of minority samples. Thanks to this modified training distribution, the NIDS learning process is improved, and performance is expected to be higher when detecting attacks from these underrepresented classes.

The second objective is realism-oriented. These methods focus on generating traffic that faithfully reproduces the complex, stateful logic of real network protocols, which may be missing from static benchmarks. By training on this more realistic data, the NIDS is expected to learn more meaningful, generalizable patterns rather than dataset-specific artifacts. This also allows the model to form a more accurate decision boundary for normal behavior, potentially reducing false positives triggered by benign traffic that was simply absent from the original, unrealistic dataset.

The third objective is robustness-oriented. These methods are used to evaluate a model by crafting, in many cases, adversarial or otherwise modified traffic that challenges the detector's decision boundaries. By using this data in the evaluation stage, researchers can measure the model's robustness and better understand the specific conditions or input transformations that lead to misclassification, which is a different goal from improving standard detection metrics.

4.3.1 Performance-Oriented Data Augmentation Techniques

Performance-oriented data augmentation seeks to raise conventional detection metrics (accuracy, recall, F1) by enriching the training set with additional synthetic traffic. The aim is to give the NIDS a larger and better-balanced training set, rather than focusing on perfect realism in every generated sample. Extra data may come from simple resampling of underrepresented records, scripted injection of attack traces into background captures, or generative models that create entirely new samples from learned distributions. Whatever the mechanism, the shared idea is that a detector exposed to a broader spectrum of benign and malicious patterns will generalize more effectively to the standard hold-out test split.

Oversampling and Resampling Techniques

Early work in network-intrusion detection drew heavily from classic oversampling techniques developed for tabular ML tasks. The simplest variant, random duplication of minority records, appears in several studies that revisit *KDD99* and *NSL-KDD* [TBLG09]. In these datasets, for example, attack classes such as User-to-Root (U2R), where an insider attempts to escalate privileges, and Remote-to-Local (R2L), where an external attacker tries to gain local access, each constitute less than 1% of the corpus, creating a severe class imbalance. A more refined option is SMOTE [CBHK02], which synthetically creates new minority samples by interpolating between existing ones. For a given minority sample, it selects one of its nearest neighbors from the same class and generates a new sample at a random point along the line segment connecting the two. Bagui and Li [BL21] show that combining random undersampling with SMOTE (RU–SMOTE) markedly improves detection of minority attack classes in several benchmark NIDS datasets. Recent work extends the idea to sequence data: Wang *et al.* evaluate 18 header-sequence transformations (translation, wrap, and masking) and report up to a 4.4-percentage-point gain in

weighted F1 across three flow datasets [WFM+24]. These methods are dataset-agnostic, require no packet-level information, and add almost no computational overhead.

Scripted Attack Injection

A second group of performance-focused methods augments packet captures by injecting scripted attacks into real background traffic. A typical tool is ID2T [CVW+20]. After the user chooses an attack template (for example, a vertical TCP-SYN port scan or a SYN-flood denial-of-service), the toolkit rewrites timestamps and IP addresses, can adjust MAC addresses, updates the necessary TCP/IP header fields, recalculates checksums, and inserts the new packets so that the synthetic conversation integrates into the original PCAP. When the enriched capture is converted to flows and added to the training set, previously under-represented classes are rebalanced; García Cordero et al. showed that an ML-based NIDS trained on MAWI traffic with ID2T-generated port-scan and SYN-flood traces flagged every injected attack while producing no alerts on the clean baseline [CHMF16].

Generative Adversarial Networks (GANs)

Generative adversarial networks (GANs) provide a data-driven alternative to rule-based injection: instead of scripting fixed templates, a generator learns the joint distribution of flow features and then produces entirely new samples. A GAN framework involves two competing neural networks: a Generator that creates synthetic data and a Discriminator that tries to distinguish the synthetic data from real data. The Generator is trained to produce data that is realistic enough to 'fool' the Discriminator [GPM+14]. GADoT [ASDS21] uses a more stable GAN called a Wasserstein GAN with gradient penalty; this version does not need class labels, so it is called *unconditional*. With these synthetic DDoS flows, GADoT then fine-tunes the detector, cutting the false-negative rate for HTTP GET flood traffic in the *IDS2017* dataset from 61.5% to 1.8%.

Protocol-Aware and Self-Supervised Augmentation

A recent protocol-aware method, Rosetta [XZN23], addresses the critical performance drop of encrypted traffic classifiers when moved to new network environments, a failure caused by overfitting to packet length sequences that are altered by normal TCP mechanisms. To solve this, Rosetta combines self-supervised learning with a synthetic, protocol-conforming augmentation strategy. These augmentations are not intended to generate entirely new, realistic sessions, but are instead derived by simulating TCP mechanisms (like packet duplication and shifts) with the explicit performance goal of making the classifier robust to these network variations. The method's workflow, which you can see in Figure 4.1, consists of three phases. In Phase I, a "Traffic Invariant Extractor" (TIE) is trained. This is achieved by applying these protocol-aware augmentations to an original TLS/TCP flow, creating multiple "Flow Variants". A self-supervised encoder is then trained to produce similar feature vectors for all variants originating from the same flow, forcing it to learn features that are robust to network condition changes. In Phase II, this well-trained TIE is used as a static feature extractor. The labeled training dataset is passed through the TIE to generate robust feature vectors, which are then used to train a simple classifier. Finally, in Phase III, traffic from diverse network environments is processed through the same TIE to produce invariant feature representations before classification. This design improves robustness to network-induced variations (e.g., delay and retransmissions) and outperforms simple sequence augmentations such as *Random Mask* and *Random Swap*, while preserving protocol-consistent behavior.

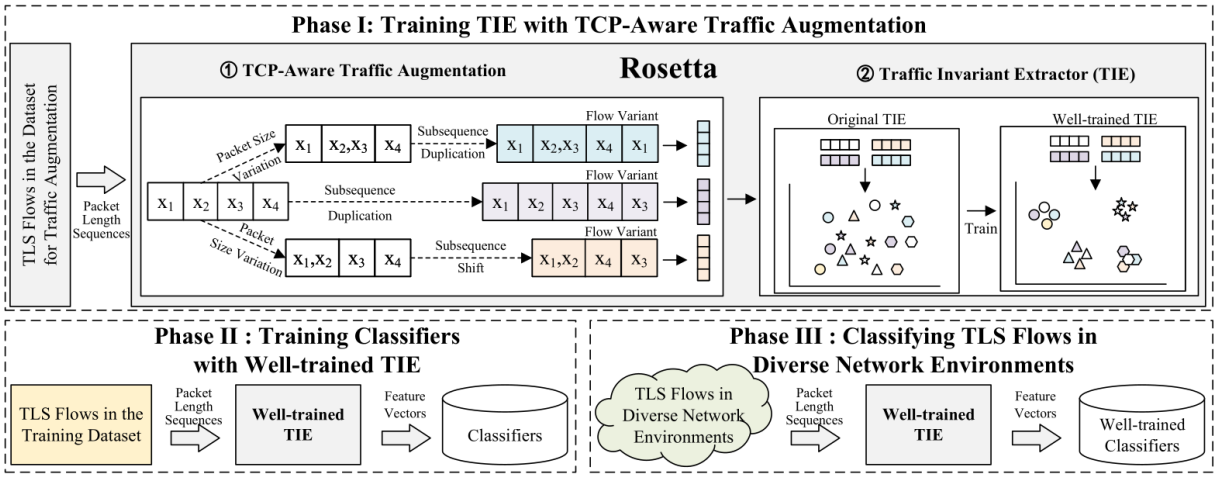


Figure 4.1: Overview of the Rosetta workflow for robust encrypted traffic classification, illustrating the three phases of Traffic Invariant Extractor (TIE) training, classifier training, and deployment. Adapted from [XZN23].

4.3.2 Realism-Oriented Traffic Augmentation

In the NIDS context, data augmentation is often used in a realism-preserving manner, deliberately creating synthetic packets and flows that mirror the protocol grammar, timing patterns, and cross-flow dependencies observed in live networks. By respecting header semantics, sequence numbers, and inter-arrival distributions, such methods focus on providing data that reflect production-like conditions without breaching privacy or incurring the expense of large-scale capture campaigns. Research on realism-preserving augmentation for NIDS has converged around two complementary directions: *Coarse-grained generators* and *fine-grained generators*.

Coarse-Grained Generators

Coarse-grained generators create synthetic traffic at the flow or aggregate-statistic rather than at the packet level; they do not model per-packet timing or header sequences. A prominent example is CTGAN [XSCIV19], which treats each flow record as a table row. It first makes skewed continuous features easier to learn by grouping each feature into a few typical value ranges and standardising values within the chosen range. Then it trains a conditional GAN that, during training, explicitly picks a category in a discrete feature and asks the generator to produce a full row consistent with that choice, so rare categories are seen often enough to be learned. To evaluate realism, the authors check if a model trained only on the generated data can perform as well on a real test set as a model trained on the original data.

DoppelGANger [LJW+20] extends this idea to generate time series with associated session metadata. Its main contribution is to separately model the static metadata of a session (e.g. user, ISP) and the sequential measurements (e.g. traffic volume over time), ensuring the two remain realistically correlated. Its follow-up, STAN [XMAR21], uses an autoregressive approach to the same problem. It learns to generate the next flow record in a sequence by predicting it from a window of the records that came immediately before it. This method is designed to explicitly capture both the temporal dependencies between consecutive flows and the statistical relationships between fields within each new record. They evaluate realism by training downstream models on the generated data and testing them on real data, as well as by comparing

key statistical properties (like temporal correlations and marginal distributions) between the synthetic and original datasets.

NetShare [YLJ+22] refines the DoppelGANger design to create a practical system for generating packet- and flow-header traces, focusing on scalability and privacy. Its end-to-end workflow, illustrated in Figure 4.2, introduces several key innovations. First, to handle high-cardinality fields, it uses domain-specific encodings. IP addresses are converted to bit vectors, while ports and protocols are mapped to an embedding vector. This embedding is pre-trained on public data, which avoids leaking privacy-sensitive distributions from the target dataset. Second, to address scalability, it splits the full trace into smaller, time-based chunks. A GAN model is fully trained on the first chunk, and this model is then used as a warm start to rapidly fine-tune parallel GAN models on the subsequent chunks, significantly reducing total training time. Third, it improves the poor privacy-fidelity tradeoff of standard DP-SGD (Differentially Private Stochastic Gradient Descent). This method adds noise at each training step to protect individual privacy, but the cumulative noise from training a complex model from scratch often destroys the data's realism. NetShare mitigates this by leveraging pre-training. It first trains a model on a large, public dataset (which requires no noise) and then fine-tunes that model on the private target data using DP-SGD. The system's realism is evaluated by comparing statistical distributions against the real data and by measuring if the synthetic data preserves the accuracy and relative algorithm rankings for downstream networking tasks like anomaly detection.

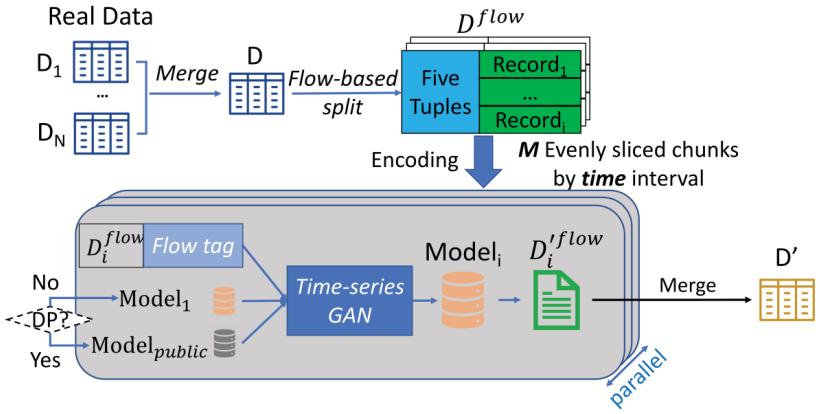


Figure 4.2: Overview of the NetShare synthetic trace generation workflow, illustrating the trace splitting, scalable model training, and differential privacy mechanisms. Adapted from [YLJ+22].

Fine-Grained Generators

Fine-grained generators operate at packet resolution, modelling byte sequences, inter-arrival times, and cross-flow dependencies to reproduce the detailed structure of live traffic. Tools such as the ns-3 network simulato [HRLR08] and the TRex traffic generator [Inc15] can replay user scripts or captured traces, sending byte-accurate packets. Their variability is scripted rather than learned from data. Recent ML-based generators learn these complex dependencies directly. NetDiffusion [JLGJ+24] reframes the task as image-based generation, converting traces into images and using a controlled diffusion process (with simple protocol checks) to achieve high statistical fidelity. Subsequent work like TrafficGPT [QML24] borrowed from large language models, using a workflow (Figure 4.3) that first splits PCAP traces into session flows and then converts them into long, reversible token sequences (including time-interval tokens). It pre-trains

a Transformer on this token data, replacing standard self-attention with a linear attention mechanism to scale the context window to about 12k tokens (≈12,032). While this captures longer dependencies, the approach remains computationally intensive. The latest line of work, exemplified by NetSSM [CJL+25], addresses this scalability bottleneck by replacing the attention mechanism entirely with a linear-time state-space model (Mamba). This model processes the sequence token-by-token using a compact, recurrent state, thereby avoiding the quadratic computational cost of the Transformer's all-to-all attention mechanism. This stateful architecture is a more natural fit for network protocols, extending the modelling context to thousands of packets and capturing complex multi-flow interactions.

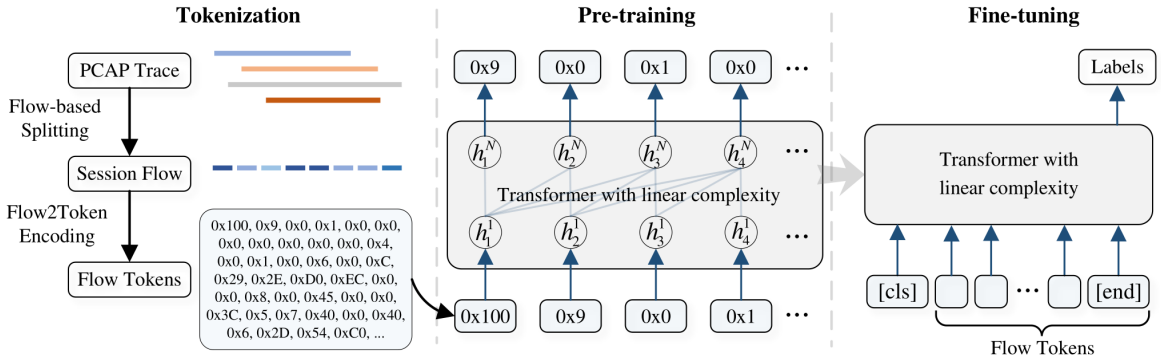


Figure 4.3: The TrafficGPT architecture for traffic analysis and generation, detailing the tokenization process, Transformer pre-training, and downstream adaptation tasks. Adapted from [QML24].

Assessing how closely synthetic traces resemble real traffic remains an open question, and no single metric has become standard. NetSSM [CJL+25] computes the Jensen–Shannon Divergence, Total Variation Distance, and Hellinger Distance, three distinct statistical metrics that quantify the dissimilarity between probability distributions. It performs this comparison on a byte-wise view of all TCP header fields, measuring the difference between the distribution of the generated synthetic data and that of the original real data to assess statistical fidelity. Using multiple metrics provides a more robust evaluation, as they capture different aspects of distributional error. NetDiffusion [JLGJ+24] uses the same three distances and additionally lists per-field header-bit difference percentages that highlight mismatches in payload length and TCP-flag usage. MirageNet is a proposed framework for generating synthetic network models of entire protocols or applications; its MiragePkt [NAN22] module is the first component, a GAN-based model designed to learn and generate the raw byte-level syntax of individual packets. This module emphasises packet-level validity. During evaluation, it parses every generated DNS packet with Scapy and reports three metrics: Correctness (percentage of well-formed packets), Fakeness (share of valid packets not seen in the training set), and a byte-string Similarity score. Scatter plots of the first-versus last-byte combinations of each payload help visualize how closely synthetic patterns match the training data.

4.3.3 Test-Set Augmentation for NIDS Robustness Testing

A primary goal of data augmentation is to improve NIDS robustness by systematically exposing a model to challenging and novel traffic. This is particularly critical for the test set, as deployment environments expose models to traffic distributions that evolve far beyond those seen during training. Without credible evidence that a test set truly stresses a model's decision boundary,

high detection scores derived during the model-construction phase can be misleading.

To effectively augment a test set to cover its weaknesses, one must first be able to assess its quality. This establishes a two-stage process for robust evaluation. The first stage involves employing assessment methodologies to quantify a test set's rigor, for which the literature describes three main families of techniques: *Neuron Coverage*, *Surprise Coverage*, and *Mutation Testing*. The second stage uses generation techniques to create new challenging data points for a rigorous evaluation.

Test set Assessment Methods

- **Neuron Coverage** (NC) adapts the classic statement-coverage idea from software testing to DL models by counting the fraction of hidden units in a neural network whose activation exceeds a predefined threshold when the network processes a test set [PCYJ17]. Formally, let $a_i(x)$ denote the post-activation value of neuron i for input x and let θ be a user-chosen threshold; a neuron is considered covered if $a_i(x) > \theta$ for at least one x in the test set. The overall NC score is the number of covered neurons divided by the total neuron count. Numerous refinements target different aspects of internal behaviour: *multi-section* NC partitions each neuron's value range into equal bins and tracks which bins are exercised; *top-k* and *layer-wise* variants normalise activations per layer to mitigate scale differences; and *combinatorial* coverage records joint activation patterns of neuron pairs or triplets to approximate control-flow interactions [YPW23, LTL+24]. These extensions share the same guiding principle, broader activation implies more thorough behavioural exploration, while offering finer granularity for test-set adequacy studies.

- **Surprise Coverage** builds on the intuition that an adequate test set should contain inputs whose internal activation patterns deviate from those induced by the training data. The canonical formulation, Surprise Adequacy (SA), proposes two complementary metrics: *Likelihood-based Surprise Adequacy* (LSA) computes the negative log-likelihood of a neuron-activation vector under a Gaussian Mixture Model fitted to training activations, whereas *Distance-based Surprise Adequacy* (DSA) measures the ℓ_2 distance between a test activation and its nearest training counterpart, normalized by the distance between that same-class reference and the nearest activation from a different class. This normalization yields a dimensionless score that is comparable across layers and classes [KYC+19] (see Figure 4.4 for a geometric illustration). Both metrics yield scalar surprise scores that can be thresholded to define coverage: a neuron-space region is deemed covered if at least one test input attains a surprise value exceeding a preset percentile of the training distribution [ZMSC21].

- **Mutation Testing** adapts fault-based assessment ideas from traditional software engineering to DL models. The procedure first creates a population of mutants by applying small, semantics-preserving transformations (called mutation operators) to the reference model [MJZ+18]. This semantics-preserving constraint ensures the mutants represent subtle internal faults rather than a complete breakdown of the model's logic. These transformations are either source-level, which alter training data and require a full retraining, or model-level, which directly modify a trained model's parameters by perturbing trainable weights (e.g., Gaussian noise injection), deactivating neurons, shuffling convolutional kernels, or altering layer hyperparameters such as dropout rates and activation functions [MJZ+18]. The objective is to determine if the test set $\mathcal{T}$ is sensitive enough to detect these injected faults. To measure this, $\mathcal{T}$ is executed on all M mutants $\{\tilde{f}_1, \ldots, \tilde{f}_M\}$.

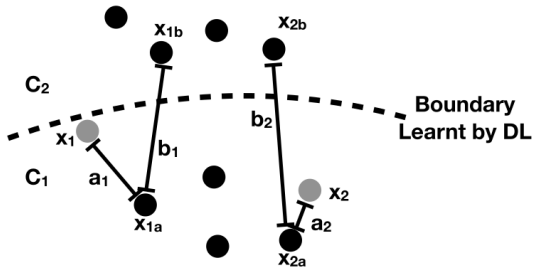


Figure 4.4: Geometric illustration of Distance-based Surprise Adequacy (DSA) in the neural network activation space. The diagram visualizes how novelty is quantified based on the relative distances between a test input and the learned training class clusters. Adapted from [KYC+19].

A mutant $\tilde{f}_j$ is considered *killed* if any input $x \in \mathcal{T}$ causes its output to differ from the original model: $\tilde{f}_j(x) \neq f(x)$, where f denotes the original model. The resulting Mutation Score is MS $= K/M$, with K the number of killed mutants (see Fig. 4.5 for an illustration of original vs. mutant decision boundaries). A high MS indicates that the test set exposes injected faults effectively, suggesting thorough scrutiny of the decision surface. Refinements such as *DeepMutation++* [JMZ+20] enlarge the operator catalogue to include weight clustering and neuron swapping, while *DeepCrime* [SYL+21] integrates metrics that measure how sensitive the model's output is to its internal parameters. This information is used to guide operator selection toward the neurons or weights that most influence the final classification decision. These advances seek higher fault diversity without requiring prohibitively large mutant pools.

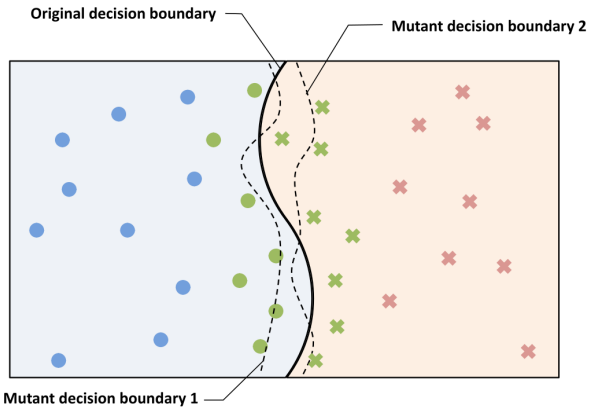


Figure 4.5: Decision boundaries of the original model and two mutant models for a binary classifier. Points near the original boundary (shown in green) are more likely to cause disagreement with mutants, thus "killing" them; mutation testing therefore favors tests in this sensitive region. Adapted from Ma *et al.*, DeepMutation (Fig. 4) [MJZ+18].

Boundary-Case Augmentation Methods

Boundary-case augmentation targets the robustness of a NIDS by generating traffic that lies close to, or deliberately crosses, the model's decision boundary. This approach seeks corner situations where small perturbations alter the detector's verdict. Boundary inputs appear in three main

forms: (**i**) adversarially modified benign flows that evade detection; (**ii**) minimally changed attacks that become false negatives; and (**iii**) fuzzing-based variants that iteratively adjust header fields or timing parameters until the classifier shows maximum uncertainty [RHJT21a, ZSM+22]. To be meaningful, the generated traces must remain protocol-compliant, invalid packets reveal implementation bugs rather than conceptual weaknesses, yet still expose gaps invisible in conventional testing.

Gradient-Guided Adversarial Augmentation Adversarial or gradient-guided augmentation creates boundary inputs by making minimal, target-aware changes to existing samples until the detector's verdict changes. Early work adapted foundational attacks from the deep learning literature, such as *FGSM* [GSS15] and *PGD* [KGB17] originally developed for image models, to numerical flow features, showing that a few feature-space steps can already lower detection scores. More recent approaches employ learned generators. NIDSGAN [ZSM+22] uses a multi-objective GAN architecture (see the workflow in Figure 4.6). Rather than creating flows from scratch, its generator learns a small perturbation vector $\Delta\vec{x}$ for an existing real attack flow $\vec{x}$. The adversarial example $\vec{x}^* = \vec{x}+\Delta\vec{x}$ is then optimised against two modules: (i) the target NIDS, which provides an adversarial loss that pushes the prediction toward benign, and (ii) a discriminator, which provides a realism loss that encourages the perturbed feature vector to match the distribution of authentic attack traffic. In practice, the generator's objective also includes a perturbation-size penalty, and explicit domain constraints are enforced by masking one-hot or protocol-bound fields and restricting numeric features to valid ranges. However, a critical limitation is that this realism is defined strictly in feature space. The paper reports evasion success rather than independent packet-level realism metrics (for example, parser acceptance, checksum correctness, or replay success). Although the authors discuss reconstructing and replaying modified PCAPs with tools like tcpreplay, full on-wire validation is not performed, so the gap between feature-space plausibility and protocol-faithful traffic remains unquantified [ZSM+22].

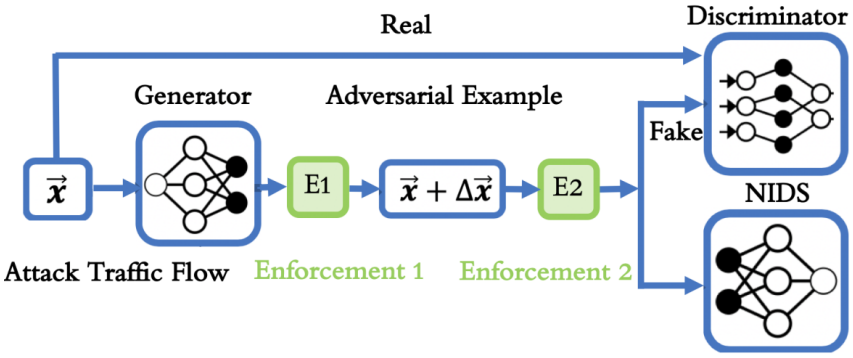


Figure 4.6: Overview of the NIDSGAN adversarial generation architecture, illustrating the generation of perturbation vectors and the dual-evaluation constraints for evasion and realism. Adapted from [ZSM+22].

LLM and Reinforcement Learning Approaches *AdvTG* [SYL+25] uses a more complex, three-phase workflow that links a language model with reinforcement learning (Figure 4.7). The goal is to generate adversarial HTTP payloads that are evasive, protocol-compliant, and functionally intact. This content-based method is designed to target NIDS positioned after a TLS/SSL decryption point, as it must read and modify the plaintext payload to be effective. The first

phase, Detection Model Training, trains the target NIDS models. These multiple packet-level, DL-based NIDS (using both text and image features) are then used as black-box reward models for the final RL stage. The second phase, Domain-Specific LLM Fine-Tuning, prepares the generator. A general-purpose LLM is fine-tuned on a large corpus of HTTP traffic, creating a Domain-Specific LLM that understands the patterns and protocol formats of HTTP. The third phase, RL-based Adversarial Generation, adapts this generator for an adversarial task. This domain-aware LLM is fine-tuned again, but this time using RL to learn evasive behavior. In this stage, the LLM is prompted with the functional fields of a malicious packet but given an opposite instruction (e.g., generate benign traffic), guiding it to find evasive content for the non-functional fields. The LLM's output is then scored by the reward models from Phase 1. This score, combined with a KL-divergence penalty to ensure the output remains statistically realistic, is used by a PPO algorithm [SWD+17] to update the LLM. The authors evaluate protocol compliance by parsing the generated HTTP traffic and checking for a valid request line, correct header-field formats, and the presence of the required header-body separator. However, this compliance check is largely limited to basic syntactic structure and a few RFC-inspired rules, and does not verify header–payload consistency or server-side behaviour, so the reported compliance rates should be interpreted as well-formedness of the HTTP format rather than a full guarantee that the generated traffic would be accepted and processed as intended by real-world implementations.

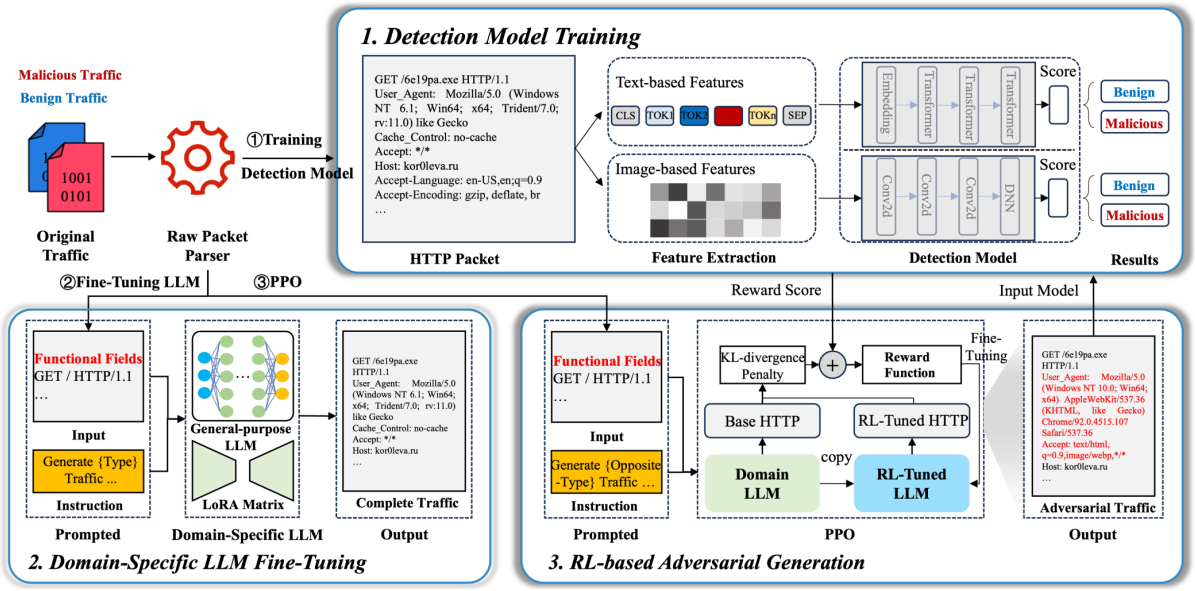


Figure 4.7: The AdvTG three-phase workflow for adversarial generation. (1) Detection Model Training: Multiple NIDS models are trained to serve as black-box reward functions. (2) Domain-Specific LLM Fine-Tuning: A general-purpose LLM is fine-tuned on HTTP traffic to learn the protocol's structure. (3) RL-based Adversarial Generation: The domain-specific LLM from Phase 2 is optimized via reinforcement learning, using feedback from the Phase 1 reward models to generate evasive traffic. Adapted from Sun *et al.* (WWW 2025), Fig. 3 [SYL+25].

These methods effectively populate the region near the detector's decision boundary, achieving a high proportion of samples where small perturbations flip the model's verdict when gradients or reliable surrogate models are available, while keeping modifications confined to a small subset of features or fields. Their main drawbacks are the need for gradient access (or many detector queries) and the limited semantic or protocol validation, typically implemented through simple feature-range or syntactic checks.

Black-Box and Fuzzing-Based Augmentation In contrast to the gradient-guided methods discussed previously, fuzzing- and search-based augmentation methods explore a model's decision boundary without relying on gradient information or detailed model internals. They operate by repeatedly mutating the input and retaining variants that drive confidence down or cause misclassification. ProGen [WYFPW24], operate in a purely black-box manner by learning a distribution mapping. This approach does not optimize a separate perturbation for each individual sample, but instead learns a general function to transform the malicious traffic distribution so that its samples statistically resemble benign traffic. ProGen's workflow, shown in Figure 4.8, implements this function using a GAN (specifically, DoppelGANger). In its (1) Training Phase, the generator $\mathcal{A}(\cdot)$ is trained to take malicious traffic ($\mathcal{F}^a$) as input and output adversarial malicious traffic that is statistically indistinguishable from real benign traffic ($\mathcal{F}^b$) to a surrogate discriminator ($\mathcal{D}'$). This generator's structure handles a flow's two distinct data types: an MLP first processes the static, flow-level meta-data (like the 5-tuple), and its output is then fed into an LSTM, which models the temporal packet series (the sequence of packet features over time). In the (2) Launching Phase, this trained generator is used to conduct an attack. A trace of the original malicious traffic is fed into the generator, which outputs a new, evasive adversarial malicious traffic trace. A Traffic Rebuilder then combines this new adversarial trace with the original, reserved malicious payload to create the final packet sequence sent to the target NIDS. The authors assess realism by plotting the distributions of a few traffic features and by checking rule-based constraints on timing, volume, ports, and flags. This goes beyond reporting detector accuracy alone, but it still focuses on simple one-dimensional statistics and hand-crafted rules, so it cannot capture more complex dependencies or guarantee that the traffic would behave correctly in a real network.

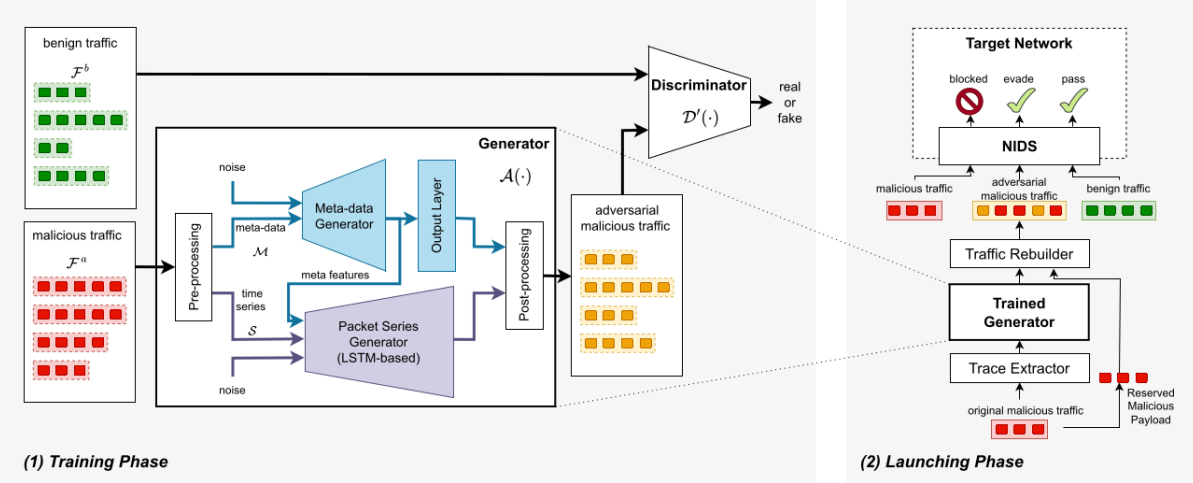


Figure 4.8: The ProGen adversarial generation workflow, illustrating the two-phase process of training a black-box generator and launching evasive malicious traffic traces. Adapted from [WYFPW24].

Beyond the networking community, DeepMetis shows that a mutation-score oracle can steer generation toward boundary cases for image classifiers. It uses a pool of DNN mutants and a multi-objective evolutionary search that mutates structured input models until they both increase mutation score and remain diverse, thereby accumulating many corner-case examples near the models' decision boundaries [RHJT21a]. To preserve realism, DeepMetis manipulates higher-level input representations (such as vectorised digit shapes or eye-model parameters) instead

of raw pixels, which keeps the generated tests largely within the valid input domain, although realism is only argued qualitatively and not validated in real deployments.

4.4 Conclusion

This chapter has established the critical role of data augmentation in addressing the well-documented deficiencies of public NIDS benchmarks. Our initial review confirmed that existing datasets, while essential for reproducibility, are frequently compromised by structural limitations and data-level artifacts. Because these flaws can lead to inflated performance metrics and a misleading indicator of a model's real-world operational effectiveness , the research community has considered data augmentation as one of the approaches in order to mitigate these inherent limitations.

To organize this growing field, Table 4.1 provides a synthesis of the augmentation strategies discussed in this chapter. These techniques are categorized according to three complementary objectives: improving model performance through better training distributions, preserving traffic realism to avoid learning artifacts, and evaluating model robustness through boundary-case generation.

Building on this general overview, the remainder of this thesis narrows its focus specifically to the third objective: test-set augmentation for NIDS robustness testing. The current methodologies within this domain face two main practical limitations. First, they typically require white-box access to the model, making them tightly coupled to specific neural architectures and difficult to transfer to proprietary systems. Second, many of these procedures produce synthetic rather than protocol-faithful traffic, introducing unrealistic artifacts that can bias evaluation. These limitations directly motivate the need for a model-agnostic, black-box approach to assessing test-set adequacy and guiding augmentation toward realistic, label-preserving data points. The following chapter develops and validates precisely this methodology.

Table 4.1: Comparison of network-traffic augmentation strategies for NIDS evaluation.

Class (Sec.)	**Primary goal**	**Typical techniques**	**Protocol safeguards in place?**
Performance-oriented	Increase training diversity and balance classes	Random oversampling, SMOTE [CBHK02]; rule-based attack injection (ID2T [CVW+20]); GANs for minority traffic (MiragePkt, GADoT)	Rare; focus is on label balance, not byte-level realism
Realism-preserving	Mirror production traffic in train/test corpora	Flow-level GANs (CTGAN, DoppelGANger, NetShare); packet-level diffusion, transformer, and state-space models (NetDiffusion, TrafficGPT, NetSSM)	Yes: header masks, checksum repair, grammar loss, DPI validation
Boundary-case (robustness)	Reveal worst-case errors near the decision boundary	Gradient/RL attacks (FGSM, NIDSGAN, AdvTG); query-based fuzzing (DeepXplore, DeepHunter, ProGen)	Basic only: pattern checks or DPI validator, often optional

Chapter 5

TATA: An Approach for Assessing Benchmark NIDS Test-Sets and Targeted Augmentation

Contents

5.1 Introduction 95
5.2 TATA Design 98
5.2.1 Overview 98
5.2.2 Optimized Latent Representation for Training Set 100
5.2.3 Test Set Quality Assessment 100
5.2.4 Test Set Targeted Augmentation 104
5.3 Conclusion 105

5.1 Introduction

The previous chapter provided a broad overview of data augmentation techniques for network intrusion detection. As summarized in Table 4.1, that analysis established the comprehensive landscape of the field, categorizing methods by their primary objectives: improving training performance, preserving traffic realism, and testing boundary cases. Building on that foundation, this chapter narrows the focus strictly to the final objective. We concentrate on the critical tasks of test set assessment and test set augmentation. This chapter brings these two targeted concepts together to answer two research questions:

- **RQ1:** How can we **assess** the quality of test sets in challenging NIDS models, ensuring a robust evaluation of their detection capabilities?
- **RQ2:** How can we systematically **augment** these test sets to better evaluate the robustness and real-world applicability of NIDS models?

To contextualize these questions, Table 5.1 provides a focused synthesis of existing methods dedicated specifically to test set assessment (RQ1) and augmentation (RQ2). This table isolates techniques based on their capacity to answer RQ1 and RQ2, exposing critical gaps in the current state of the art.

As detailed in the table, the primary limitation across the assessment methods (Neuron Coverage, Surprise Coverage, and Mutation Testing) is their strict **need for model access**. Specifically, Neuron Coverage and Surprise Coverage require hidden-layer activations, while Mutation Testing demands read-write control over weights and hyperparameters, permissions that closed-source NIDS rarely provide. Furthermore, these methods are **not adapted to non NN models**, rendering them completely inapplicable to widely deployed algorithms such as Random Forests. Beyond access restrictions, these techniques face severe operational constraints. Neuron Coverage demonstrates **high sensitivity to hyperparameters**, requiring case-by-case expert calibration of activation thresholds and normalization rules. Mutation Testing suffers from a **lack of adaptability to model changes**, meaning that any architectural update requires the design of entirely new mutation operators, and advanced variants often dictate that the system **needs a new training run per dataset**. Methodologically, these metrics **ignore training-test alignment** because they evaluate the test set in isolation. This indifference allows test inputs to yield high coverage scores even when they simply replay patterns the model has already memorized. Finally, methods like Surprise Coverage rely on **only proximity-based analysis**. By reducing novelty to mere point-wise proximity between test samples and training data within the model's activation space, a few outliers can artificially inflate the metric while large regions of untested traffic remain entirely uncovered.

On the augmentation side, current approaches are equally restricted. Methods often default to **manual and unguided test set augmentation**, which inserts traffic randomly rather than systematically targeting the models' true weaknesses. Alternatively, they rely on the **use of a data generator that may produce unrealistic data points**. These generators frequently output synthetic traffic instead of protocol-faithful packets, introducing unrealistic artifacts that can heavily bias the evaluation and overstate the detector's true robustness.

Table 5.1: Methods & limitations for test set assessment (RQ1) & augmentation (RQ2)

Method category	**Studies**	**RQ1**	**RQ2**	**Limitations**	
Neuron coverage	[PCYJ17, MJXZ+18, MJXX+19]	✓	✗	• Need for model access • Ignores training–test alignment	• High sensitivity to hyperparameters • Not adapted to non NN models
Surprise coverage	[KFY19, WCT21, KFY23]	✓	✗	• Need for model access • Not adapted to non NN models	• Only proximity-based analysis
Mutation testing	[HMX+19, TKW+20, HJT21]	✓	✗	• Need for model access • Lack of adaptability to model changes	• Ignores training–test alignment
Mutation testing	[RHJT21b]	✓	✓	• Same limitations as [HMX+19, TKW+20, HJT21] • Needs a new training run per dataset	• Use of a data generator that may produce unrealistic data points
N/A	[FEAD24b, FA24]	✗	✓	• Focus solely on increasing proximity	• Manual and unguided test set augmentation

To overcome these shortcomings, we introduce our method, **TATA (Test-set Assessment and Targeted Augmentation)**, a framework explicitly designed to diagnose the deficiencies of a test set and remedy them with realistic traffic. To answer **RQ1**, TATA relies on a contrastive

auto-encoder that projects both training and test data points into a structured latent space. This representation enables the simultaneous approximation of decision boundaries, without requiring access to model internals, and the characterization of the test set's spatial alignment relative to the training distribution.

Within that space, TATA gauges the difficulty of a test set by three complementary, dataset-centric metrics: **(i)** Diversity measures how broadly the test data points cover the latent space, penalizing redundancy and rewarding a wide spread of samples. **(ii)** Proximity quantifies how close test data points are to regions occupied by differently labeled training data points; smaller distances indicate that a point sits near the decision boundary, making it intrinsically harder to classify **(iii)** Scarcity evaluates whether the test data points are uniformly distributed across all distinct decision regions, ensuring that every potential error zone is exercised instead of stressing just one corner of the space. High values on the three axes indicate a challenging benchmark, whereas low values reveal specific gaps.

Once these specific coverage gaps are diagnosed, TATA directly addresses **RQ2** by applying targeted data augmentation to systematically fill them. A critical constraint here is that the added data points must be realistic, as evaluating a NIDS prior to deployment requires valid network traffic rather than statistical artifacts. Consequently, purely synthetic approaches are insufficient. Such methods, ranging from network simulators that approximate protocol behavior to generative models like GANs or diffusion models that produce numeric feature vectors, fail to capture real-world hardware inconsistencies or produce technically irreproducible traffic patterns. Instead, TATA generates data by leveraging a live testbed, ensuring that every data point results from actual packet exchanges that strictly adhere to network protocols and timing constraints.

However, blindly augmenting the test set with testbed-based generated data points is inefficient for targeting the specific deficiencies (diversity, proximity, or scarcity) revealed by our assessment. To ensure the augmented data precisely fills these gaps, the generation process requires intelligent guidance via the configuration of the underlying infrastructure. This involves manipulating factors that fundamentally alter the features of the generated flow while preserving its validity, for example, testbed-specific parameters (e.g. attack tool rates or payload sizes) or broader network environmental conditions (e.g. link latency, jitter, or bandwidth constraints). Since the search space of these configuration parameters is vast and non-linear, TATA delegates this task to a Reinforcement Learning (RL) agent. Acting as an adaptive configurator, the agent learns to manipulate these configurations, causing the testbed to generate valid traffic that maps precisely to underrepresented regions of the latent space. Crucially, because the agent's observations originate from this latent representation rather than classifier gradients, the entire pipeline remains model-agnostic.

Together, these three components (the contrastive autoencoder that builds a model-agnostic latent space, the triplet of dataset-centered metrics, and the reinforcement-learning traffic generator) form an integrated pipeline. As a whole, TATA specifically resolves the four critical limitations identified in Table 5.1:

- **Need for model access.** TATA entirely avoids this dependency by operating purely within a structured latent space learned directly from the input data points, requiring zero knowledge of the NIDS architecture, weights, or hidden activations.

- **Only proximity-based analysis.** TATA broadens the assessment perspective by introducing two additional criteria (diversity and scarcity). Consequently, a test set is flagged as inadequate if it is redundant or unbalanced, ensuring a comprehensive evaluation beyond mere closeness to decision boundaries.

- **Ignores training-test alignment.** By projecting both the training and test sets into the identical contrastive latent space, TATA explicitly maps, compares, and quantifies the spatial relationship between the two distributions.
- **Unguided augmentation and unrealistic data points.** TATA replaces manual, protocol-breaking generation with a reinforcement-learning agent that controls a live testbed. This guarantees the generation of highly realistic, protocol-compliant network traffic precisely targeted at the specific coverage gaps revealed by the assessment metrics.

The remainder of this chapter is structured as follows. Section 5.2.1 provides an overview of the framework, Section 5.2.3 formalises the test-set quality metrics, and Section 5.2.4 details the targeted augmentation procedure. Section 6.2 then presents the experimental setup, outlining the datasets in Section 3.2.2 and the contrastive auto-encoder architecture in Section 6.2.3.

5.2 TATA Design

5.2.1 Overview

TATA operates on a labeled NIDS training set ($\mathcal{D}_{\text{train}}$) and test set ($\mathcal{D}_{\text{test}}$), each containing network traffic data points $(\mathbf{x}, l_k)$, where $\mathbf{x}$ is a high-dimensional feature vector including information about traffic flow (e.g., the number of bytes exchanged), and $l_k \in \mathcal{L} = \{l_1, \ldots, l_n\}$ is the label (e.g., benign or a type of attack). Fig. 5.1 outlines TATA using an illustrative example with $\mathcal{L} = \{\text{Benign, Attack-1, Attack-2}\}$. TATA proceeds in three steps: *(i)* a preliminary phase that constructs a structured latent space from $\mathcal{D}_{\text{train}}$, *(ii)* phase 1 that assesses $\mathcal{D}_{\text{test}}$'s quality, and *(iii)* phase 2 that augments $\mathcal{D}_{\text{test}}$. $\mathcal{L}$ is the set of labels that the NIDS must classify.

Preliminary phase: structured latent space

The preliminary phase provides the latent space foundation by training a **contrastive autoencoder** with encoder f, parameterised by $\boldsymbol{\theta}$, on $\mathcal{D}_{\text{train}}$. During training, the contrastive objective [HCL06, K$^+$] shapes the latent space into well-separated clusters, each containing encoded training data points with similar labels and closely aligned input feature values, while preserving reconstruction accuracy (details in Section 5.2.2).

This process yields a trained f that encodes each $\mathbf{x} \in \mathcal{D}_{\text{train}}$ into a latent representation

$$\mathbf{z} = f(\mathbf{x}; \boldsymbol{\theta}), \tag{5.1}$$

forming

$$\mathcal{Z}_{\text{train}} = \{\mathbf{z} \mid \mathbf{z} = f(\mathbf{x}; \boldsymbol{\theta}),\ \mathbf{x} \in \mathcal{D}_{\text{train}}\}. \tag{5.2}$$

With the latent representations established, we determine the number of clusters in $\mathcal{Z}_{\text{train}}$ using the silhouette score, then apply k-means to assign each $\mathbf{z} \in \mathcal{Z}_{\text{train}}$ to a cluster C_{i,l_k} (where i is the cluster index and l_k is the majority label among its members). All clusters form the set $\mathcal{C}$ knowing that a single label may correspond to multiple clusters (e.g., $C_{1,\text{Benign}}$ and $C_{2,\text{Benign}}$ in Fig. 5.1), which reflects intra-class variability in $\mathcal{D}_{\text{train}}$.

This structural organisation offers distinct operational advantages. By reducing the data's dimensionality, the contrastive autoencoder not only speeds up subsequent computations but also approximates the decision boundaries inherent in $\mathcal{D}_{\text{train}}$, all without being tied to a specific NIDS model.

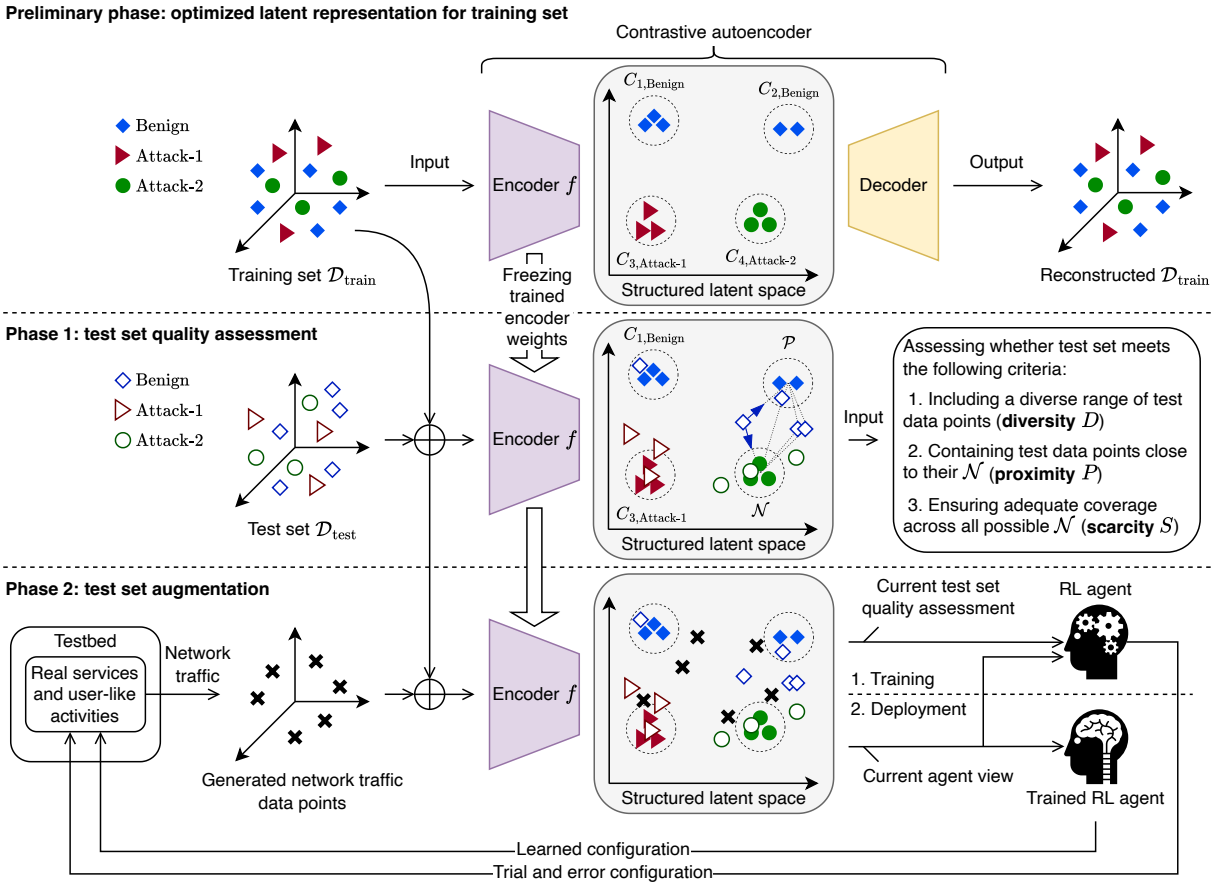


Figure 5.1: TATA, a method to assess and augment the quality of a test set

Phase 1: test set assessment

In phase 1, we use the trained f to encode $\mathcal{D}_{\text{test}}$ into the same structured latent space as $\mathcal{D}_{\text{train}}$. We obtain $\mathcal{Z}_{\text{test}} = \{\mathbf{z} \mid \mathbf{z} = f(\mathbf{x}; \boldsymbol{\theta}), \mathbf{x} \in \mathcal{D}_{\text{test}}\}$, which positions each test data point relative to $\mathcal{D}_{\text{train}}$'s distribution.

Once projected, we contextualise each test point by identifying its spatial relationships to the training clusters. For each $\mathbf{z} \in \mathcal{Z}_{\text{test}}$ with input label l_k , we compute its Euclidean distance

$$\|\mathbf{z} - \boldsymbol{\mu}_{C_{i,l_j}}\| \tag{5.3}$$

to every cluster centroid $\boldsymbol{\mu}_{C_{i,l_j}}$ in $\mathcal{C}$. Among these distances, the cluster C_{i,l_j} whose label l_j matches l_k and minimizes $\|\mathbf{z} - \boldsymbol{\mu}_{C_{i,l_j}}\|$ is termed as its positive cluster $\mathcal{P}(\mathbf{z})$. Conversely, the cluster C_{i,l_j} whose label differs ($l_j \neq l_k$) and minimizes that distance is its negative cluster $\mathcal{N}(\mathbf{z})$. Fig. 5.1 illustrates this pairing by arrows linking a Benign encoded test data point to its positive and negative clusters, a spacial relationship that represents the intrinsic difficulty of making the correct classification decision.

Leveraging these defined relationships, we assess the test set's quality. we define three complementary metrics (diversity, proximity, and scarcity) to assess how effectively $\mathcal{D}_{\text{test}}$ challenges the NIDS in its detection task. They are computed in the structured latent space on a **per-cluster basis** (details in Section 5.2.3). We then aggregate each metric's results across all clusters $C_{i,l_k} \in \mathcal{C}$ to measure how effectively $\mathcal{D}_{\text{test}}$ challenges each subgroup of $\mathcal{D}_{\text{train}}$.

Phase 2: test set augmentation

In **phase 2** (detailed in Section 5.2.4), the RL agent is trained by iteratively interacting, through trial and error, with a configurable testbed that it directs to generate network traffic (see Fig. 5.1). At each iteration, the agent first consults its current view of the structured latent space and then selects a traffic-generation configuration. The resulting network traffic is merged into the evolving $\mathcal{D}_{\text{test}}$ and, after encoding, is re-assessed with TATA's predefined metrics, yielding a quality measure that serves as the reward. This reward guides the agent's configuration choices, which are refined over multiple iterations. Once training, the agent's learned policy can be used on the same or other NIDS datasets without requiring further retraining (i.e., without additional reward signals).

5.2.2 Optimized Latent Representation for Training Set

To construct the structured latent space introduced in Section 5.2.1, we train a contrastive auto-encoder composed of an encoder f and a decoder h. Given a high-dimensional feature vector $\mathbf{x} \in \mathcal{D}_{\text{train}}$, the encoder produces a p-dimensional latent embedding $\mathbf{z} = f(\mathbf{x}; \boldsymbol{\theta}) \in \mathbb{R}^p$, while the decoder reconstructs $\hat{\mathbf{x}} = h(\mathbf{z}; \boldsymbol{\phi})$. Training minimizes a hybrid objective that couples standard reconstruction error with a pairwise contrastive term, thereby ensuring that the latent space captures both feature-based similarity and class separation:

$$\min_{\boldsymbol{\theta},\boldsymbol{\phi}} \underbrace{\mathbb{E}_{\mathbf{x}} \|\mathbf{x} - \hat{\mathbf{x}}\|_2^2}_{\text{Reconstruction loss}} + \lambda \underbrace{\mathbb{E}_{\mathbf{x}_i,\mathbf{x}_j} \Big[\underbrace{(1 - y_{ij})\big(d_{ij}^2 + (1 - \cos(\mathbf{z}_i, \mathbf{z}_j))\big)}_{\text{Closeness (when } y_{ij}=0)} + \underbrace{y_{ij}\big([m - d_{ij}]_+\big)^2}_{\text{Separation (when } y_{ij}=1)} \Big]}_{\text{Contrastive loss}} \quad (5.4)$$

where $d_{ij} = \|\mathbf{z}_i - \mathbf{z}_j\|_2$, $[\cdot]_+ = \max(0, \cdot)$, $\lambda > 0$ balances the two terms, and $m > 0$ is the contrastive margin [HCL06][15].

For each pair $(\mathbf{x}_i, \mathbf{x}_j)$ we define

$$y_{ij} = \begin{cases} 0, & \text{if } l_i = l_j \text{ and } \cos(\mathbf{x}_i, \mathbf{x}_j) \geq \tau \\ 1, & \text{otherwise,} \end{cases}$$

where l_i and l_j are the class labels and $\tau \in [0, 1]$ is a cosine-similarity threshold computed in the input space. When $y_{ij} = 0$ (same label, sufficiently similar features) the loss pulls the embeddings together by reducing both Euclidean and angular distance; when $y_{ij} = 1$ (different label or low similarity) it pushes them apart until their Euclidean distance exceeds m.

Optimizing Eq. (5.4) yields the embedded training set $\mathcal{Z}_{\text{train}} = \{\mathbf{z} = f(\mathbf{x}; \boldsymbol{\theta}) \mid \mathbf{x} \in \mathcal{D}_{\text{train}}\}$, which serves as the foundation for the assessment and augmentation phases that follow.

5.2.3 Test Set Quality Assessment

Because TATA's metrics are computed on a per-cluster basis, we begin by defining the subset of encoded test data points tied to each cluster C_{i,l_k}. Specifically, for each C_{i,l_k}, we focus on $\mathcal{Z}_{\text{test}}^{(C_{i,l_k})} = \{\mathbf{z} \in \mathcal{Z}_{\text{test}} \mid \mathcal{P}(\mathbf{z}) = C_{i,l_k}\}$, the $\mathcal{Z}_{\text{test}}$ subset of **encoded test data points whose positive cluster is** C_{i,l_k}. To support metrics computation, we also introduce the function $\text{PairNeg}(\mathcal{Z}_{\text{test}}, C_{i,l_k}) = \big\{\big(\mathbf{z}, \mathcal{N}(\mathbf{z})\big) \mid \mathbf{z} \in \mathcal{Z}_{\text{test}}^{(C_{i,l_k})}\big\}$ that pairs every $\mathbf{z} \in \mathcal{Z}_{\text{test}}^{(C_{i,l_k})}$ with its

[15] The formulation follows the pairwise contrastive loss of Hadsell *et al.* [HCL06]; multi-sample variants such as SupCon [K+] could be substituted without affecting the remainder of the pipeline.

negative cluster $\mathcal{N}(\mathbf{z})$. Each pair $\big(\mathbf{z}, \mathcal{N}(\mathbf{z})\big)$ indicates the nearest differently labeled cluster that could challenge the classification of $\mathbf{z}$ away from its positive cluster C_{i,l_k} (*i.e.* leading to a classification error so).

5.2.3.1 Proximity

Because diverse data points lying far from decision boundaries are typically simpler to classify, proximity (P) measures how close data points in $\mathcal{Z}_{\text{test}}^{(C_{i,l_k})}$ are to their negative clusters. A higher P suggests that $\mathcal{D}_{\text{test}}$ includes data points near differently labeled training data point, which are more likely to challenge the NIDS.

Formally, for each cluster C_{i,l_k}, we define:

$$d_{\text{test}_i} = \Big\{ \|\mathbf{z} - \boldsymbol{\mu}_{\mathcal{N}(\mathbf{z})}\| \;\Big|\; (\mathbf{z}, \mathcal{N}(\mathbf{z})) \in \text{PairNeg}\big(\mathcal{Z}_{\text{test}}, C_{i,l_k}\big) \Big\} \tag{5.5}$$

as the set of Euclidean distances from each $\mathbf{z} \in \mathcal{Z}_{\text{test}}^{(C_{i,l_k})}$ to the centroid of its negative cluster. To determine whether these encoded test data points lie farther from C_{i,l_k} than their training counterparts $\mathbf{z} \in \mathcal{Z}_{\text{train}}$ assigned to C_{i,l_k} (and thus pose a greater challenge to the NIDS), we analogously define d_{train_i}:

$$d_{\text{train}_i} = \Big\{ \|\mathbf{z} - \boldsymbol{\mu}_{\mathcal{N}(\mathbf{z})}\| \;\Big|\; (\mathbf{z}, \mathcal{N}(\mathbf{z})) \in \text{PairNeg}\big(\mathcal{Z}_{\text{train}}, C_{i,l_k}\big) \Big\} \tag{5.6}$$

Fig. 5.2 illustrates d_{test} (dashed) and d_{train} (solid) derived for the set $\mathcal{Z}_{\text{test}}^{(C_{1,\text{Benign}})}$. Using the

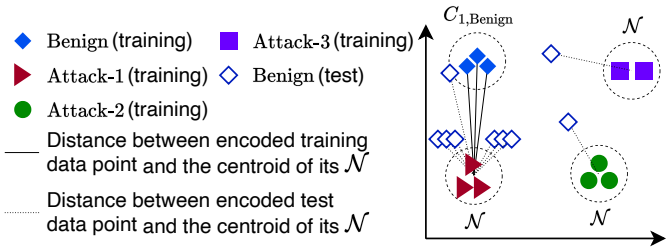


Figure 5.2: Training vs. Test Distances for $C_{1,\text{Benign}}$

empirical cumulative distribution functions F_{test_i} and F_{train_i} of d_{test_i} and d_{train_i} , we derive the maximum positive difference using the one-sided Kolmogorov–Smirnov statistic $\text{KS}_{C_{i,l_k}} \in [0, 1]$. A high value means that the distances in d_{test_i} are significantly smaller than those in d_{train_i} , so the encoded test data points are very close to their negative clusters. To capture the worst-case, we define P as the maximum KS value computed across all $\mathcal{Z}_{\text{test}}^{(C_{i,l_k})}$ (for $C_{i,l_k} \in \mathcal{C}$):

$$P \;=\; \max_{C_{i,l_k} \in \mathcal{C}} \Big(\text{KS}_{C_{i,l_k}} \Big). \tag{5.7}$$

5.2.3.2 Scarcity

Beyond proximity, a robust test approach must ensure that test data points appear across multiple decision boundaries where misclassifications may arise. Scarcity (S) evaluates how uniformly data points in $\mathcal{Z}_{\text{test}}^{(C_{i,l_k})}$ are spread across **all possible negative clusters**, Fig. 5.3 depicts an optimal scenario where each of the three possible negative clusters of $C_{1,\text{Benign}}$ is assigned an

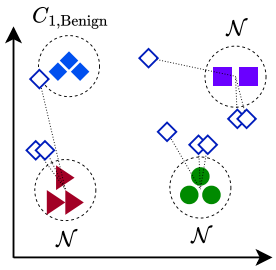


Figure 5.3: Optimal scarcity scenario

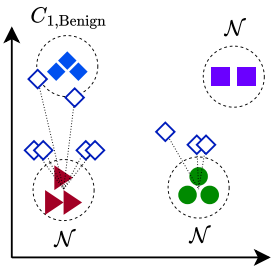


Figure 5.4: Non-optimal scarcity scenario

equal number of encoded test data points (three points each). In contrast, Fig. 5.4 shows a non-optimal scenario and reflects thus a poor scarcity.

Formally, we define the set of all possible negative clusters of C_{i,l_k} as:

$$\mathcal{N}_{\text{all}}(C_{i,l_k}) = \{\, C_{j,l_p} \in \mathcal{C} \mid j \neq i,\ l_p \neq l_k \,\} \tag{5.8}$$

For each $C_{j,l_p} \in \mathcal{N}_{\text{all}}\big(C_{i,l_k}\big)$, we define the number of associated data points as:

$$\text{PointCount}(C_{j,l_p}) = \Big|\, \{\, \mathbf{z} \mid (\mathbf{z}, \mathcal{N}(\mathbf{z})) \in \text{PairNeg}(\mathcal{Z}_{\text{test}}, C_{i,l_k}) \ \wedge\ \mathcal{N}(\mathbf{z}) = C_{j,l_p} \}\Big| \tag{5.9}$$

Normalizing $\text{PointCount}(C_{j,l_p})$ by $|\text{PairNeg}(\mathcal{Z}_{\text{test}}, C_{i,l_k})|$ yields the distribution R_{i,l_k} . The complement of the Gini Coefficient measures its uniformity:

$$R_{i,l_k} = \Big\{ \frac{\text{PointCount}(C_{j,l_p})}{\big|\text{PairNeg}\big(\mathcal{Z}_{\text{test}},\, C_{i,l_k}\big)\big|}, \forall\, C_{j,l_p} \in \mathcal{N}_{\text{all}}\big(C_{i,l_k}\big) \Big\}, G_{i,l_k} = 1 - \text{Gini}\big(R_{i,l_k}\big) \tag{5.10}$$

Values of G_{i,l_k} range from 0 to 1, where higher values indicate a more even distribution among negative clusters. S is the average of G_{i,l_k} for all $C_{i,l_k} \in \mathcal{C}$:

$$S = \frac{1}{|\mathcal{C}|} \sum_{C_{i,l_k} \in \mathcal{C}} G_{i,l_k}. \tag{5.11}$$

5.2.3.3 Diversity

A common testing requirement is to cover a wide range of test cases. Diversity (D) captures the variability among the data points in $\mathcal{Z}_{\text{test}}^{(C_{i,l_k})}$. A higher D reflects a $\mathcal{D}_{\text{test}}$ in which individual data points exhibit minimal redundancy and more fully cover the available feature space.

To quantify D, we employ the Vendi Score (V_{i,l_k}) [FD23], a metric that interprets diversity as the effective number of unique elements within a dataset. Unlike simple variance measures, the Vendi Score assesses the structure of the data through a three-step analysis:

(i) Similarity Measurement. First, a kernel-based similarity matrix $K \in \mathbb{R}^{N \times N}$ is constructed, where each entry $K_{ij} = k(\mathbf{z}_i, \mathbf{z}_j)$ represents the pairwise similarity between the test data points. As shown in the heatmaps of Fig. 5.5, a block-diagonal structure (large yellow squares) indicates high redundancy, where groups of points are identical to one another.

(ii) Eigenvalue Analysis. To determine if these similarities represent underlying redundancy, the eigenvalues $\lambda_1, \ldots, \lambda_N$ of the normalized matrix is computed. In simple terms, these values reveal the global structure of the encoded test data points by quantifying the magnitude

of distinct variations. If the encoded test set is redundant (e.g., many copies of the same traffic flow), the information is condensed into a few large eigenvalues, effectively ignoring the duplicates. Conversely, if the encoded test data points are distinct and varied, the information is distributed more evenly across the eigenvalues, reflecting a richer and more complex test set.

(iii) Entropic Scoring. Finally, we calculate the exponential of the Shannon entropy of these eigenvalues:

$$V_{i,l_k} = \exp\left(-\sum_{j=1}^{N} \lambda_j \ln \lambda_j\right) \tag{5.12}$$

This transformation converts the information distribution described in (ii) into an interpretable count. As illustrated in Fig. 5.5, this ensures the metric scales linearly with the effective number of distinct data points, rather than the raw test set size. Furthermore, as shown in Fig. 5.5, the use of eigenvalues ensures robustness to correlations, awarding higher diversity only when data points differ in meaningful, uncorrelated attributes (e.g., changing both shape and color) rather than superficial noise.

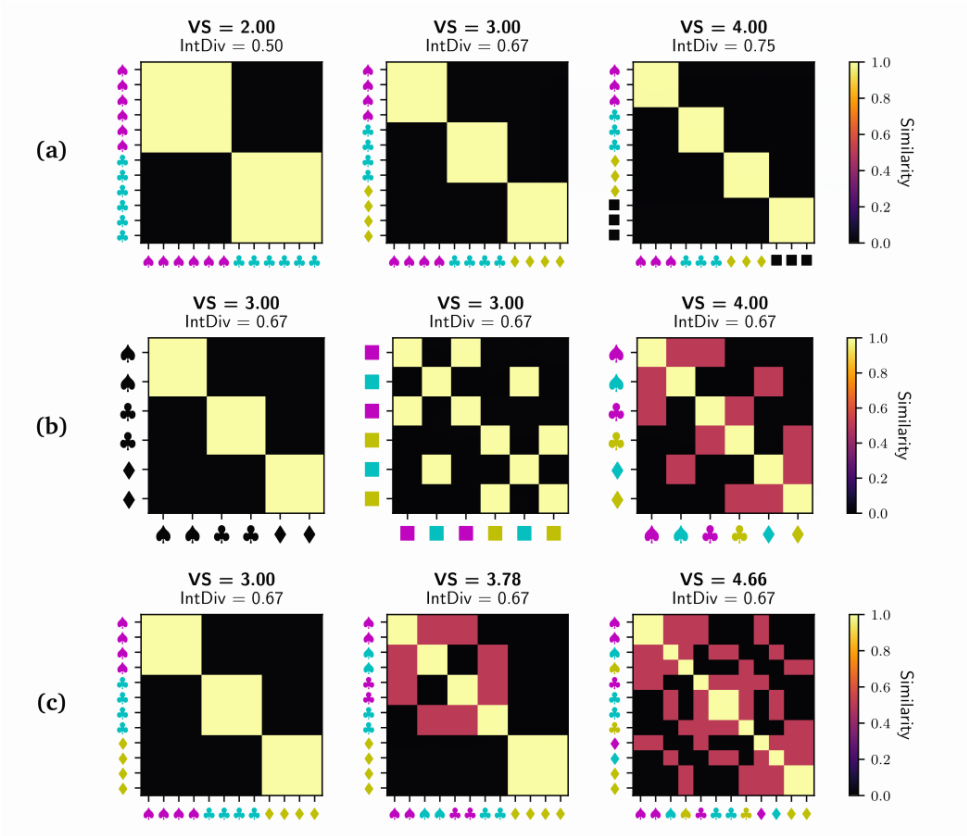


Figure 5.5: Visual intuition of the Vendi Score (VS) on synthetic data [FD23]. The axes represent individual data points (shapes and colors), and the heatmaps display their pairwise similarity. **(a)** Vendi Score interprets diversity as the effective number of unique elements: the score scales linearly ($2.00 \rightarrow 4.00$) with the number of distinct clusters (yellow blocks). **(c)** Vendi Score is robust to feature correlations: It increases ($3.00 \rightarrow 4.66$) as attributes (e.g., shape and color) become less correlated, rewarding complex variation over simple redundancy.

Since V_{i,l_k} ranges from 1 (minimal diversity) to the size of $\mathcal{Z}_{\text{test}}^{(C_{i,l_k})}$ (maximal diversity), we

min–max normalise it to $[0, 1]$. Finally, we average these normalised scores across all clusters to obtain :

$$D = \frac{1}{|\mathcal{C}|} \sum_{C_{i,l_k} \in \mathcal{C}} V_{i,l_k} \tag{5.13}$$

5.2.4 Test Set Targeted Augmentation

Reinforcement Learning (RL) is a machine learning paradigm in which an autonomous agent learns to make sequential decisions by interacting with a dynamic environment. At each time step, the agent observes the environment, takes an action, and receives a numerical reward signal. The overarching objective is to learn an optimal policy, a mapping from observations to actions, that maximizes the expected cumulative reward over time. While traditional RL assumes the agent has complete access to the environment's underlying state, modeled as a Markov Decision Process (MDP), many complex scenarios involve incomplete or noisy information. To handle this, the framework is extended to a Partially Observable Markov Decision Process (POMDP), where the agent must infer the optimal action from a compressed or restricted observation of the true state.

Building on this foundation, we formulate the augmentation of $\mathcal{D}_{\text{test}}$ as a POMDP and train a Deep Reinforcement Learning (DRL) agent. The agent relies on a policy π, a learnable rule that maps the current partial view of the latent space (the observation) to the next testbed configuration (the action). This agent interacts with the POMDP to maximize three explicit test set quality objectives: diversity, proximity, and scarcity. The POMDP is formally defined by the tuple $(\mathcal{S}, \mathcal{A}, P, R, \Omega, O, \gamma)$:

- **States $\mathcal{S}$.** Each state $s \in \mathcal{S}$ represents the complete structured latent space obtained in phase 1, which comprises both the clusters $C_{i,l_k} \in \mathcal{C}$ and the set $\mathcal{Z}_{\text{test}}$. In practice, because this full information can be very large and changes as $\mathcal{Z}_{\text{test}}$ grows, it remains unobserved by the agent.

- **Actions $\mathcal{A}$.** Each action $a \in \mathcal{A}$ specifies a traffic-generation configuration (e.g., bandwidth throttling, latency injection, or packet corruption) that the testbed uses to generate one new network traffic data point.

- **Transition Function P.** The hidden transition function P governs how the state evolves: when an action a is executed in state s, the resulting state s' reflects the updated structured latent space after the newly generated data point is encoded via f and appended to $\mathcal{Z}_{\text{test}}$.

- **Reward R.** After each action, the agent receives a scalar reward r quantifying the improvement in test set quality in terms of diversity D , proximity P, and scarcity S. Formally, $R = \frac{1}{\frac{w_D}{D} + \frac{w_P}{P} + \frac{w_S}{S}}$, where w_D, w_P, and w_S are tunable hyperparameters.

- **Observations Ω.** Each observation $o \in \Omega$ is a compressed summary of the structured latent space, comprising the cluster centroids $\boldsymbol{\mu}_{C_{i,l_k}}$ (for all $C_{i,l_k} \in \mathcal{C}$) and basic statistics (mean, min, max, variance, std) computed over $\mathcal{Z}_{\text{test}}$.

- **Observation Function O.** The function $O(o \mid s)$ maps the full state s to the aggregated observation o, encapsulating the compression of detailed latent information into the limited view available to the agent.

- **Discount Factor** $\gamma \in [0, 1]$**.** This parameter balances immediate versus future rewards in the RL process.

As illustrated in Phase 2 of Fig. 5.1, the augmentation process operates as a closed-loop cycle managed by the RL agent. During training, the agent first observes the cluster centroids and the summary of the initial $\mathcal{Z}_{\text{test}}$, denoted by o_0, while its policy π is randomly initialized. We implement this policy using Deep Reinforcement Learning (DRL), where a neural network approximates π. At each step t, the agent selects a traffic-generation configuration a_t (i.e., how the testbed should create one network traffic data point, labeled as "Trial and error configuration" in Fig. 5.1). In response, the configured testbed instantiates real network traffic under those conditions, yielding a new data point. It is then encoded, $f(\mathbf{x})$, and appended to $\mathcal{Z}_{\text{test}}$.

Formally, this interaction realises the POMDP dynamics. The transition function $P(s_{t+1} \mid s_t, a_t)$ updates the latent state to incorporate the newly encoded data points; the observation function $O(o_{t+1} \mid s_{t+1})$ compresses that state into the next partial view fed to the agent.

The updated $\mathcal{Z}_{\text{test}}$ is summarized to yield, with the cluster centroids, the new partial observation o_{t+1} , and the test set is evaluated using the three quality metrics to compute the reward r_t. Upon receiving the transition tuple $\big(o_t, a_t, r_t, o_{t+1}\big)$, the agent updates π with the aim of increasing future rewards. This process repeats for multiple steps, with each step adding one new encoded data point to $\mathcal{Z}_{\text{test}}$. If a target test set quality is reached or a maximum step budget is met, the current episode terminates and the environment resets to the original test set (or a fresh copy), after which a new episode begins. Over multiple episodes, π converges to a better policy π^*, learning a strategy for configuring the testbed to generate network traffic flows that progressively enhance the diversity, proximity, and scarcity of the test set.

Once the training phase is complete, π^* can be reused during testing to augment either the same $\mathcal{D}_{\text{test}}$ or another NIDS dataset. In the latter case, the preliminary phase and phase 1 must first be carried out to establish a compatible structured latent space for that new NIDS dataset; the pre-trained agent can then directly apply π^* to guide the generation of additional network traffic.

5.3 Conclusion

This chapter introduced TATA, a novel, model-agnostic framework designed to address the critical research challenge of assessing and improving the quality of NIDS benchmark test sets. We began by formulating two central research questions focused on developing robust assessment metrics and guiding targeted, realistic data augmentation. TATA provides an integrated solution to both.

Our methodology is founded on a three-phase pipeline. First, a contrastive autoencoder constructs a structured latent space from the training data, creating a model-agnostic representation where clusters approximate decision boundaries. Second, within this space, we introduced a suite of three complementary metrics—Diversity, Proximity, and Scarcity—to perform a fine-grained assessment of the test set's quality. Together, these metrics evaluate the breadth, difficulty, and uniformity of the test data, moving beyond the narrow focus of prior art. Finally, TATA employs a reinforcement learning agent that interacts with a network testbed. Guided by the continuous feedback from our assessment metrics, the agent learns to generate protocol-compliant traffic that specifically targets and remedies the identified deficiencies in the test set.

By design, TATA overcomes several key limitations of existing approaches. Its operation in a data-centric latent space removes the dependency on model internals, making it a true black-box methodology applicable to any NIDS, including proprietary systems and non-neural models. The

combination of its three assessment metrics provides a more holistic and actionable measure of test-set quality than previously possible.

Having detailed the theoretical and operational framework of TATA, the subsequent chapter will be dedicated to its empirical validation. We will present a series of experiments designed to demonstrate the efficacy of our assessment metrics and the ability of the targeted augmentation process to systematically enhance NIDS test sets for more robust evaluation.

Chapter 6

Evaluation of TATA

Contents

6.1 Introduction 107
6.2 TATA Evaluation Environment and Implementation Details . . 108
6.2.1 Datasets 108
6.2.2 NIDS Models 108
6.2.3 Contrastive Autoencoder 109
6.3 Preliminary Phase: Evaluating the Contrastive Autoencoder . 109
6.4 Phase 1: Test Set Assessment Metrics in Practice 110
6.4.1 Diversity 110
6.4.2 Proximity 110
6.4.3 Scarcity 112
6.5 Phase 2: Targeted Augmentation Evaluation 112
6.6 Datasets Analysis 116
6.7 Discussion of Results 117
6.8 Conclusion 119

6.1 Introduction

In the previous chapter we presented TATA, a three-phase pipeline that (i) projects network-traffic data into a discriminative latent space, (ii) diagnoses the weaknesses of a benchmark test set through the *diversity*, *proximity*, and *scarcity* metrics, and (iii) corrects those weaknesses by controlling a traffic-generation testbed with reinforcement learning. This chapter provides an experimental evaluation, showing how well each component of TATA performs in practice and how the pipeline as a whole answers the two research questions:

- **RQ1:** How can we assess the quality of test sets for NIDS models to ensure a robust evaluation of their detection capabilities?
- **RQ2:** How can we systematically augment these test sets to better evaluate the robustness and real-world applicability of NIDS models?

Section 6.3 evaluates the preliminary phase, which constructs the discriminative latent space, by comparing the contrastive auto-encoder with a standard encoder–decoder trained exclusively on reconstruction loss across ten random splits of IDS2017 and IDS2018. Clustering quality is assessed with silhouette scores, k-means accuracy, and two class-sensitive indices, the Adjusted Rand Index (ARI) and the Normalized Mutual Information (NMI), both of which measure how closely the resulting clusters align with the ground-truth classes.

Section 6.4 addresses RQ1. By systematically manipulating the test set, we isolate the influence of each individual metric. Specifically, we evaluate diversity by adding new traffic classes, assess proximity by selecting ranked subsets of boundary points, and examine scarcity by performing controlled redistributions of misclassifications. Results are gathered for three detectors (RF, SVM, and DNN) to study how these metrics relate to overall detection performance.

Section 6.5 tackles RQ2. An offline RL agent, trained on a buffer of 500 000 transitions collected from IDS2018, is deployed to augment IDS2017. We compare TATA's targeted augmentation against three baselines: a randomly acting agent, the NetShare GAN [YLJ+22], and a neuron-coverage-guided approach [KFY19]. We measure both the resulting changes in the quality metrics and the corresponding variations in detection performance.

Finally, Section 6.6 broadens the scope by applying the assessment metrics specifically to the test partitions of fourteen public traffic-classification datasets, ranging from NSL-KDD (2009) to ToN-IoT (2020). The aim of this longitudinal analysis is to determine whether the complexity of these test sets has progressively increased over the years.

6.2 TATA Evaluation Environment and Implementation Details

All experiments reported in this part were run on a dedicated, bare-metal server equipped with Ubuntu 22.04.3 LTS, an Intel Xeon Gold 6258R CPU, 500 GB RAM, and a NVIDIA RTX A6000 GPU (48 GB VRAM). The complete TATA pipeline (data preparation, contrastive auto-encoder training, metric computation, and reinforcement-learning augmentation) was implemented in `Python 3.10` using `PyTorch 2.1` (CUDA 12.1) for deep learning, `scikit-learn 1.4` for classical models, and `d3rlpy` for RL algorithms and training. Source code, reproducibility scripts, and configuration files are publicly available.[16]

6.2.1 Datasets

We evaluate TATA on the revised versions IDS2017 and IDS2018. Unless stated otherwise, each dataset is split *once* into stratified training, validation, and test partitions in a 60/20/20 ratio: $\mathcal{D}_{\text{train}}$, $\mathcal{D}_{\text{val}}$, and $\mathcal{D}_{\text{test}}$. Numerical features are standardised by subtracting the mean and dividing by the standard deviation computed on $\mathcal{D}_{\text{train}}$ only, and then applied to $\mathcal{D}_{\text{val}}/\mathcal{D}_{\text{test}}$ to avoid leakage. Categorical fields (when present) are one-hot encoded. We use macro-F1 as the main metric for model selection and reporting, and we also report per-class precision and recall.

6.2.2 NIDS Models

To demonstrate TATA's model-agnostic design, we train three multi-class detectors frequently used in NIDS studies: a Random Forest (RF), a Support Vector Machine with RBF kernel (SVM), and a small feed-forward neural network (DNN). The DNN has three fully connected ReLU layers (128 units each) and a softmax output. These models are widely used in NIDS

[16] `https://gitlab.inria.fr/oanser/tata`

research, showing strong results on IDS2017 and IDS2018 [Owe23]. We select hyper-parameters by random search on $\mathcal{D}_{\text{val}}$, keeping the configuration that maximizes macro-F1.

The random search explores 30 candidate configurations per model:

- **RF:** number of trees $\{256, 512, 768, 1024\}$, maximum depth $\{12, 20, 32, None\}$, and minimum samples per leaf $\{1, 2, 4\}$.
- **SVM (RBF):** cost $C \in \{10^{-2}, 10^{-1}, 1, 10, 10^2\}$ and kernel width $\gamma \in \{10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}\}$, with class weights inversely proportional to class frequency.
- **DNN:** learning-rate $\{10^{-4}, 5\times10^{-4}, 10^{-3}, 5\times10^{-3}\}$, batch size $\{64, 128, 256\}$, and dropout $\{0, 0.2, 0.5\}$ applied after each hidden layer.

The configuration that maximizes macro-F1 on $\mathcal{D}_{\text{val}}$ is retained for the main experiments. For RF and SVM we use scikit-learn defaults except for the sampled hyper-parameters above. The DNN is trained with cross-entropy on standardised inputs; the checkpoint that achieves the best macro-F1 on $\mathcal{D}_{\text{val}}$ is selected for testing. Training uses the GPU for DNN/auto-encoder models and the CPU for scikit-learn models.

6.2.3 Contrastive Autoencoder

The contrastive autoencoder is a multilayer perceptron selected after a grid search over 216 combinations of candidate architectures and training hyperparameters. The search space varied the network depth ($\{2, 3, 4\}$ hidden layers), width ($\{32, 64, 128\}$ neurons per layer in descending order, e.g., 128−64−32), latent dimension ($\{2, 3, 5, 8\}$), learning rate ($\{10^{-2}, 5\times10^{-3}, 10^{-3}\}$), batch size ($\{64, 128, 256\}$), contrastive margin m ($\{5, 10, 15\}$), and contrastive weight λ ($\{0.05, 0.1, 0.5\}$). To favor compactness without sacrificing cluster quality, we selected the smallest model whose silhouette-guided k-means accuracy remained within 1% of the global maximum. This optimal configuration consists of three fully connected layers with 64, 32, and 16 neurons, ReLU activations, and a three-dimensional latent space. The model is trained for 250 epochs using a batch size of 128, a learning rate of 10^{-3}, a contrastive margin of $m = 10$, and a regularization factor of $\lambda = 0.1$.

6.3 Preliminary Phase: Evaluating the Contrastive Autoencoder

The goal of the preliminary phase is to construct a structured latent space that approximates the decision boundaries that any downstream NIDS model might produce, yet remains entirely model-agnostic. To verify that the contrastive auto-encoder builds the intended structured latent space, we compute clustering metrics and compare its embeddings with those produced by a standard auto-encoder trained solely on reconstruction loss. Both models are trained on ten independent, stratified 60 % splits of IDS2017 and IDS2018, each obtained with a different random seed, and evaluated with four clustering metrics:

- Silhouette score: measures intra-cluster cohesion vs. inter-cluster separation (1 is best);
- k-Means accuracy (KM acc.): percentage of points whose cluster label matches the dominant ground-truth label of that cluster;
- Adjusted Rand Index (ARI): measures the agreement between predicted and true labels, mathematically corrected so that random clustering scores near zero and perfect alignment scores 1 [HA85];

- Normalized Mutual Information (NMI): mutual dependence between labels and clusters, normalized to $[0, 1]$ [SG02].

Table 6.1 shows that the contrastive auto-encoder consistently outperforms the vanilla model. Its silhouette score approaches 1, indicating tight, well-separated clusters, and its KM accuracy exceeds 99 %, meaning almost every cluster is pure with respect to the ground-truth label. ARI and NMI of 1.00 further confirm that the learned clusters align almost perfectly with class labels, whereas the vanilla model exhibits substantial label mixing (ARI ≈ 0.5, NMI ≈ 0.7) and high variance across seeds.

Table 6.1: Clustering quality of the contrastive vs. vanilla auto-encoder (mean ± standard deviation over ten random splits).

Metric	IDS2017		IDS2018	
	Contrastive	**Vanilla**	**Contrastive**	**Vanilla**
Silhouette score	0.96 ± 0.00	0.73 ± 0.05	0.98 ± 0.00	0.76 ± 0.02
KM accuracy (%)	99.90 ± 0.02	59.81 ± 6.76	99.99 ± 0.01	62.88 ± 6.25
ARI	1.00 ± 0.00	0.46 ± 0.09	1.00 ± 0.00	0.56 ± 0.06
NMI	1.00 ± 0.00	0.66 ± 0.04	1.00 ± 0.00	0.72 ± 0.04

Takeaway: The contrastive autoencoder yields a highly discriminative and well-structured latent space, outperforming a vanilla autoencoder.

6.4 Phase 1: Test Set Assessment Metrics in Practice

Having introduced each metric's rationale in the previous chapter, we now address RQ1 by examining whether their values vary in alignment with the detector performance of the implemented NIDSs. We use a single 60/20 $\mathcal{D}_{\text{train}}$– $\mathcal{D}_{\text{test}}$ stratified split on the IDS2017 dataset (IDS2018 yields similar observations). Because this split alone gives each metric a single value, we selectively manipulate $\mathcal{D}_{\text{test}}$ to produce varying test set qualities, as detailed in the subsections below. We measure correlation using Pearson's r for linear relationships and Spearman's ρ for rank-based (monotonic) trends, so both absolute differences and relative orderings are captured.

6.4.1 Diversity

We investigate whether including additional traffic types (labels) in the NIDS evaluation (an inherently more challenging scenario) correlates with higher diversity. To do this, we generate all sub-test sets from IDS2017's $\mathcal{D}_{\text{test}}$ corresponding to each label combination of size k (k ranges from 1 to 9 since IDS2017 includes 9 types of traffic).

We then compute and plot the resulting diversity values in Fig. 6.1, observing a perfect correlation (Pearson's $r = 0.99$, Spearman's $\rho = 0.98$) between the number of labels and mean diversity. As k grows from 1 to 9, diversity rises monotonically from near-minimal to a maximum of 0.1209.
Takeaway: The diversity metric captures the increased evaluation challenge that results from adding more label types, which increases the range of traffic types the NIDS must handle.

6.4.2 Proximity

To assess proximity, we compute each encoded test data point's distance to its negative cluster. We rank these distances from largest to smallest and split the ordered list into 100 cumulative

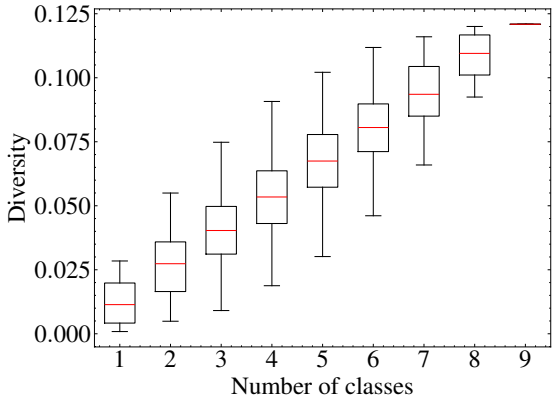


Figure 6.1: Box plots of the diversity metric for sub-test sets of size $k = 1, \ldots, 9$

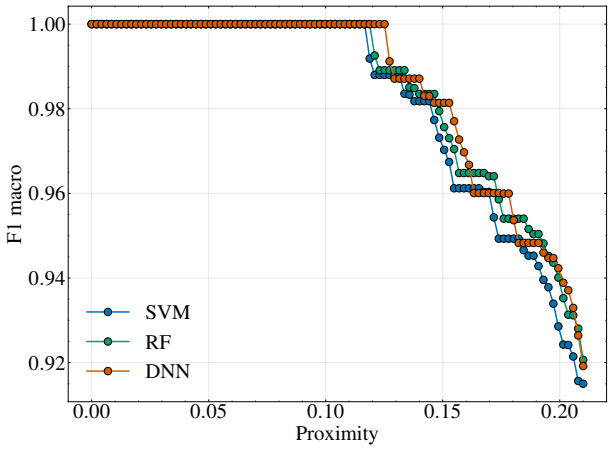


Figure 6.2: Effect of Proximity on NIDS Performance

sub-test sets: the first contains the top 1 %, the second the top 2 % (including the first), and so on. As we move from sets with the largest distances to those with smaller ones, the average distance decreases, thereby increasing the proximity metric. Using each sub-test set, we evaluate whether higher proximity levels challenge the NIDS, as indicated by increased error.

In Fig. 6.2, each circular marker represents a sub-test set, with its average proximity (x-axis) plotted against the macro-F1 of the three NIDS (RF, DNN, SVM). All three NIDS maintain macro-F1=1.0 across more than half the sub-test sets (proximity $\leq$ 0.11–0.14), but macro-F1 gradually declines (to about 0.92 for RF/DNN and 0.91 for SVM) as more borderline data points appear. We observe a strong negative Spearman correlation ($\rho = -0.90 \pm 0.01$) and a similarly negative Pearson correlation ($r = -0.86 \pm 0.01$), both averaged across the three NIDS, confirming that higher proximity corresponds to lower macro-F1. Their nearly identical trajectories (pairwise correlations Pearson > 0.98, Spearman > 0.97) indicate that proximity is agnostic to the ML model.

Takeaway: Proximity captures the difficulty posed by borderline test data points, as higher proximity values coincide with lower NIDS performance.

6.4.3 Scarcity

As with proximity, we investigate whether scarcity correlates with $\mathcal{D}_{\text{test}}$'s difficulty and thus affects NIDS performance. Defined in Sec. 5.2.3.2, scarcity measures how uniformly encoded test data points distribute across all possible negative clusters. To produce sub-test sets with different scarcity levels, we run each NIDS on $\mathcal{D}_{\text{test}}$, record misclassified data points, and map them to their latent representations. We then systematically redistribute these misclassified instances among the available negative clusters in 10 steps, transitioning from a less-uniform (clustered) to a more-uniform (dispersed) arrangement. As with proximity, we therefore create sub-test sets with an increasing scarcity and evaluate how this impacts the NIDS performance.

Because our objective is to create a wide range of types of misclassification errors (*i.e.* mixing

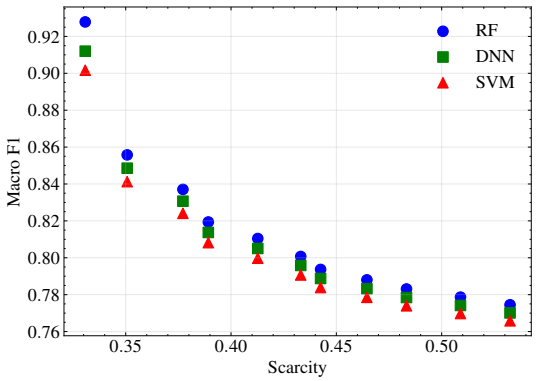


Figure 6.3: Effect of Scarcity on NIDS Performance

different labels), macro-F1 is also considered, correcting imbalance effect of a particular label. In overall, a strong negative Spearman correlation ($\rho = -1 \pm 0$, averaged across models) and a strong negative Pearson correlation ($r = -0.8885 \pm 0.0055$) clearly indicate that higher scarcity corresponds to a lower macro-F1. In details, Figure 6.3 shows that, after the first redistribution step, macro-F1 shows a sharp decline. Performance then continues to decrease more gradually and monotonically; for instance, the macro-F1 of the RF-based NIDS falls below 0.78 when the scarcity level reaches 0.53.

Takeaway: A more uniform spread of misdetection data points across decision boundaries increases scarcity and decreases the macro-averaged F1-score, confirming scarcity's relevance as a difficulty indicator.

6.5 Phase 2: Targeted Augmentation Evaluation

Scenario Setup

In this second phase, TATA focuses on augmenting $\mathcal{D}_{\text{test}}$ with Benign flows only, as a proof of concept. The RL agent relies on a two-host testbed (client and server) exchanging Benign SSH traffic (a protocol present in IDS2017 and IDS2018). To avoid interference from background processes and to keep the network path fully under control, the two SSH endpoints are hosted in separate KVM/QEMU virtual machines on the same bare-metal server described in Section 6.2. Each VM runs Ubuntu 22.04, is provisioned with 2 vCPU, 8 GB RAM, and boots from a read-only qcow2 snapshot so that every episode starts from a clean state. The two VMs communicate through a bridged virtual NIC that is isolated from other host traffic, while a separate management bridge handles orchestration and logging. This separation ensures that the experimental

link carries only the synthetic SSH flows required for augmentation and that no background processes perturb the measured network conditions.

RL Training

RL ideally operates at millisecond timescales, allowing the collection of numerous transition tuples (o_t, a_t, r_t, o_{t+1}) for effective learning, but our full pipeline (traffic generation, flow conversion, encoding with f and metrics computation) requires seconds, making a conventional online loop infeasible. Thus, we employ offline RL, training on a static dataset of transitions (o_t, a_t, r_t, o_{t+1}) derived from IDS2018 while reserving IDS2017 for final testing. We omit the reverse scenario because IDS2018's larger cluster set $\mathcal{C}$ exceeds what an agent trained on IDS2017 can handle as observation input.

To construct these transition tuples, we first split IDS2018 into $\mathcal{D}_{\text{train}}$ and an initial $\mathcal{D}_{\text{test}}$ and augment the latter iteratively with a random traffic-generation configuration (action) on the SSH traffic, drawn from uniform parameter ranges (forming the action set a_t): loss [5% , 10%], jitter [4 ms, 10 ms], delay [10 ms, 40 ms], duplication [0.1%, 5%], corruption [0.1%, 10%], reordering [0.1%, 50%], and correlation [50%, 100%]. The action is applied with the Linux `tc qdisc` command on the client side while running a predefined SSH scenario. This scenario is a fixed shell script that simulates a realistic SSH session on the server:

- Generates two large files (1 MB and 2 MB) filled with random bytes;
- Creates a sub-directory and performs ten rounds of file copy, append, and directory listing operations;
- Issues compound commands such as `find`, `du`, and `tar` to stress CPU and I/O;
- Appends additional content to the first large file and mirrors a growing log file;
- Alters file permissions and produces human-readable listings;
- Removes selected files and the sub-directory to emulate clean-up.

Sleep intervals of 0.2–0.3 s between blocks introduce timing variability, and all commands run non-interactively over an SSH connection authenticated with a pre-installed RSA key.

After each action, we convert the resulting network traffic into flow(s) using CICFlowMeter, after which we apply our pipeline (metric computation and reward calculation with weights $w_P = w_S = w_D = 1$). If 5, 000 steps have elapsed or the reward exceeds 0.9, we reset the initial $\mathcal{D}_{\text{test}}$ to begin a new episode. Across these episodes, we collect roughly 500, 000 transitions without any filtering, deliberately retaining failed episodes to ensure the offline RL algorithms (CQL [KZTL20] and TD3+BC [FG21], chosen for continuous-action support and for penalizing out-of-dataset actions to remain within known transitions, and tuned via grid search) encounter a diverse range of outcomes.

Evaluation

After training, we evaluate each RL agent for 20 episodes on IDS2017, using different random seeds to define an 60–20 train–test split each time. During these evaluations, the agent manipulates traffic configurations on the same SSH testbed, but no further policy updates or reward signals occur. Newly generated flows are labeled benign and added to the IDS2017 test set. Finally, we measure how this augmentation affects (i) our test-set quality metrics and (ii) NIDS

detection performance (macro-averaged precision and recall). We report mean and standard deviation over the 20 episodes. Three baselines are considered:

- **Random Agent:** The agent selects traffic-shaping parameters (loss, jitter, delay, etc.) at random, using no learned strategy.
- **GAN-Based Method (Netshare[YLJ+22]):** Netshare employs a GAN to synthesize network traffic. Although it attempts to mimic real-world distributions, its generated flows may not fully capture the complexity of realistic network behavior.
- **Augmentation guided with NC [PCYJ17]:** This variant uses the TATA's offline RL pipeline but replaces the default reward with a neuron-coverage metric from the DNN-based NIDS. During training, after each newly generated traffic flow, we measure the fraction of activated neurons (e.g., 70% activation yields a reward of 0.7).

Results

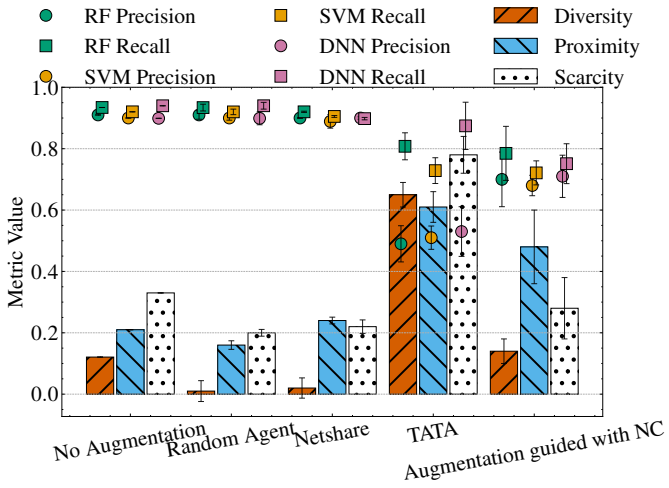


Figure 6.4: Impact of TATA and Baselines on Test-Set Quality Metrics and Detection Performance

We compare only methods with publicly available, working code that can be adapted to NIDS; others, such as [RHJT21a], focus on image-based augmentation and are therefore omitted.

Fig. 6.4 summarizes our results, plotting each augmentation method alongside the test set (no augmentation) on the x-axis.

Results are focused on CQL-based findings, as TD3+BC produces similar outcomes that do not change our conclusions. The y-axis shows both quality metrics (diversity, proximity, and scarcity) and NIDS performance metrics (macro-averaged precision and recall), all normalized to [0,1].

Both the Random agent and Netshare dramatically reduce diversity, driving it close to zero. However, the impact on proximity is negligible (the Random agent lowers it slightly, while Netshare raises it slightly) whereas scarcity declines for both. Together, these changes have nearly no effect on NIDS performance. In contrast, TATA increases proximity by about 190%, scarcity by roughly 136%, and diversity by around 437% , whereas the augmentation guided with NC focuses on proximity, increasing it by approximately 129%. This limitation arises because

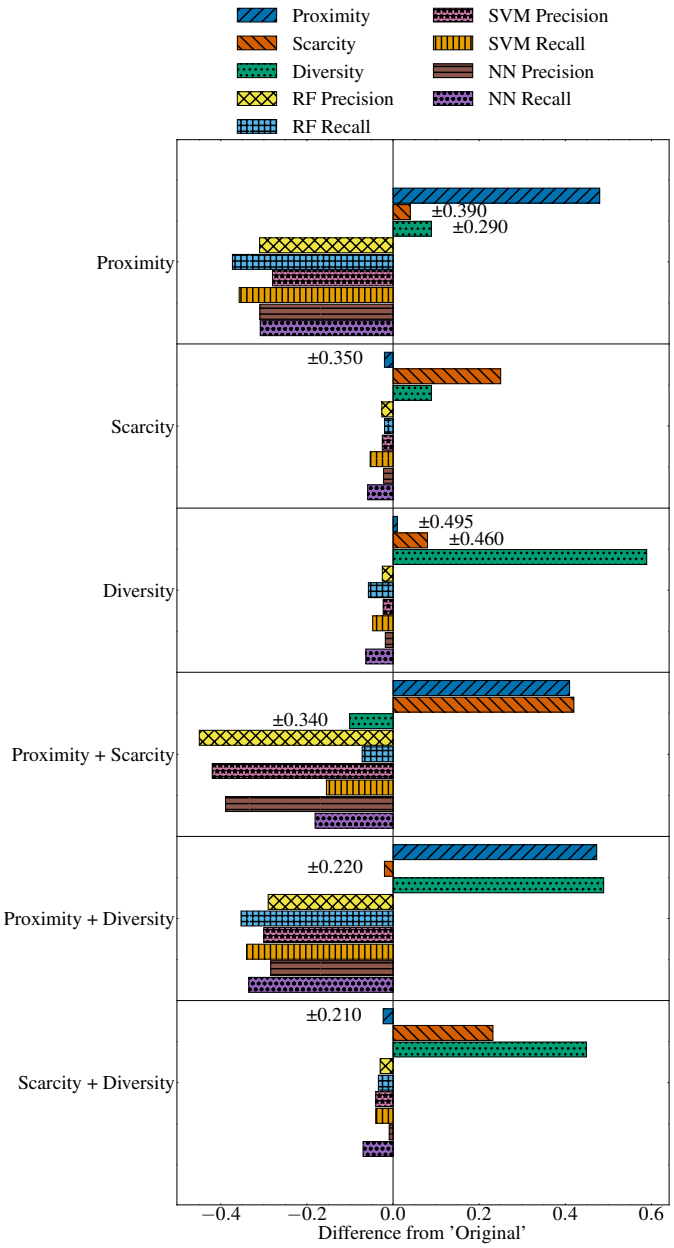


Figure 6.5: Impact of Partial Metric Combinations on Test-Set Quality and NIDS Performance

neuron coverage targets data points near negative clusters, those most stressful to the NIDS, while overlooking diversity and scarcity. These differences also influence precision and recall. For example, for the RF-based NIDS, TATA lowers precision by roughly 46 %, compared with about 23 % for NC, while recall drops by around 13 % with TATA versus 16 % with NC. We see that scarcity, by distributing misclassified points more widely, strongly affects precision (tied to false positives).

Takeaway: TATA comprehensively stresses the NIDS through diversity, proximity, and scarcity, creating a more challenging test set (thus lower precision/recall) than augmentation guided with NC, which focuses solely on proximity.

To have a higher understanding of the importance of the quality metrics, the reward is now calculated from a single metrics or any combination of two of them resulting in 6 different cases reported in Fig. 6.5 where D, P, and S stands for diversity, proximity, and scarcity respectively. Horizontal bars, centered at zero, illustrate how much each configuration improves or degrades performance relative to the original set. Combinations that include proximity $(P), (P, S), (P, D)$ offer the greatest challenge to the NIDSs, while the others have minimal impact. This is because proximity targets the decision boundary of negative clusters, where the NIDS struggles most with detection. For example, with the RF-based NIDS, in the (P, S) combination, recall declines by about 8% (from 0.934 to 0.862) and precision by about 49% (0.910 to 0.460), whereas in (S, D), they fall by roughly 4% (0.934 to 0.900) and 3% (0.910 to 0.880), respectively.

Takeaway: Proximity emerges as the critical factor in stressing the NIDSs, whereas scarcity and diversity offer complementary effects by covering all possible decision boundaries and avoiding redundant test cases, together expanding test coverage.

6.6 Datasets Analysis

We applied TATA's metrics to various networking-related datasets, including intrusion detection and other traffic-classification benchmarks, to assess their test sets over time.

The datasets considered are: NSL-KDD (2009), CTU-13 (2011), ISCX IDS 2012, UNSW-NB15 (2015), CIC-UNSW (2015), ISCX Tor (2016), VPN-NonVPN (2016), CIC-IDS 2017, CSE-CIC-IDS 2018, Bot-IoT (2018), CIC-DDoS 2019, and ToN-IoT (2020).

Having already detailed IDS2017 and IDS2018 in the previous chapter, we give in the following a brief overview of the other datasets:

- NSL-KDD (2009) — a curated refinement of DARPA'98/ KDD'99 that removes redundant records and balances difficulty across samples. It retains the classic four attack categories (DoS, Probe, R2L, U2R) with 41 engineered features, and provides predefined train/test splits.

- CTU-13 (2011) — thirteen real botnet scenarios captured in a lab network at CTU, labelled as *botnet*, *normal*, or *background*. Data are provided as NetFlow and PCAP; the flows are highly imbalanced and include diverse malware families.

- ISCX IDS 2012 — a week-long, 24-host laboratory capture with benign and malicious sessions over common protocols (HTTP, SMTP, SSH, FTP). The dataset contains about 2.5M flows and four attack types, enabling temporal analyses across consecutive days.

- UNSW-NB15 (2015) — generated in a cyber-range using IXIA traffic generators and real services. It covers nine attack families (e.g., Fuzzers, Reconnaissance, Exploits, DoS/DDoS, Generic, Backdoors, Shellcode, Worms) with 49 flow features extracted by Argus/Bro tools.

- CIC-UNSW (2015) — an augmented variant with CIC-style flow extraction and an expanded feature set (∼80 attributes), intended to harmonise representations with later CIC corpora.

- ISCX Tor (2016) — traffic from five users with and without the Tor network, labelled by application class. It targets encrypted-traffic identification using 28 statistical flow features.

- VPN-NonVPN (2016) — multi-application traffic (chat, email, audio/video, file transfer, browsing) captured with and without a VPN tunnel. The 84-feature representation supports studies on encrypted vs. cleartext discrimination.

- Bot-IoT (2018) — a large-scale IoT testbed with benign device traffic and botnet-generated attacks (e.g., DoS/DDoS, reconnaissance, information theft). It contains tens of millions of records and 46 flow features; pronounced class imbalance is typical.

- CIC-DDoS 2019 — a dedicated DDoS corpus covering multiple vectors and reflection/amplification protocols (e.g., UDP, SYN, NTP, LDAP). It provides about 80 flow features and protocol-specific attack scenarios.

- ToN-IoT (2020) — an IoT/IIoT environment with heterogeneous data sources (network flows, device telemetry, host logs). The network sub-dataset comprises ∼0.48M flows, nine attack types, and 42 features, supporting cross-domain evaluations.

Using ten 60/20 training–test splits, Figure 6.6 plots our metrics. As shown, proximity remains fairly stable (mostly 0.2–0.3), though early versions of IDS2017 and IDS2018 (with labeling errors) reach around 0.5. Diversity shows a broader range, from near zero to about 0.5, without a clear monotonic trend. Scarcity largely tracks proximity until CIC-IDS2017 (errors present), then aligns more with diversity and fluctuates more. Across all ten splits, each metric exhibits minimal, often negligible, standard deviation.

These findings suggest that, despite increasingly complex testbed designs over time (including more attacks, applications, and intricate topologies), the core quality of these datasets' test sets remains consistent, and their inherent difficulty has not substantially increased. In contrast, widely used image classification benchmarks, MNIST and CIFAR-10, exhibit higher complexity. Notably, we adapted our contrastive autoencoder with convolutional layers for image data.

Takeaway: Even with recent advances in dataset design, reference datasets differ widely in difficulty across application domains, and our metrics reveal that most NIDS test sets still pose only a moderate challenge, highlighting the network-security community's ongoing shortage of truly demanding, well-suited datasets.

6.7 Discussion of Results

The experimental validation of TATA offers several key insights that extend beyond the immediate performance metrics. This discussion interprets the significance of our findings, connecting them to the foundational research questions and the broader challenges in NIDS evaluation.

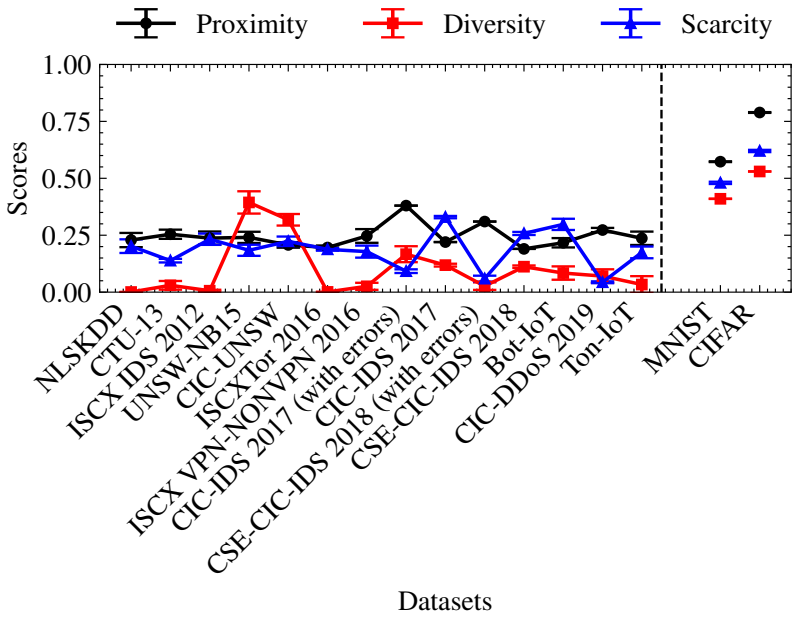


Figure 6.6: Test Sets Assessment of NIDS Datasets

A central implication of our work is the demonstrated insufficiency of single-metric optimization for robust test-set augmentation. The comparison between TATA and the neuron-coverage-guided baseline was particularly revealing. While the NC-guided agent successfully improved proximity, its narrow focus created a test set challenging in only one dimension. TATA's multi-objective approach, in contrast, produced a benchmark that was simultaneously diverse, challenging, and comprehensive, inducing a more significant stress test on the NIDS models. This finding suggests a paradigm shift: effective robustness evaluation does not stem from finding a single hardest type of data point, but from ensuring the test set covers a wide and varied landscape of challenges.

The success of the contrastive autoencoder as the foundation for this framework is also significant. It confirms that a model-agnostic, data-centric latent space can serve as a highly effective proxy for the decision boundaries of diverse downstream models. This result directly addresses the critical limitation of prevailing white-box assessment techniques. By decoupling test-set assessment from model internals, TATA provides a practical and transferable methodology for the NIDS community, especially in scenarios involving proprietary systems or non-neural models where methods like Neuron Coverage are infeasible.

Ultimately, these results demonstrate that merely increasing the size of a test set does not guarantee a better evaluation. To reliably test a model's robustness, data augmentation must specifically target the inherent weaknesses of the dataset. By guiding traffic generation with the diversity, proximity, and scarcity metrics, TATA creates significantly harder test sets compared to unguided or single-metric baselines. Furthermore, because it generates protocol-compliant traffic without requiring access to the model's internal architecture, this targeted approach offers a practical way to bridge the gap between static public benchmarks and real-world network deployments.

6.8 Conclusion

This chapter evaluated the TATA framework, demonstrating that combining a discriminative latent space, dataset-centric quality metrics, and reinforcement learning provides a practical, model-agnostic approach to robust test set augmentation. The current system has a few practical limitations that we plan to address in future research.

First, the current pipeline is computationally intensive. Assessing a new dataset requires training a contrastive autoencoder entirely from scratch, conducting extensive hyperparameter grid searches, and performing iterative clustering and centroid assignment. To significantly reduce this computational overhead, future iterations will explore utilizing a foundational pre-trained encoder, requiring only the fine-tuning of lightweight adapters for new datasets.

Second, the traffic generation scope exhibits limited diversity. The current augmentation phase functions as a proof of concept that exclusively produces benign SSH flows. Future work will expand the testbed to encompass additional benign protocols as well as active attack traffic. This expansion will likely require a multi-agent reinforcement learning design, where specialized agents manage the generation of specific traffic types.

Third, the reinforcement learning agent currently operates under a fixed-cluster assumption. Because the observation space is structurally tailored to a predefined number of latent clusters, the agent cannot dynamically adapt to new datasets that present a vastly different or highly variable structural topology. Evolving the state representation to handle an arbitrary number of clusters is a necessary step for broader algorithmic applicability.

Finally, the evaluation of the RL-driven augmentation was restricted primarily to the IDS2017 and IDS2018 benchmarks. The agent's ability to generalize its traffic-shaping policy to datasets with fundamentally different feature distributions, such as UNSW-NB15 or Bot-IoT, remains unverified. Assessing TATA's performance across a wider, highly heterogeneous suite of NIDS benchmarks therefore constitutes a central priority for our ongoing work.

Conclusion

Problem Statement

Machine-learning-based network intrusion detection has been the subject of intense research for more than two decades. Many works have proposed new architectures, feature sets and training strategies, often demonstrating excellent performance on public benchmarks. However, the dominant use cases have focused on relatively static settings: a detector is tuned once on a fixed dataset, evaluated on a predefined split, and then conceptually "frozen" for deployment. In contrast, real networks are dynamic. Traffic evolves with new applications and usage patterns, attackers adapt their behavior, and public intrusion datasets only approximate operational conditions. This gap between static evaluation and dynamic reality motivates the need for NIDS that are not only accurate on a given benchmark, but also robust in how they are configured and in how their performance is assessed before deployment.

In this thesis, we began by examining in detail the position of ML-based NIDS within the broader defense stack. We first contrasted classical rule-based engines with machine-learning and deep-learning-based detectors, highlighting the move from explicit signatures to data-driven models. We then surveyed ML-based NIDS across different observation granularities, from packet- and flow-level systems to time-series, graph-based, and mixed-granularity approaches, and reviewed a wide range of adaptation mechanisms, including online and continual learning, drift detection, active and semi-supervised labeling, transfer learning, and meta-learning. This analysis highlighted that most existing systems concentrate on updating model weights or decision scores, while leaving the underlying configuration of the detector and the evaluation workflow largely unchanged. At the same time, we analyzed the construction of widely used datasets such as IDS2017 and IDS2018 and identified structural limitations that can hide important weaknesses of trained detectors.

These observations led us to focus on two complementary aspects of robustness for ML-based NIDS: the robustness of their configuration under changing conditions, and the robustness of their final assessment before deployment. To address these overarching challenges,

Our problem is therefore divided into the following parts:

- Propose a solution to configure ML-based NIDS that can adapt when traffic characteristics shift, without rerunning costly hyperparameter optimization from scratch for each new deployment context. The configuration mechanism should operate under realistic constraints, in particular for unsupervised anomaly detectors that are trained only on benign traffic and for which attack labels are unavailable or rare.

- Propose a method to characterize the quality of benchmark test sets used for NIDS evaluation, to determine whether they provide sufficient coverage of benign and malicious behaviors and contain genuinely challenging cases.

- Propose a method to augment these test sets with additional, realistic network traffic so that pre-deployment evaluation is more likely to reveal weaknesses of trained detectors, while respecting protocol constraints at packet and flow level.

To address these points, we designed two main lines of contribution. The first is a meta-learning-based configuration framework that treats NIDS configuration as a supervised problem at the meta-level and learns to predict effective hyperparameters and decision thresholds from descriptors of the deployment context. The second is TATA, a framework for test-set assessment and targeted traffic augmentation. It provides a systematic methodology to evaluate the intrinsic difficulty of NIDS evaluation datasets using model-agnostic metrics. When a test set is found to be insufficiently challenging, TATA employs a reinforcement learning agent coupled with a controlled network testbed to generate supplementary traffic that reveals detector weaknesses.

We now summarize the main results obtained along these two axes, before discussing the limitations of our approach and possible extensions for future work.

Contributions

The work presented in this thesis has been disseminated through four main publications. The meta-learning configuration framework was initially introduced at the IEEE/IFIP Network Operations and Management Symposium (NOMS 2023). This work was subsequently extended and adapted to attack-free AD training settings at the IEEE Conference on Local Computer Networks (LCN 2024), and further validated by an empirical study on the configuration robustness of unsupervised systems at NOMS 2026. Finally, the test-set assessment and targeted augmentation framework (TATA) was published separately at the European Symposium on Research in Computer Security (ESORICS 2025). These publications correspond to the two core contributions summarized below.

Meta-Learning Framework for Automated Configuration

To overcome the inherent limitations of conventional hyperparameter optimization, we designed a meta-learning framework that treats NIDS configuration as a supervised problem at the meta-level. This system is designed to operate in dynamic environments where traffic patterns shift and attack behaviors evolve, requiring frequent reconfiguration. The framework relies on a two-phase architecture separating knowledge consolidation from operational inference.

The offline phase constructs a meta-model by using knowledge from historical pre-deployment episodes. It maps compact, detector-agnostic descriptors of network contexts (meta-features) to their optimal configurations found by an oracle. The online phase leverages this meta-model to infer a near-optimal configuration for a new network context in a single shot. By bypassing the iterative search process of traditional optimization, this approach drastically reduces the computational cost and time-to-deploy.

This contribution was developed in three successive steps:

- **NOMS 2023 (AnNet workshop) — supervised configuration transfer.** In [AFC23], we introduced the core idea of learning from prior tuning experiences to predict hyperparameters for a new context without re-running an expensive optimization loop from scratch. The focus is on accelerating configuration in standard (labeled) evaluation settings, and we demonstrated that the meta-learning approach is significantly faster than Bayesian optimization while keeping detection performance essentially unchanged in most cases.

- **LCN 2024 (Short paper) — attack-free configuration and joint thresholding.** In [AFC24], we extended the framework to address a major practical constraint in deployment: training anomaly-based intrusion detectors on attack-free (benign-only) datasets. In this setting, conventional automated optimization is difficult because it typically requires a validation set with attack samples. The paper therefore introduces an automated approach that infers hyperparameters and the decision threshold jointly using only benign traffic at configuration time, enabling deployment in realistic zero-day scenarios. This extension also substantially improves the time-to-configuration compared to standard optimization baselines.

- **NOMS 2026 — empirical study of configuration robustness for unsupervised NIDS.** In [AFC26], we complement the above methods with an empirical analysis of how strongly unsupervised NIDS performance depends on the chosen configuration when the operating context changes. The study quantifies performance degradation under configuration reuse across different daily contexts and benchmarks, and it motivates continuous automated reconfiguration as a practical requirement rather than an optional refinement.

Overall, the thesis presents a unified view of this configuration line of work, from the initial meta-learning strategy [AFC23] to its deployment-oriented extension for attack-free anomaly detection and threshold selection [AFC24], and to an empirical validation of why configuration robustness matters in practice [AFC26].

TATA: test-set assessment and targeted augmentation

The second contribution is TATA, a framework dedicated to the assessment and augmentation of NIDS test sets. Its goal is to determine whether a test set contains challenging enough cases and, when needed, to enrich it with additional realistic traffic.

TATA first learns a structured latent representation of the training data to approximate decision boundaries, then encodes the test set in this same space to extract model-agnostic quality metrics. These metrics quantify the difficulty of the test set and reveal whether it sufficiently challenges the detector. The framework then uses a reinforcement learning agent, connected to a controlled network testbed, to generate new protocol-faithful flows that target under-tested regions and expose weaknesses of the NIDS.

This contribution is presented in [AFCK25], published at **ESORICS 2025**. It formalizes the test-set quality problem for benchmark-based NIDS evaluation and proposes a practical augmentation strategy guided by explicit, model-agnostic metrics.

Experimental validation of both frameworks

A third contribution lies in the experimental evaluation of these two systems. We evaluated the performance of our contributions through extensive experiments on the IDS2017 and IDS2018 benchmarks.

For the meta-learning framework, we compared it against Bayesian Optimization. The results demonstrated that our method recovers configurations yielding detection performance close to the theoretical optimum while reducing reconfiguration time by up to two orders of magnitude (a speedup factor of roughly 160).

For TATA, we demonstrated that the proposed metrics accurately correlate with detection difficulty across multiple architectures (Random Forest, SVM, DNN). Furthermore, we showed that the targeted augmentation strategy is significantly more effective than random replay or

non-adaptive generation at uncovering model weaknesses. By generating valid traffic that sits on the decision boundary, TATA revealed limitations that static benchmarks failed to expose, providing a more honest and robust assessment of operational readiness.

Together, these experiments confirm the practical effectiveness of the two proposed contributions and highlight their complementary roles: one strengthens configuration robustness, the other strengthens evaluation robustness.

Limits and Perspectives

While this thesis advances the state of the art in NIDS adaptability and evaluation, several limitations remain that define the scope of our findings and point toward future research.

Practical and deployment limitations. Our meta-learning framework relies on a rich repository of historical tuning episodes to build its predictive model. Consequently, its effectiveness is constrained by the diversity of this meta-dataset. In a "cold-start" scenario (such as a deployment in a radically new environment with no prior compatible history) the meta-model may fail to infer an optimal configuration, requiring a fallback to traditional optimization methods until sufficient history is accumulated.

Similarly, TATA relies on a controlled network testbed and a reinforcement learning agent that drives traffic generators. In the current implementation, the focus is on one protocol. Extending this setup to multiple protocols, more complex applications, and higher traffic volumes would increase engineering costs and raise questions about scalability and maintenance. At this stage, TATA should be seen as a proof of concept, rather than as a turnkey tool ready for direct integration into production SOCs.

Extension to Heterogeneous Data Granularities. Our current frameworks are optimized for flow-based NIDS. A natural progression is to extend the meta-learning approach to handle packet-level sequences or graph-based representations directly. Developing meta-features that capture the topological structure of host interactions or the sequential dynamics of raw packets could allow the configuration framework to generalize across vastly different network architectures.

From Assessment to Hardening. Currently, TATA is used primarily for assessment (revealing weaknesses). A significant opportunity lies in integrating TATA directly into the training loop, using the generated "hard" examples not just for evaluation, but for improvement. Establishing a continuous feedback loop where the challenging samples generated by TATA are injected back into the training set would enable a form of **adversarial training**. This allows the NIDS to retrain and adjust its internal decision boundaries to specifically recognize and resist the vulnerabilities exposed during assessment.

Federated Configuration Learning and Privacy-Preserving Collaboration. To address the cold-start problem, future work could explore federated learning for the configuration meta-model. In this setting, several organizations would help train the model together without sharing their local network data. Each organization would keep its data locally and only share training updates. This could help a newly deployed NIDS benefit from experience collected in other environments, especially when local traffic or attack patterns are still limited. The main challenge is to share enough information to improve the model while keeping local network details private. Future work should therefore study how federated learning can support meta-learning for NIDS and when such collaboration really improves robustness across different environments.

Broader perspectives for ML-based NIDS

Beyond the concrete extensions listed above, several broader trends are likely to shape the next generation of machine-learning-based NIDS. These perspectives go beyond improving a single model or benchmark: they concern how NIDS will be evaluated, maintained, and trusted in continuously evolving infrastructures, and how defenders may need to respond to new capabilities such as Generative AI.

Toward continuous, closed-loop evaluation and adaptation

A key direction for the NIDS community is to move from "one-shot" evaluation to continuous evaluation, similar in spirit to continuous integration practices in software engineering. Instead of treating the benchmark test set as a final and sufficient proof of quality, future NIDS pipelines will increasingly need: (i) frequent re-assessment on fresh traffic samples, (ii) explicit checks that the evaluation data still covers challenging regions of the feature space, and (iii) mechanisms to generate or collect additional "hard" cases when coverage is insufficient. The perspective here is not only to improve metrics, but to make evaluation itself an active process that continuously questions whether the detector is being tested in meaningful conditions.

In that vision, frameworks such as TATA can be seen as building blocks for a broader "evaluation loop": the system does not merely report a score, but also diagnoses weaknesses, identifies under-tested regions, and proposes new test instances that are realistic and protocol-compliant. The larger challenge for the field is to generalize such approaches across many protocols, applications, and deployment contexts, while keeping the process affordable and maintainable at scale—especially since today's realistic testbeds can be expensive to engineer and operate. Over time, a mature NIDS evaluation loop may resemble a *security-oriented digital twin* of the monitored environment, in which detection models are continuously stress-tested against evolving benign behaviors and attacker strategies.

Data-centric NIDS: realism, representativeness, and labeling constraints

Another major perspective is the continued shift from model-centric progress (new architectures) to data-centric progress: how data is collected, processed, labeled, and updated becomes as important as the model choice itself. In NIDS, this point is amplified by two structural constraints:

- Label scarcity and ambiguity in operational networks: high-quality attack labels are rare, costly, and sometimes disputed.
- Distribution mismatch between public benchmarks and real deployments, where background traffic, software stacks, and usage patterns differ.

As a result, strong research directions include: (i) self-supervised or weakly supervised learning that relies less on labels, (ii) systematic sampling strategies to select what should be labeled by experts, and (iii) evaluation protocols that explicitly measure robustness under realistic shifts (new services, new clients, new encryption patterns, etc.). In this context, the ability to configure detectors from benign traffic only (an explicit objective in our configuration framework) should be viewed as part of a broader push toward deployability under imperfect supervision.

A related perspective is that "representativeness" should become a primary evaluation criterion: rather than assuming that a single benchmark split is representative, future work should develop methods to quantify *what is missing* (in benign diversity, boundary cases, rare behaviors, and attack variants), and to document such gaps transparently. This naturally connects to

community-wide efforts on reproducibility, dataset documentation, and benchmark governance, which are increasingly necessary as NIDS models become harder to compare fairly.

Generative AI as both an opportunity and a challenge

A central evolution that can no longer be ignored is the arrival of Generative AI (GenAI) (including large language models (LLMs) and other generative models) as both (i) a tool for defenders and (ii) a capability available to attackers.

For defenders, GenAI opens several opportunities that align with the objectives of this thesis:

- **Synthetic traffic generation at scale.** Realistic data is hard to share and expensive to obtain. GenAI can complement controlled testbeds by generating diverse traffic traces, new attack variants, and "edge" behaviors that may not appear in small benchmarks [JLGJ+24, CSBG24]. The key scientific challenge is not only to generate data, but to ensure validity and fidelity: generated samples must respect protocol semantics, timing constraints, and causal structure. Otherwise, synthetic data risks teaching detectors artifacts that do not exist in real networks. A promising direction is therefore *constraint-aware generation*, where generative models are guided by protocol grammars, simulators, or rule-based validators to keep outputs realistic.

- **Scenario-to-traffic compilation.** Another perspective is to use GenAI as an interface layer: analysts could describe a scenario in natural language ("simulate a misconfigured service exposed to scanning, followed by a slow exfiltration attempt"), and the system could translate it into concrete testbed actions and traffic generation scripts [WWQ+24, DLMV+24]. This would reduce the human cost of producing meaningful evaluation cases and could accelerate the development of richer, more diverse benchmarks. In such a workflow, GenAI does not replace testbeds but acts as a "planner" that proposes candidate scenarios that are then executed and validated in controlled environments.

- **Assistance for operational workflows.** NIDS alerts are often hard to interpret [AAM22, VKvE+23]. GenAI systems can help summarise alerts, correlate them with context (asset inventories, past incidents, known misconfigurations), and propose investigation steps [KRN+25, AAM25]. This can improve the practical value of detection in Security Operations Centers (SOCs), where analyst time is limited. However, this benefit depends on careful control of reliability: explanations must be traceable to evidence, and systems must be designed to avoid "plausible but wrong" conclusions [KRN+25].

At the same time, GenAI acts as a source of pressure: attackers can use it to automate reconnaissance, produce large volumes of adapted payloads, and quickly rephrase or mutate behaviors to evade static signatures. This reinforces the need for NIDS that are robust not only to drift, but also to adaptive adversaries. In other words, GenAI strengthens the case for the kind of continuous, stress-oriented evaluation proposed above.

Toward autonomous and trustworthy NIDS in real infrastructures

A longer-term perspective is the emergence of more autonomous NIDS, where configuration, thresholding, evaluation, and model updates become partially automated. In that vision, systems will likely operate as closed loops: they observe traffic, select or infer configurations, evaluate themselves on targeted tests, and trigger updates when performance risks increase. The contributions of this thesis can be interpreted as steps in that direction: a configuration mechanism

that reduces repeated costly tuning, and an evaluation mechanism that moves beyond static test sets toward targeted stress testing.

However, increasing autonomy raises a parallel requirement: trustworthiness. Operational NIDS decisions have real costs (alert fatigue, missed detections, incident response overhead). Future research should therefore emphasize:

- Calibration and uncertainty awareness (knowing when the model is unsure, and reacting accordingly).
- Explainability adapted to defenders (simple, evidence-based rationales, not only feature importance plots).
- Safety mechanisms (fallback modes when the system detects drift or low-confidence situations).
- Robustness to adversarial manipulation, including poisoning of feedback loops and exploitation of automated reconfiguration.

In addition, trustworthiness is also a governance problem: organizations need clear processes for validating model updates, tracking data provenance, and documenting why a given configuration or model version was selected.

List of Figures

1 Thesis organization: research questions, contributions, and chapters. 7

1.1 Example of GIPS signature-group generation on a packet stream. (a) Data stream that contains repeated exploit payloads. (b) Manual rule composition, where an analyst inspects payloads and chooses a signature string. (c) Automatic single-signature generation, which tends to select frequent but non-discriminative substrings such as "http". (d) GIPS learns a signature group of co-occurring substrings (for example, "root" and "admin") that more precisely characterizes the exploit. Adapted from [SY23]. 14

1.2 Example of a deep-learning-based NIDS pipeline (KitNET in Kitsune): an input feature vector is mapped to an ensemble of autoencoders, and reconstruction errors are aggregated into a final anomaly score. Adapted from [MDES18]. 17

1.3 Overview of the OWAD workflow, illustrating the four stages of drift detection and model refinement: (1) Calibration, (2) Shift detection, (3) Explanation, and (4) Adaptation. Adapted from [HWL23] 24

1.4 Overview of the ENIDrift workflow, illustrating its three primary stages: (1) iP2V embedding, (2) Ensemble detection, and (3) Learning new distribution. Adapted from [Wan22]. 26

1.5 Overview of ReCDA. **Stage 1 — Representation Enhancement:** apply *Drift-aware Perturbation* (feature-wise mixing of a historical flow D^h with a recent flow D^r) and *Representation Alignment* (contrastive loss) to learn features that are drift-aware yet invariant. **Stage 2 — Classifier Tuning:** use *Instructive Sampling* to select a small labeled subset from D^h, and update the encoder and classifier with cross-entropy plus a *Representation-Consistency Regularizer*; no new annotations are required. Adapted from [XLW24]. 27

1.6 **Active–SVDD overview.** Active querying scores unlabeled points with a convex combination of a margin score (distance to the SVDD boundary) and a k-NN neighborhood score that favors points surrounded by similar unlabeled samples but not close to the labeled benign region; the final query score is $\tau\lambda_1 + (1-\tau)\lambda_2$. Adapted from Görnitz [GKRB09]. 29

1.7 Overview of the k-NN distance diagnostic pipeline for detecting sampling bias between labeled training data (D_T) and unlabeled deployment data (D_U). Adapted from [TDK+24]. 31

1.8 Architecture of the adversarial domain adaptation framework for NIDS, illustrating the joint training of the shared generator (G), domain discriminator (D), and intrusion classifier. Adapted from [SVB20]. 32

1.9 Overview of the FC-Net few-shot detection pipeline, illustrating the pairwise comparison process using F-Net for feature extraction and C-Net for similarity scoring. Adapted from [XSD20]. 34

1.10 Overview of the PTN-IDS prototype-based few-shot detection pipeline, illustrating the meta-training process and prototype computation for deployment. Adapted from [NMW24]. 34

1.11 Overview of the non-parametric dynamic thresholding and pruning process. The plot illustrates how candidate anomalous segments of the smoothed prediction error are evaluated and filtered based on their relative peak magnitudes. Adapted from [HCL+18]. 37

2.1 Illustration of the NIDS workflow from configuration and training through validation and final evaluation, culminating in deployment. 43
2.2 Two-phase meta-learning framework for NIDS autoconfiguration. The offline phase builds a meta-model from past tuning outcomes; the online phase uses a small sample from a new context to infer a configuration in one shot. 49
2.3 Meta-model construction in the offline phase. Each past context is summarized into meta-features, paired with the optimal configuration found offline, and used to train a supervised predictor. 51

3.1 Daily traffic composition in IDS2017 (D0–D3) and IDS2018 (D4–D13). Rows denote days and columns denote traffic classes (benign and 31 attack families), ordered by taxonomy ID. Cell color shows the class share per day (% of flows) using a log-scaled color mapping; "Att." denotes unsuccessful attack attempts. 59
3.2 Flow Count Across Days 60
3.3 Nested cross-validation workflow: the inner loop selects the configuration on training data, while the outer loop evaluates generalization on held-out test folds. Adapted from [LSG+23]. 61
3.4 Configuration generation, learning and prediction time of the NIDS 66
3.5 Relative performance of MtL calculated as a ratio between (MtL-HPO) and HPO (an infinite value due to division by zero is marked as 100% on day 13) 66
3.6 Detection performance according to the meta-dataset size (#days) 67
3.7 Comparison of different configurations: default, single overall, under-sampling, MtL, optimal 69
3.8 Daily optimal configurations transferability 70
3.9 Transferability of optimal configurations across attacks. 71
3.10 F1 score with a daily configuration 72
3.11 Precision with a daily configuration 73
3.12 Configuration generation time. 74
3.13 Performance gain of semi-supervised `MtL` computed as the difference between the performance of `MtL` and BO. 74

4.1 Overview of the Rosetta workflow for robust encrypted traffic classification, illustrating the three phases of Traffic Invariant Extractor (TIE) training, classifier training, and deployment. Adapted from [XZN23]. 85
4.2 Overview of the NetShare synthetic trace generation workflow, illustrating the trace splitting, scalable model training, and differential privacy mechanisms. Adapted from [YLJ+22]. 86
4.3 The TrafficGPT architecture for traffic analysis and generation, detailing the tokenization process, Transformer pre-training, and downstream adaptation tasks. Adapted from [QML24]. 87
4.4 Geometric illustration of Distance-based Surprise Adequacy (DSA) in the neural network activation space. The diagram visualizes how novelty is quantified based on the relative distances between a test input and the learned training class clusters. Adapted from [KYC+19]. 89

4.5 Decision boundaries of the original model and two mutant models for a binary classifier. Points near the original boundary (shown in green) are more likely to cause disagreement with mutants, thus "killing" them; mutation testing therefore favors tests in this sensitive region. Adapted from Ma *et al.*, DeepMutation (Fig. 4) [MJZ+18]. 89
4.6 Overview of the NIDSGAN adversarial generation architecture, illustrating the generation of perturbation vectors and the dual-evaluation constraints for evasion and realism. Adapted from [ZSM+22]. 90
4.7 The AdvTG three-phase workflow for adversarial generation. (1) Detection Model Training: Multiple NIDS models are trained to serve as black-box reward functions. (2) Domain-Specific LLM Fine-Tuning: A general-purpose LLM is fine-tuned on HTTP traffic to learn the protocol's structure. (3) RL-based Adversarial Generation: The domain-specific LLM from Phase 2 is optimized via reinforcement learning, using feedback from the Phase 1 reward models to generate evasive traffic. Adapted from Sun *et al.* (WWW 2025), Fig. 3 [SYL+25]. 91
4.8 The ProGen adversarial generation workflow, illustrating the two-phase process of training a black-box generator and launching evasive malicious traffic traces. Adapted from [WYFPW24]. 92

5.1 TATA, a method to assess and augment the quality of a test set 99
5.2 Training vs. Test Distances for $C_{1,\text{Benign}}$ 101
5.3 Optimal scarcity scenario 102
5.4 Non-optimal scarcity scenario 102
5.5 Visual intuition of the Vendi Score (VS) on synthetic data [FD23]. The axes represent individual data points (shapes and colors), and the heatmaps display their pairwise similarity. **(a)** Vendi Score interprets diversity as the effective number of unique elements: the score scales linearly ($2.00 \rightarrow 4.00$) with the number of distinct clusters (yellow blocks). **(c)** Vendi Score is robust to feature correlations: It increases ($3.00 \rightarrow 4.66$) as attributes (e.g., shape and color) become less correlated, rewarding complex variation over simple redundancy. 103

6.1 Box plots of the diversity metric for sub-test sets of size $k = 1, \ldots, 9$ 111
6.2 Effect of Proximity on NIDS Performance 111
6.3 Effect of Scarcity on NIDS Performance 112
6.4 Impact of TATA and Baselines on Test-Set Quality Metrics and Detection Performance 114
6.5 Impact of Partial Metric Combinations on Test-Set Quality and NIDS Performance 115
6.6 Test Sets Assessment of NIDS Datasets 118

List of Tables

1.1 Categorization of Drift-Detection Mechanisms 23
1.2 Synthesis of adaptation families for ML/DL-based network intrusion-detection systems. 38

2.1 Meta-features used to build the context descriptor $\phi(\mathcal{D})$ (fast, model-agnostic). Items marked † require labels (supervised-only). 50

3.1 Features considered in each dataset 58
3.2 Comparison performance results (best value in green, worst value in red, intermediate value in black) 64
3.3 List of the datasets 65

4.1 Comparison of network-traffic augmentation strategies for NIDS evaluation. . . . 94

5.1 Methods & limitations for test set assessment (RQ1) & augmentation (RQ2) 96

6.1 Clustering quality of the contrastive vs. vanilla auto-encoder (mean ± standard deviation over ten random splits). 110

A.1 Summary of the 37 peer-reviewed intrusion detection systems analyzed in this work. 138
A.2 Summary of the 37 peer-reviewed intrusion detection systems analyzed in this work. 139
A.3 Summary of the 37 peer-reviewed intrusion detection systems analyzed in this work. 140
A.4 Summary of the 37 peer-reviewed intrusion detection systems analyzed in this work. 141
A.5 Summary of the 37 peer-reviewed intrusion detection systems analyzed in this work. 142
A.6 List of acronyms used in this thesis. 143

Annexes

Appendix A

Première Annexe

Table A.1: Summary of the 37 peer-reviewed intrusion detection systems analyzed in this work.

Reference	**Description**	**Configuration**
[WLZH23]	An explanation framework for DL-NIDS that captures temporal context and generates active defense rules.	Grid Search
[PSPW23]	A transformer-based model that detects SSH impostors by analyzing keystroke dynamics from client packet timing and lengths in encrypted traffic.	Manual
[SY23]	A minHash-based streaming method that identifies big groups of similar data and generates signature-groups to detect zero-day attacks.	Manual
[MSM+24]	An in-network serverless data layer that coordinates flow measurement and edge aggregation to supply high-fidelity features for distributed ML defenses.	Manual
[RSW+23]	An unsupervised GRU autoencoder that captures temporal context across device states and actions, flagging abnormal actions via a dynamic threshold.	Manual
[UHH+24]	An adaptive misuse detection system that learns from SIEM rules versus historical benign events to detect rule evasions and attribute the likely evaded rules.	Grid Search
[CS17]	Identifies attacker ECUs on CAN by fingerprinting transmitter voltage signals and updating these fingerprints online to stay robust to drift.	Manual
[CS16]	A CAN IDS that fingerprints ECUs by estimating clock skew from periodic message timing, enabling attacker identification and detecting fabrication, suspension, and masquerade attacks.	Manual
[YdHE16]	A semantics-aware ICS anomaly detector that turns protocol fields into interpretable features and flags rare bins in per-entity probability models.	Manual
[MDES18]	An online, unsupervised NIDS that clusters correlated traffic features to build an ensemble of small autoencoders and flags anomalies by reconstruction error.	Manual
[SBC+19]	An air-gapped radio frequency side-channel IDS for substations that infers equipment state from low-frequency magnetic emissions and cross-checks SCADA data.	Manual

Table A.2: Summary of the 37 peer-reviewed intrusion detection systems analyzed in this work.

Reference	**Description**	**Configuration**
[YHY+20]	A non-invasive ICS IDS that mines control invariants from SCADA logs to build a control graph and detects and localizes PLC intrusions by monitoring edge-weight deviations.	Manual
[HPB+20]	A runtime APT detector that encodes whole-system provenance and uses evolutionary clustering to flag anomalous states and transitions.	Automatic
[NP15]	A Suricata-based NIDS redesigned for many-core processors and programmable NICs with per-core engines and dynamic host offload to reach 79 Gbps throughput.	Manual
[YXX+23]	Detects APTs by embedding provenance graph snapshots with graph2vec and modeling their temporal dynamics with TextRCNN, then ranking suspicious subgraphs.	Grid Search
[WZM+21]	A symbolic-execution guided NIDS that extracts TCP implementation models and forks connection state on ambiguous packets to defeat evasion.	Manual
[WTvS+22]	Four parameter-free ICS detectors (min–max, gradient, steady-time, histogram) that raise interpretable alerts on sensor deviations.	parameter-free

Table A.3: Summary of the 37 peer-reviewed intrusion detection systems analyzed in this work.

Reference	**Description**	**Configuration**
[WCH+22]	A federated contrastive-learning IDS for IoT gateways that learns a compact benign representation and flags anomalies by cosine similarity to a benign template with a percentile threshold; robust under non-IID clients.	Manual
[SKL+23]	A real-time CAN attacker identifier that injects bit errors to force the source ECU into error passive, then detects priority reduction on its retransmissions to pinpoint the node, with checks for controller reset, one-shot mode, and fast transmission.	Manual
[SBF+23]	A zero-byte CAN defense that authenticates transmitters via secret inter-frame spacing sequences; a trusted officer validates the spacing and interrupts any violating message.	Manual
[KSJ+23]	A real-time provenance IDS that streams a message-passing GNN to sketch evolving graphs and uses an adversarially trained transformer to flag anomalous embedding sequences at billion-edge scale.	Manual
[SLH23]	A lightweight CAN IDS that selects per-signal time-series features and applies PCA to learn cross-signal dependencies, detecting fabrication, suspension, and masquerade attacks.	Manual

Table A.4: Summary of the 37 peer-reviewed intrusion detection systems analyzed in this work.

Reference	Description	Configuration
[KJJ+23]	A robust in-network counting sketch for programmable switches that uses cross-layer minimum-update on a pyramid of counters to keep accurate per-flow measurements across changing traffic distributions without reconfiguration.	Manual
[JBEM+25]	A node-level provenance IDS that encodes temporal edges with a GNN, auto-thresholds and clusters outliers, then reconstructs concise attack paths via causality tracing for high-quality attribution.	Automatic
[WWL+25]	An ICS IIDS that mines state-space models from historical process data and raises alerts when the running sum of prediction errors grows unusually large, producing interpretable alarms.	Manual
[NG24]	A fully online self-supervised IDS framework for IoT that self-labels traffic, estimates model trust, and schedules weight updates without offline data.	Manual
[GWB24]	A process-level provenance IDS that precomputes each node's root causes and injects them into the GNN embedding, improving detection and yielding root-cause-linked attributions.	Automatic
[AOA+24]	Derives and enforces state-aware invariants for ICS by combining PLC program analysis with state-partitioned trace mining, then checks them in a PLC runtime monitor to raise tighter, more sensitive alerts.	Automatic
[DLXQ24]	A voltage-based CAN IDS that builds per-ID fingerprints with a one-class GMM, removing the need for ECU↔ID mappings and detecting both periodic and aperiodic malicious frames.	Manual

Table A.5: Summary of the 37 peer-reviewed intrusion detection systems analyzed in this work.

Reference	**Description**	**Configuration**
[DBMS+24]	An early-flow NIDS that uses RNNs with attention to classify attacks from packet headers before flow termination, reducing packets and time to detection.	Manual
[IPS+23]	A SCADA-plus-physical correlation IDS for ICS that derives process impact from OPC data and flags phase-specific API violations and disruptive control signals.	Manual
[WYW24]	A knowledge-guided graph NIDS that models each flow as a heterogeneous temporal graph and uses an HTGNN (a GNN that handles different node/edge types and their time order) with an adjustable early-detection decoder for better transferability.	Manual
[WSL+24]	A rule-based provenance IDS that auto-tunes node, edge, and alert-threshold parameters using a differentiable tag-propagation model optimized by gradient descent.	Automatic
[CZA+16]	A specification-mining NIDS for BACnet that auto-extracts device capabilities from documentation (PICS/EDE) and enforces type, value, and method rules on live traffic.	Automatic
[GPM+21]	A reactive CAN-bus defense that uses relay-controlled topology reconfiguration and a Bus Guardian to isolate intruders, filter, and redirect traffic under DoS.	Manual
[CLL+24]	A whole-system provenance IDS that learns temporal and structural patterns with a GNN encoder-decoder and reconstructs compact attack summary graphs at run time.	Manual
[uRZRX24]	A provenance IDS that combines Word2Vec semantic encoding, a GNN for contextual structure, and an embedding-recycling database to enable real-time node-level detection.	Manual

Table A.6: List of acronyms used in this thesis.

Acronym	Meaning
AD	Anomaly Detection
AD NIDS	Anomaly Detection Network Intrusion Detection System
AI	Artificial Intelligence
AL	Active Learning
ANS	Agence du Numérique en Santé (French Digital Health Agency)
ARI	Adjusted Rand Index
AUC	Area Under the Curve
AUCPR / AUPRC	Area Under the Precision–Recall Curve
AUROC	Area Under the ROC Curve
BO	Bayesian Optimization
C-Net	Comparison network (component of FC-Net)
CCS	ACM Conference on Computer and Communications Security
CIC-IDS2017	CIC-IDS2017 benchmark dataset
CNN	Convolutional Neural Network
CSE-CIC-IDS2018	CSE-CIC-IDS2018 benchmark dataset
CV	Cross-validation
DANN	Domain-Adversarial Neural Network
DDoS	Distributed Denial of Service
DI-NIDS	Domain-invariant network intrusion detection system
DINUM	Direction interministérielle du numérique (French interministerial digital directorate)
DL	Deep Learning
DNN	Deep Neural Network
DoS	Denial of Service
DP-SGD	Differentially Private Stochastic Gradient Descent
DRL	Deep Reinforcement Learning
DSA	Distance-based Surprise Adequacy
DT	Decision Tree
ESORICS	European Symposium on Research in Computer Security
EWMA	Exponentially Weighted Moving Average
F-Net	Feature extractor (component of FC-Net)
FN	False Negative(s)
FP	False Positive(s)
FPR	False Positive Rate
GAN	Generative Adversarial Network
GenAI	Generative AI
GMM	Gaussian Mixture Model
GNN	Graph Neural Network
GPM	Gradient Projection Memory
HPO	Hyperparameter Optimization
IAT	Inter-arrival time
ICS	Industrial Control Systems
IDS2017	Shorthand for the CIC-IDS2017 dataset

Continued on next page

Acronym	Meaning
IDS2018	Shorthand for the CSE-CIC-IDS2018 dataset
IF	Isolation Forest
IPFIX	IP Flow Information Export
k-NN	k-nearest neighbors
LCN	IEEE Conference on Local Computer Networks
LLM	Large Language Model
LOF	Local Outlier Factor
LSA	Likelihood-based Surprise Adequacy
MCC	Matthews Correlation Coefficient
ML	Machine Learning
MLP	Multi-Layer Perceptron
MS	Mutation Score
MtL	Meta-learning
NC	Neuron Coverage (metric); also used as Nearest-Centroid (oracle) depending on context
NDSS	Network and Distributed System Security Symposium
NIDS	Network Intrusion Detection System
NIST	National Institute of Standards and Technology
NMI	Normalized Mutual Information
NOMS	IEEE/IFIP Network Operations and Management Symposium
NN	Neural Network
OCSVM	One-Class Support Vector Machine
OSVM	One-class SVM
PCA	Principal Component Analysis
PCAP	Packet capture file (raw packet trace)
POMDP	Partially Observable Markov Decision Process
PR	Precision–Recall
RF	Random Forest
RFr	Random Forest regressor
R2L	Remote-to-Local
RL	Reinforcement Learning
RNN	Recurrent Neural Network
ROC	Receiver Operating Characteristic
ROC-AUC	Area Under the ROC Curve
RU-SMOTE	Random undersampling with SMOTE
SA	Surprise Adequacy
S&P	IEEE Symposium on Security and Privacy
SDN	Software-Defined Network(s)
SIEM	Security Information and Event Management
SMOTE	Synthetic Minority Over-sampling Technique
SOC	Security Operations Center
SPAN	Switch-Port Analyzer
SSL	Semi-Supervised Learning
SVM	Support Vector Machine
SVDD	Support Vector Data Description

Continued on next page

Acronym	Meaning
TATA	Test-set Assessment and Targeted Augmentation
TIE	Traffic Invariant Extractor
TLS/SSL	Transport Layer Security / Secure Sockets Layer
TN	True Negative(s)
TP	True Positive(s)
TPR	True Positive Rate
U2R	User-to-Root
VS	Vendi Score

Bibliography

[AAM22] Bushra A. Alahmadi, Louise Axon, and Ivan Martinovic. 99% false positives: A qualitative study of SOC analysts' perspectives on security alarms. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 2783–2800, Boston, MA, USA, 2022. USENIX Association.

[AAM25] Abdullah Aldaihan, Fahad Alotaibi, and Sergio Maffeis. Clouseau: A hierarchical multi-agent approach for autonomous attack investigation. In *2025 IEEE Annual Computer Security Applications Conference (ACSAC)*, pages 516–532. IEEE, 2025.

[ACT13] Paul Aitken, Benoît Claise, and Brian Trammell. Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information. RFC 7011, September 2013.

[AFC23] Omar Anser, Jérôme François, and Isabelle Chrisment. Auto-tuning of hyperparameters for detecting network intrusions via meta-learning. In *NOMS 2023 - IEEE/IFIP Network Operations and Management Symposium (NOMS), AnNet Workshop*, pages 1–6, 2023.

[AFC24] Omar Anser, Jérôme François, and Isabelle Chrisment. Automated machine learning configuration to learn intrusion detectors on attack-free datasets. In *2024 IEEE 49th Conference on Local Computer Networks (LCN)*, pages 1–7, 2024.

[AFC26] Omar Anser, Jérôme François, and Isabelle Chrisment. An empirical study on configuration robustness of unsupervised ids. In *2026 IEEE/IFIP Network Operations and Management Symposium (NOMS)*, 2026. HAL: `https://hal.science/view/index/docid/5656209`.

[AFCK25] Omar Anser, Jérôme François, Isabelle Chrisment, and Daishi Kondo. Tata: Benchmark nids test sets assessment and targeted augmentation. In *ESORICS 2025 - 30th European Symposium on Research in Computer Security*, 2025. HAL: hal-05148383.

[ALS23] Giovanni Apruzzese, Pavel Laskov, and Johannes Schneider. Sok: Pragmatic assessment of machine learning for network intrusion detection. In *2023 IEEE 8th European Symposium on Security and Privacy (EuroS&P)*, pages 592–614. IEEE, 2023.

[AM16] Blake Anderson and David McGrew. Identifying encrypted malware traffic with contextual flow data. In *Proceedings of the 2016 ACM Workshop on Artificial*

Intelligence and Security (AISec '16), pages 35–46, Vienna, Austria, 2016. Association for Computing Machinery.

[AM24] Fahad Alotaibi and Sergio Maffeis. Mateen: Adaptive ensemble learning for network anomaly detection. In *Proceedings of the 27th International Symposium on Research in Attacks, Intrusions and Defenses*, RAID '24, page 215–234, New York, NY, USA, 2024. Association for Computing Machinery.

[AMB21] Elie Alhajjar, Paul Maxwell, and Nathaniel D. Bastian. Adversarial machine learning in network intrusion detection systems. *Expert Systems with Applications*, 186:115782, 2021.

[AOA+24] Syed Ghazanfar Abbas, Muslum Ozgur Ozmen, Abdulellah Alsaheel, Arslan Khan, Z. Berkay Celik, and Dongyan Xu. SAIN: Improving ICS attack detection sensitivity via State-Aware invariants. In *33rd USENIX Security Symposium (USENIX Security 24)*, Philadelphia, PA, 2024. USENIX Association.

[APC22] Giovanni Apruzzese, Luca Pajola, and Mauro Conti. The cross-evaluation of machine learning-based network intrusion detection systems. *IEEE Transactions on Network and Service Management*, 19(4):5152–5169, 2022.

[APM17] Chuadhry Mujeeb Ahmed, Venkata Reddy Palleti, and Aditya P. Mathur. WADI: A water distribution testbed for research in the design of secure cyber-physical systems. In *Proceedings of the 3rd International Workshop on Cyber-Physical Systems for Smart Water Networks (CySWater '17)*. ACM, 2017.

[AQP+22] Daniel Arp, Erwin Quiring, Feargus Pendlebury, Alexander Warnecke, Fabio Pierazzi, Christian Wressnegger, Lorenzo Cavallaro, and Konrad Rieck. Dos and don'ts of machine learning in computer security. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 3971–3988. USENIX Association, 2022.

[ASDS21] Maged AbdelAty, Sandra Scott-Hayward, Roberto Doriguzzi-Corin, and Domenico Siracusa. GADoT: Gan-based adversarial training for robust DDoS attack detection. In *Proceedings of the 2021 IEEE Conference on Communications and Network Security (CNS)*, pages 119–127. IEEE, 2021.

[Axe00] Stefan Axelsson. Intrusion detection systems: A survey and taxonomy. Technical Report 99-15, Department of Computer Engineering, Chalmers University of Technology, Göteborg, Sweden, March 2000.

[AYL+11] Fatemeh Amiri, Mohammad Mahdi Rezaei Yousefi, Caro Lucas, Azadeh Shakery, and Nasser Yazdani. Mutual information-based feature selection for intrusion detection systems. *Journal of Network and Computer Applications*, 34(4):1184–1199, 2011.

[Bar24] Andreas Barttels. Detection of command and control traffic using suricata, 2024. States that the free ET Open ruleset contains ~50 000 Snort/Suricata signatures, highlighting the scale of real-world rule corpora.

[BB12] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13:281–305, 2012.

[BCR+18] Atilim Güneş Baydin, Robert Cornish, David Martinez Rubio, Mark Schmidt, and Frank Wood. Online learning rate adaptation with hypergradient descent. In *Proceedings of the 6th International Conference on Learning Representations (ICLR)*, 2018.

[BG16] Anna L. Buczak and Erhan Guven. A survey of data mining and machine learning methods for cyber security intrusion detection. *IEEE Communications Surveys & Tutorials*, 18(2):1153–1176, 2016.

[BHA09] Stéphane Bernard, Laurent Heutte, and Sébastien Adam. Influence of hyperparameters on random forest accuracy. In Jon Atli Benediktsson, Josef Kittler, and Fabio Roli, editors, *Multiple Classifier Systems*, volume 5519 of *Lecture Notes in Computer Science*, pages 171–180. Springer, 2009.

[BJR17] Robert A. Bridges, Jessie D. Jamieson, and Joel W. Reed. Setting the threshold for high-throughput detectors: A mathematical approach for ensembles of dynamic, heterogeneous, probabilistic anomaly detectors. In *2017 IEEE International Conference on Big Data (BigData)*, pages 1071–1078, 2017.

[BKNS00] Markus M. Breunig, Hans-Peter Kriegel, Raymond T. Ng, and Jörg Sander. Lof: Identifying density-based local outliers. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, pages 93–104. ACM, 2000.

[BL21] Sikha Bagui and Kunqi Li. Resampling imbalanced data for network intrusion detection datasets. *Journal of Big Data*, 8(6), 2021.

[BM01] Rebecca G. Bace and Peter Mell. Intrusion detection systems. Special publication 800-31, NIST, 2001.

[BM16] M. C. Belavagi and P. Muniyal. Performance evaluation of supervised machine learning algorithms for intrusion detection. *Procedia Computer Science*, 89:117–123, 2016.

[BMZ14] Sandeep Bhatt, Pratyusa K Manadhata, and Loai Zomlot. The operational role of security information and event management systems. *IEEE Security & Privacy*, 12(5):35–41, 2014.

[Bre01a] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[Bre01b] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[BS11] Robin Berthier and William H. Sanders. Specification-based intrusion detection for advanced metering infrastructures. In *IEEE Pacific Rim International Symposium on Dependable Computing (PRDC)*, pages 184–193, 2011.

[BTO06] Luc Bernaille, Rémi Teixeira, and Kimberly Ostroumov. Early application identification. In *Proc. ACM IMC*, 2006.

[BZEH06] Damiano Bolzoni, Emmanuele Zambon, Sandro Etalle, and Pieter H. Hartel. Poseidon: A 2-tier anomaly-based intrusion detection system. In *Proceedings of the 4th IEEE International Workshop on Information Assurance (IWIA 2006)*, pages 144–156. IEEE Computer Society, 2006.

[Can17] Canadian Institute for Cybersecurity. Cicflowmeter (formerly iscxflowmeter). `https://www.unb.ca/cic/research/applications.html`, 2017. Generates bidirectional flows and computes >80 features separately for forward/backward directions.

[CASA25] Mohamed Hashim Changrampadi, Magnus Almgren, Pablo Picazo- Sanchez, and Ahmed Ali-Eldin. Snort Meets Transformers: Accelerating Transformer-Based Network Traffic Classification for Real-Time Performance. In *Proceedings of the 18th European Workshop on Systems Security (EuroSec '25)*, pages 33–39, Rotterdam, Netherlands, 2025. Association for Computing Machinery.

[CBHK02] Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer. SMOTE: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, 16:321–357, 2002.

[CBK09] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM Computing Surveys*, 41(3):15:1–15:58, 2009.

[CDGS07] Manuel Crotti, Maurizio Dusi, Francesco Gringoli, and Luca Salgarelli. Traffic classification through simple statistical fingerprinting. *SIGCOMM Comput. Commun. Rev.*, 37(1):5–16, January 2007.

[CHK07] Yu Chen, Kai Hwang, and Wei-Shinn Ku. Distributed Change-Point detection of DDoS attacks: Experimental results on DETER testbed. In *DETER Community Workshop on Cyber Security Experimentation and Test 2007 (DETER 2007)*, Boston, MA, August 2007. USENIX Association.

[CHMF16] Carlos Garcia Cordero, Sascha Hauke, Max Mühlhäuser, and Mathias Fischer. Analyzing flow–based anomaly intrusion detection using replicator neural networks. In *14th Annual Conference on Privacy, Security and Trust (PST)*, pages 317–324. IEEE, 2016.

[CJ22] Dylan Chou and Meng Jiang. A survey on data-driven network intrusion detection. *ACM Computing Surveys*, 54(9):182:1–182:36, 2022.

[CJL+25] Andrew Chu, Xi Jiang, Shinan Liu, Arjun Bhagoji, Francesco Bronzino, Paul Schmitt, and Nick Feamster. Netssm: Multi-flow and state-aware network trace generation using state-space models. *arXiv preprint arXiv:2503.22663*, 2025.

[CKD19] Younghun Chae, Natallia Katenka, and Lisa DiPippo. An adaptive threshold method for anomaly-based intrusion detection systems. In *2019 IEEE 18th International Symposium on Network Computing and Applications (NCA)*, pages 505–510, 2019.

[CLL+24] Zijun Cheng, Qiujian Lv, Jinyuan Liang, Yan Wang, Degang Sun, Thomas Pasquier, and Xueyuan Han. Kairos: Practical intrusion detection and investigation using whole-system provenance. In *2024 IEEE Symposium on Security and Privacy (SP)*, pages 3533–3551, 2024.

[CMB24] Marco Cantone, Claudio Marrocco, and Alessandro Bria. On the cross-dataset generalization of machine learning for network intrusion detection. *arXiv:2402.10974*, 2024.

[Cor21] David Cortes. Revisiting randomized choices in isolation forests. *arXiv preprint arXiv:2110.13402*, 2021.

[Cor23] David Cortes. isotree: Isolation-based outlier detection. `https://isotree.readthedocs.io/en/latest/`, 2023.

[CPV23] Marta Catillo, Antonio Pecchia, and Umberto Villano. Machine learning on public intrusion datasets: Academic hype or concrete advances in nids? In *Proceedings of the 53rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks - Supplemental Volume (DSN-S)*, 2023.

[CS16] Kyong-Tak Cho and Kang G. Shin. Fingerprinting electronic control units for vehicle intrusion detection. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 911–927, Austin, TX, 2016. USENIX Association.

[CS17] Kyong-Tak Cho and Kang G. Shin. Viden: Attacker identification on in-vehicle networks. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS '17)*, pages 1109–1123, New York, NY, USA, 2017. Association for Computing Machinery.

[CSBG24] Joscha Cüppers, Adrien Schoen, Gregory Blanc, and Pierre-François Gimenez. Flowchronicle: Synthetic network flow generation through pattern set mining. In *Proceedings of the 20th International Conference on emerging Networking EXperiments and Technologies*, pages 1–20, 2024.

[CTBC22] Yousra Chabchoub, Maurras U. Togbé, Aliou Boly, and Raja Chiky. An in-depth study and improvement of isolation forest. *IEEE Access*, 10:10219–10237, 2022.

[CV95] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

[CVW+20] Carlos Garcia Cordero, Emmanouil Vasilomanolakis, Aidmar Wainakh, Max Mühlhäuser, and Simin Nadjm-Tehrani. On generating network traffic datasets with synthetic attacks for intrusion detection. *ACM Transactions on Privacy and Security*, 0(0):39:1–39:39, September 2020.

[CZA+16] Marco Caselli, Emmanuele Zambon, Johanna Amann, Robin Sommer, and Frank Kargl. Specification mining for intrusion detection in networked control systems. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 791–806, Austin, TX, August 2016. USENIX Association.

[DAM23] Ayesha Siddiqua Dina, B. Siddique A. and D. Manivannan. Fs3: Few-shot and self-supervised framework for efficient intrusion detection in internet of things networks. In *Proceedings of the 39th Annual Computer Security Applications Conference (ACSAC '23)*, Austin, TX, USA, 2023. Association for Computing Machinery.

[DBMS+24] Taki Eddine Toufik Djaidja, Bouziane Brik, Sidi Mohammed Senouci, Abdelwahab Boualouache, and Yacine Ghamri-Doudane. Early network intrusion detection enabled by attention mechanisms and rnns. *IEEE Transactions on Information Forensics and Security*, 19, 2024.

[DCMSH+20] R. Doriguzzi-Corin, S. Millar, S. Scott-Hayward, J. Martínez-del Rincón, and D. Siracusa. Lucid: A practical, lightweight deep learning solution for ddos attack detection. *IEEE Transactions on Network and Service Management*, 2020.

[DLMV+24] Gelei Deng, Yi Liu, Víctor Mayoral-Vilches, Peng Liu, Yuekang Li, Yuan Xu, Tianwei Zhang, Yang Liu, Martin Pinzger, and Stefan Rass. PentestGPT: Evaluating and harnessing large language models for automated penetration testing. In *33rd USENIX Security Symposium (USENIX Security 24)*, pages 847–864, Philadelphia, PA, USA, 2024. USENIX Association.

[DLXQ24] Zhouyan Deng, Jiajia Liu, Yijie Xun, and Junman Qin. Identifierids: A practical voltage-based intrusion detection system for real in-vehicle networks. *IEEE Transactions on Information Forensics and Security*, 19, 2024.

[E+21] G. Engelen et al. Troubleshooting an intrusion detection dataset: the cicids2017 case study. In *WTMC 2021*, 2021.

[ELKJ20] Mahmoud Said Elsayed, Nhien-An Le-Khac, and Anca D. Jurcut. Insdn: A novel sdn intrusion dataset. *IEEE Access*, 8:165263–165284, 2020.

[ELL+22] Gints Engelen, Lisa Liu, Timothy Lynar, Daryl Essam, and Wouter Joosen. Error prevalence in NIDS datasets: A case study on CIC-IDS-2017 and CSE-CIC-IDS-2018. In *Proc. IEEE Conf. on Communications and Network Security (CNS)*, pages 254–263, 2022.

[FA24] Robert Flood and David Aspinall. Measuring the complexity of benchmark nids datasets via spectral analysis. In *2024 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, 2024.

[FD23] Dan Friedman and Adji Bousso Dieng. The vendi score: A diversity evaluation metric for machine learning, 2023.

[FEAD24a] Robert Flood, Gints Engelen, David Aspinall, and Lieven Desmet. Bad design smells in benchmark nids datasets. In *Proceedings of the 2024 IEEE 9th European Symposium on Security and Privacy (EuroS&P)*, 2024.

[FEAD24b] Robert Flood, Gints Engelen, David Aspinall, and Lieven Desmet. Bad design smells in benchmark NIDS datasets. In *2024 IEEE 9th European Symposium on Security and Privacy (EuroS&P)*, pages 658–675. IEEE, 2024.

[FG21] Scott Fujimoto and Shixiang (Shane) Gu. A minimalist approach to offline reinforcement learning. In *Advances in Neural Information Processing Systems*, 2021.

[FKE+15] Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Tobias Springenberg, Manuel Blum, and Frank Hutter. Efficient and robust automated machine learning. In *Advances in Neural Information Processing Systems*, 2015.

[FOS23] Hamdi Friji, Alexis Olivereau, and Mireille Sarkiss. Efficient network representation for GNN-based intrusion detection. In *Applied Cryptography and Network Security (ACNS 2023)*, volume 13905 of *Lecture Notes in Computer Science*, pages 532–554. Springer, 2023.

[FWSE11] Jérôme François, Shaonan Wang, Radu State, and Thomas Engel. Bottrack: Tracking botnets using netflow and pagerank. In *Proceedings of the 10th International IFIP TC 6 Networking Conference (NETWORKING 2011)*, volume 6640 of *Lecture Notes in Computer Science*, pages 1–14, Berlin, Heidelberg, 2011. Springer.

[GAL+16] Amin Ghafouri, Waseem Abbas, Aron Laszka, Yevgeniy Vorobeychik, and Xenofon Koutsoukos. Optimal thresholds for anomaly-based intrusion detection in dynamical environments. In *7th Conference on Decision and Game Theory for Security (GameSec)*, volume 9996 of *LNCS*, pages 415–434, 2016.

[GGSZ14] Sebastian García, Martin Grill, Jan Stiborek, and Alejandro Zunino. An empirical comparison of botnet detection methods. *Computers & Security*, 45:100–123, 2014.

[GKRB09] Nico Görnitz, Marius Kloft, Konrad Rieck, and Ulf Brefeld. Active learning for network intrusion detection. In *Proceedings of AISec*, pages 47–54, 2009.

[GLZ25] Ragini Gupta, Shinan Liu, and Ruixiao Zhang. Generative active adaptation for drifting and imbalanced network intrusion detection. In *Proceedings of KDD*, pages 1–13, 2025.

[Goo25] Google Transparency Report. Https encryption on the web. `https://transparencyreport.google.com/https?hl=en`, 2025. Accessed 2025-11-12.

[GPE20] Sebastián García, Agustín Parmisano, and María José Erquiaga. Iot-23: A labeled dataset with malicious and benign iot network traffic, 2020. Dataset.

[GPM+14] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems 27 (NeurIPS 2014)*, pages 2672–2680, 2014.

[GPM+21] Bogdan Groza, Lucian Popa, Pal-Stefan Murvay, Yuval Elovici, and Asaf Shabtai. CANARY - a reactive defense mechanism for controller area networks based on active RelaYs. In *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, August 2021.

[GPZL08] Guofei Gu, Roberto Perdisci, Junjie Zhang, and Wenke Lee. Botminer: clustering analysis of network traffic for protocol- and structure-independent botnet detection. In *Proceedings of the 17th Conference on Security Symposium*, SS'08. USENIX Association, 2008.

[GSS15] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations (ICLR 2015), San Diego, CA, USA, May 7–9 2015, Conference Track Proceedings*, 2015.

[GWB24] Akul Goyal, Gang Wang, and Adam Bates. R-CAID: Embedding root cause analysis within provenance-based intrusion detection. In *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2024.

[GŽB+14] João Gama, Indrė Žliobaitė, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. A survey on concept drift adaptation. *ACM Computing Surveys*, 46(4):44:1–44:37, 2014.

[HA85] Lawrence Hubert and Phipps Arabie. Comparing partitions. *Journal of Classification*, 2(1):193–218, 1985.

[HCKB19] Sahand Hariri, Matias Carrasco Kind, and Robert J. Brunner. Extended isolation forest. *IEEE Transactions on Knowledge and Data Engineering*, 33(4):1479–1489, 2019.

[HCL06] R. Hadsell, S. Chopra, and Y. LeCun. Dimensionality reduction by learning an invariant mapping. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, 2006.

[HCL+18] Kyle Hundman, Valentino Constantinou, Christopher Laporte, Ian Colwell, and Tom Soderstrom. Detecting spacecraft anomalies using lstms and nonparametric dynamic thresholding. In *24th ACM SIGKDD Int. Conf. on Knowledge Discovery & Data Mining*, pages 387–395, 2018.

[HJT21] Nargiz Humbatova, Gunel Jahangirova, and Paolo Tonella. DeepCrime: Mutation testing of deep learning systems based on real faults. In *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA '21)*, pages 67–78, New York, NY, USA, 2021. Association for Computing Machinery.

[HKV19] Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren, editors. *Automated Machine Learning: Methods, Systems, Challenges*. Springer, Cham, 2019.

[HLBP21] Ryan Heartfield, George Loukas, Anatolij Bezemskij, and Emmanouil Panaousis. Self-configurable cyber-physical intrusion detection for smart homes using reinforcement learning. *IEEE Transactions on Information Forensics and Security*, 16:1720–1735, 2021.

[HLWX23] Zijun Hang, Yuliang Lu, Yongjie Wang, and Yi Xie. Flow-MAE: Leveraging Masked AutoEncoder for Accurate, Efficient and Robust Malicious Traffic Classification. In *Proceedings of the 26th International Symposium on Research in Attacks, Intrusions and Defenses (RAID '23)*, pages 297–314, Hong Kong, China, 2023. Association for Computing Machinery.

[HMX+19] Qiang Hu, Lei Ma, Xiaofei Xie, Bing Yu, Yang Liu, and Jianjun Zhao. DeepMutation++: A mutation testing framework for deep learning systems. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 1158–1161. IEEE, 2019.

[HPB+20] Xueyuan Han, Thomas Pasquier, Adam Bates, James Mickens, and Margo Seltzer. UNICORN: Runtime provenance-based detector for advanced persistent threats. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*. Internet Society, 2020.

[HRLR08] Thomas R. Henderson, George F. Riley, Mathieu Lacage, and Sumit Roy. Network simulations with the ns-3 simulator. In *ACM SIGCOMM Conference Demonstrations*, page 527, 2008.

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[HWL23] Chen Hu, Tianyu Wang, and Zhiqiang Lin. Anomaly detection in the open world: Normality-shift detection, explanation, and adaptation. In *Proceedings of the 2023 Network and Distributed System Security Symposium (NDSS)*, pages 1–15, 2023.

[HWS+24] Kevin Hsieh, Mike Wong, Santiago Segarra, Sathiya Kumaran Mani, Trevor Eberl, Anatoliy Panasyuk, Ravi Netravali, Ranveer Chandra, and Srikanth Kandula. NetVigil: Robust and Low-Cost anomaly detection for East-West data center security. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, pages 1771–1789, Santa Clara, CA, April 2024. USENIX Association.

[Inc15] Cisco Systems Inc. Trex realistic traffic generator. `https://trex-tgn.cisco.com/`, 2015. Accessed 31 July 2025.

[IPS+23] Moses Ike, Kandy Phan, Keaton Sadoski, Romuald Valme, and Wenke Lee. Scaphy: Detecting modern ics attacks by correlating behaviors in scada and physical. In *2023 IEEE Symposium on Security and Privacy (SP)*, pages 20–37, 2023.

[JBEM+25] Baoxiang Jiang, Tristan Bilot, Nour El Madhoun, Khaldoun Al Agha, Anis Zouaoui, Shahrear Iqbal, Xueyuan Han, and Thomas Pasquier. Orthrus: achieving high quality of attribution in provenance-based intrusion detection systems. In *Proceedings of the 34th USENIX Conference on Security Symposium*, SEC '25, USA, 2025. USENIX Association.

[JDO+17] Max Jaderberg, Valentin Dalibard, Simon Osindero, Wojciech M. Czarnecki, Jeff Donahue, and et al. Population based training of neural networks. In *Proceedings of the 5th International Conference on Learning Representations (ICLR)*, 2017.

[JLGJ+24] Xi Jiang, Shinan Liu, Aaron Gember-Jacobson, Arjun Nitin Bhagoji, Paul Schmitt, Francesco Bronzino, and Nick Feamster. Netdiffusion: Network data augmentation through protocol-constrained traffic generation. *Proc. ACM Meas. Anal. Comput. Syst.*, 8(1), February 2024.

[JM22] P. Jairu and A. B. Mailewa. Network anomaly uncovering on cicids–2017 dataset: A supervised artificial intelligence approach. In *Proceedings of the IEEE International Conference on Electro/Information Technology (EIT)*, pages 347–352. IEEE, 2022. Feature-selection study that isolates 14 most discriminative NetFlow-style attributes.

[JMZ+20] Wuwei Jin, Lei Ma, Yangsong Zhang, Yoshinobu Kano, and Jianjun Zhao. Deepmutation++: A comprehensive mutation testing framework for deep learning systems. In *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2020.

[Jol02] Ian T. Jolliffe. *Principal Component Analysis*. Springer Series in Statistics. Springer, 2 edition, 2002.

[JSD+17] Roberto Jordaney, Kumar Sharad, Santanu Kumar Dash, Zhi Wang, Davide Papini, Ilia Nouretdinov, and Lorenzo Cavallaro. TRANSCEND: Detecting concept

drift in malware classification models. In *26th USENIX Security Symposium (USENIX Security '17)*, pages 625–642, Vancouver, BC, Canada, 2017. USENIX Association.

[JWGZ24] Zian Jia, Haohuang Wen, Zhongshu Gu, and Tao Zhang. MAGIC: Detecting Advanced Persistent Threats via Masked Graph Representation Learning. In *33rd USENIX Security Symposium*, 2024. To appear.

[K+] Prannay Khosla et al. Supervised contrastive learning. *Advances in neural information processing systems*.

[KDZ24] Yeongwoo Kim, György Dán, and Quanyan Zhu. Human-in-the-loop cyber intrusion detection using active learning. *IEEE Transactions on Information Forensics and Security*, 19(6):8658–8674, 2024.

[KFY19] Jinhan Kim, Robert Feldt, and Shin Yoo. Guiding deep learning system testing using surprise adequacy. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, pages 1039–1049. IEEE, 2019.

[KFY23] Jinhan Kim, Robert Feldt, and Shin Yoo. Evaluating surprise adequacy for deep learning system testing. *ACM Transactions on Software Engineering and Methodology*, 32(2):42:1–42:29, 2023.

[KGB17] Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. Adversarial examples in the physical world. In *5th International Conference on Learning Representations (ICLR 2017), Toulon, France, April 24–26 2017, Workshop Track Proceedings*. OpenReview.net, 2017.

[KGVK19] Ansam Khraisat, Iqbal Gondal, Peter Vamplew, and Joarder Kamruzzaman. Survey of intrusion detection systems: techniques, datasets and challenges. *Cybersecurity*, 2(20), 2019.

[KJJ+23] Sian Kim, Changhun Jung, Rhongho Jang, David Mohaisen, and DaeHun Nyang. A robust counting sketch for data plane intrusion detection. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*. Internet Society, 2023.

[KK04] H. A. Kim and Brad Karp. Autograph: Toward automated, distributed worm signature detection. In *Proceedings of the 13th USENIX Security Symposium*, pages 271–286. USENIX Association, 2004.

[KL25] Naomi Kim and Seungwon Lee. Continual learning with strategic selection and forgetting for network intrusion detection. *Computers & Security*, 128:103003, 2025.

[KLL+22] Minji Kim, Dongeun Lee, Kookjin Lee, Doowon Kim, Sangman Lee, and Jinoh Kim. Deep sequence models for packet stream analysis and early decisions. In *2022 IEEE 47th Conference on Local Computer Networks (LCN)*, pages 56–63, 2022.

[Ko00] Calvin Ko. Logic induction of valid behavior specifications for intrusion detection. In *Proc. IEEE Symposium on Security and Privacy*, pages 142–153, 2000.

[KRN+25] Diana Kramer, Lambert Rosique, Ajay Narotam, Elie Bursztein, Patrick Gage Kelley, Kurt Thomas, and Allison Woodruff. Integrating large language models into security incident response. In *Proceedings of the Twenty-First Symposium on Usable Privacy and Security (SOUPS 2025)*, pages 133–148. USENIX Association, 2025.

[KSJ+23] Isaiah J. King, Xiaokui Shu, Jiyong Jang, Kevin Eykholt, Taesung Lee, and H. Howie Huang. Edgetorrent: Real-time temporal graph representations for intrusion detection. In *Proceedings of the 26th International Symposium on Research in Attacks, Intrusions and Defenses*, RAID '23, 2023.

[KSK25] Ippokratis Koukoulis, Ilias Syrigos, and Thanasis Korakis. Self-supervised transformer-based contrastive learning for intrusion detection systems. arXiv preprint arXiv:2505.08816, 2025.

[KW17] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations (ICLR)*, 2017.

[KYC+19] Wontae Kim, Sanghyun Yoo, Junhee Cho, Robert Feldt, and Shin Yoo. Surprise adequacy: A broadly applicable metric for the testing and monitoring of deep learning systems. In *Proceedings of the 41st International Conference on Software Engineering (ICSE)*, 2019.

[KZTL20] Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative q-learning for offline reinforcement learning. In *Advances in Neural Information Processing Systems*, 2020.

[LBBH98] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[LBC+06] Xin Li, Fang Bian, Mark Crovella, Christophe Diot, Ramesh Govindan, Gianluca Iannaccone, and Anukool Lakhina. Detection and identification of network anomalies using sketch subspaces. In *Proc. 6th ACM SIGCOMM Internet Measurement Conf. (IMC 2006)*, pages 147–152. ACM, 2006.

[LCD04] Anukool Lakhina, Mark Crovella, and Christophe Diot. Characterization of network-wide anomalies in traffic flows. In *Proceedings of the 4th ACM SIGCOMM Conference on Internet Measurement (IMC '04)*, pages 201–206, Taormina, Italy, 2004. ACM.

[Led24] Brianna Leddy. Pikabot malware: Insights, impact & attack analysis. `https://www.darktrace.com/blog/pikabot-malware-battling-a-fast-moving-loader-malware-in-the-wild`, 2024. Darktrace *Inside the SOC* blog, 19 Mar 2024.

[LEL+22] Lisa Liu, Gints Engelen, Timothy M. Lynar, Daryl L. Essam, and Wouter Joosen. Error prevalence in nids datasets: A case study on cic-ids-2017 and cse-cic-ids-2018. *2022 IEEE Conference on Communications and Network Security (CNS)*, 2022.

[LGH+22] Maxime Lanvin, Pierre-François Gimenez, Yulong Han, Frédéric Majorczyk, Ludovic Mé, and Eric Totel. Errors in the CICIDS2017 dataset and the significant differences in detection performances it makes. In *17th Int. Conf. on Risks and Security of Internet and Systems (CRiSIS)*, 2022.

[LGH+23] Maxime Lanvin, Pierre-François Gimenez, Yufei Han, Frédéric Majorczyk, Ludovic Mé, and Eric Totel. Towards understanding alerts raised by unsupervised network intrusion detection systems. In *Proceedings of the 26th International Symposium on Research in Attacks, Intrusions and Defenses*, RAID '23, page 135–150, New York, NY, USA, 2023. Association for Computing Machinery.

[LHF+00] Richard Lippmann, Joshua W. Haines, David J. Fried, Jonathan Korba, and Kumar Das. The 1999 DARPA off-line intrusion detection evaluation. *Computer Networks*, 34(4):579–595, 2000.

[LJW+20] Zinan Lin, Alankar Jain, Chen Wang, Giulia C. Fanti, and Vyas Sekar. Using gans for sharing networked time series data: Challenges, initial promise, and open questions. In *Proceedings of the ACM Internet Measurement Conference (IMC '20)*, IMC '20, pages 464–483, New York, NY, USA, 2020. Association for Computing Machinery.

[LLD+19] Jie Lu, Anjin Liu, Fan Dong, Feng Gu, João Gama, and Guangquan Zhang. Learning under concept drift: A review. *IEEE Transactions on Knowledge and Data Engineering*, 31(12):2346–2363, 2019.

[LLP23] Siamak Layeghy, Wayne W. Lo, and Marius Portmann. DI-NIDS: Domain-invariant network intrusion detection system. *Knowledge-Based Systems*, 273:110626, 2023.

[LNL24] Sara Lorenzo, Diego Nieto, and Jorge López. Online self-supervised deep learning for intrusion detection systems. *IEEE Transactions on Information Forensics and Security*, 19(6):923–937, 2024.

[LPG24] Mengyuan Li, Tao Peng, and Xiaolin Gui. SPIDER: A semi-supervised continual learning-based network intrusion detection system. In *Proceedings of IEEE INFOCOM*, pages 1450–1459, 2024.

[LSG+23] Myles J. Lewis, Athina Spiliopoulou, Katriona Goldmann, Costantino Pitzalis, Paul McKeigue, and Michael R. Barnes. nestedcv: an R package for fast implementation of nested cross-validation with embedded feature selection designed for transcriptomics and high-dimensional data. *Bioinformatics Advances*, 3(1):vbad048, 2023.

[LSM99] Wenke Lee, Salvatore J. Stolfo, and Kui W. Mok. A data mining framework for building intrusion detection models. In *Proceedings of the 1999 IEEE Symposium on Security and Privacy*, pages 120–132, 1999. MADAM ID framework.

[LTL+24] Yibing Liu, Chris Xing Tian, Haoliang Li, Lei Ma, and Shiqi Wang. Neuron activation coverage: Rethinking out-of-distribution detection and generalization. *To appear in Proc. ICLR 2024*, 2024. arXiv:2306.02879.

[LTZ08a] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. Isolation forest. In *Proceedings of the 2008 Eighth IEEE International Conference on Data Mining*, pages 413–422. IEEE, 2008.

[LTZ08b] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. Isolation forest. In *Proceedings of the 8th IEEE International Conference on Data Mining (ICDM)*, pages 413–422, 2008.

[LWL+23] Peiyang Li, Ye Wang, Qi Li, Zhuotao Liu, Ke Xu, Ju Ren, Zhiying Liu, and Ruilin Lin. Learning from limited heterogeneous training data: Meta-learning for unsupervised zero-day web attack detection across web domains. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security (CCS '23)*, Copenhagen, Denmark, 2023. Association for Computing Machinery.

[Mal05] Børge Malmedal. Using netflow for slow portscan detection. Master's thesis, Norwegian University of Science and Technology (NTNU), 2005.

[MDES18] Yisroel Mirsky, Tomer Doitshman, Yuval Elovici, and Asaf Shabtai. Kitsune: An Ensemble of Autoencoders for Online Network Intrusion Detection. In *25th Annual Network and Distributed System Security Symposium (NDSS '18)*, 2018.

[MFCMC+18] Gabriel Maciá-Fernández, José Camacho, Roberto Magán-Carrión, Pedro García-Teodoro, and Roberto Therón. Ugr'16: A new dataset for the evaluation of cyclostationarity-based network idss. *Computers & Security*, 73:411–424, 2018.

[MJXX+19] Lei Ma, Felix Juefei-Xu, Minhui Xue, Bo Li, Li Li, Yang Liu, and Jianjun Zhao. DeepCT: Tomographic combinatorial testing for deep learning systems. In *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 614–618. IEEE, 2019.

[MJXZ+18] Lei Ma, Felix Juefei-Xu, Fuyuan Zhang, Jiyuan Sun, Minhui Xue, Bo Li, Chunyang Chen, Ting Su, Li Li, Yang Liu, Jianjun Zhao, and Yadong Wang. DeepGauge: Multi-granularity testing criteria for deep learning systems. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering (ASE '18)*, pages 120–131, New York, NY, USA, 2018. Association for Computing Machinery.

[MJZ+18] Lei Ma, Felix Juefei-Xu, Jianjun Zhao, Qiang Liu, Tiejun Gao, Yunhua Tian, Xiangyu Wang, Yadong Wang, Yao Feng, and Hiroshi Suwa. Deepmutation: Mutation testing of deep learning systems. In *IEEE International Conference on Software Quality, Reliability and Security (QRS)*, 2018.

[MLC] Gerhard Münz, Sa Li, and Georg Carle. Traffic anomaly detection using k-means clustering. In *Proceedings of Leistungs-, Zuverlässigkeits- und Verlässlichkeitsbewertung von Kommunikationsnetzen und Verteilten Systemen (4th GI/ITG-Workshop MMBnet 2007).*

[MLL+24] Liam Daly Manocchio, Siamak Layeghy, Wai Weng Lo, Gayan K. Kulatilleke, Mohanad Sarhan, and Marius Portmann. Flowtransformer: A transformer framework for flow-based network intrusion detection systems. *Expert Systems with Applications*, 2024.

[MR25] Carlos Martín and Isabel Rekhis. CND-IDS: Continual novelty detection for intrusion detection systems. In *Proceedings of the 32nd USENIX Security Symposium*, pages 1983–1999, 2025.

[MS15] Nour Moustafa and Jill Slay. UNSW-NB15: A comprehensive data set for network intrusion detection systems (unsw-nb15 network data set). In *2015 Military Communications and Information Systems Conference (MilCIS)*, pages 1–6, Canberra, Australia, 2015. IEEE.

[MSM+24] Seyed Mohammad Mehdi Mirnajafizadeh, Ashwin Raam Sethuram, David Mohaisen, DaeHun Nyang, and Rhongho Jang. Enhancing network attack detection with distributed and in-network data collection system. In *33rd USENIX Security Symposium (USENIX Security 24)*, Philadelphia, PA, USA, 2024. USENIX Association.

[MT16] Aditya P. Mathur and Nils Ole Tippenhauer. SWaT: A water treatment testbed for research and training on ICS security. In *2016 International Workshop on Cyber-Physical Systems for Smart Water Networks (CySWater)*. IEEE, 2016.

[NAN22] Santosh Kumar Nukavarapu, Mohammed Ayyat, and Tamer Nadeem. MirageNet – towards a gan-based framework for synthetic network traffic generation. In *2022 IEEE Global Communications Conference (GLOBECOM)*, pages 3089–3095. IEEE, 2022.

[NG24] Mert Nakıp and Erol Gelenbe. Online self-supervised deep learning for intrusion detection systems. *IEEE Transactions on Information Forensics and Security*, 2024.

[NKS05] James Newsome, Brad Karp, and Dawn Song. Polygraph: Automatically generating signatures for polymorphic worms. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, pages 226–241. IEEE, 2005.

[NLD+19] Quoc Phong Nguyen, Kar Wai Lim, Dinil Mon Divakaran, Kian Hsiang Low, and Mun Choon Chan. Gee: A gradient-based explainable variational autoencoder for network anomaly detection. In *2019 IEEE Conference on Communications and Network Security (CNS)*, pages 91–99. IEEE, 2019.

[NMH+10] Shishir Nagaraja, Prateek Mittal, Chi-Yao Hong, Matthew Caesar, and Nikita Borisov. Botgrep: Finding p2p bots with structured graph analysis. In *Proceedings of the 19th USENIX Security Symposium*, 2010.

[NMW24] Nadia Niknami, Vahid Mahzoon, and Jie Wu. PTN-IDS: Prototypical network solution for the few-shot detection in intrusion detection systems. In *Proceedings of the 49th IEEE Conference on Local Computer Networks (LCN)*. IEEE, 2024.

[NP15] Jaehyun Nam and KyoungSoo Park. Haetae: Scaling the Performance of Network Intrusion Detection with Many-Core Processors. In *18th International Symposium on Research in Attacks, Intrusions, and Defenses (RAID)*, pages 253–275, 2015.

[Ope] Open Information Security Foundation. Suricata: Open-Source IDS/IPS and Network-Security-Monitoring Engine . Accessed: 2025-05-25.

[Owe23] Philippe Owezarski. Investigating adversarial attacks against random forest-based network attack detection systems. In *NOMS 2023-2023 IEEE/IFIP Network Operations and Management Symposium*, 2023.

[PASR21] Fabio Pierazzi, Giovanni Apruzzese, Markus Schnitzer, and Konrad Rieck. Insomnia: Toward concept-drift robustness in network intrusion detection. In *Proceedings of the 28th ACM Conference on Computer and Communications Security (CCS)*, pages 1675–1690, 2021.

[Pax99] Vern Paxson. Bro: A system for detecting network intruders in real-time. *Computer Networks*, 31(23–24):2435–2463, 1999.

[PCYJ17] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. Deepxplore: Automated whitebox testing of deep learning systems. In *Proceedings of the 26th ACM Symposium on Operating Systems Principles (SOSP)*, 2017.

[PSPW23] Julien Piet, Aashish Sharma, Vern Paxson, and David Wagner. Network detection of interactive SSH impostors using deep learning. In *32nd USENIX Security Symposium (USENIX Security 23)*, Anaheim, CA, USA, 2023. USENIX Association.

[PWB19] Philipp Probst, Marvin N. Wright, and Anne-Laure Boulesteix. Hyperparameters and tuning strategies for random forest. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 9(3):e1301, 2019.

[QML+23] Jian Qu, Xiaobo Ma, Jianfeng Li, Xiapu Luo, Lei Xue, Junjie Zhang, Zhenhua Li, Li Feng, and Xiaohong Guan. An Input-Agnostic hierarchical deep learning framework for traffic fingerprinting. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 589–606, Anaheim, CA, August 2023. USENIX Association.

[QML24] Jian Qu, Xiaobo Ma, and Jianfeng Li. Trafficgpt: Breaking the token barrier for efficient long traffic analysis and generation. *arXiv preprint arXiv:2403.05822*, 2024.

[Qui93] J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.

[RAHL19] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V. Le. Regularized evolution for image classifier architecture search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4780–4789, 2019.

[RHJT21a] Vincenzo Riccio, Nargiz Humbatova, Gunel Jahangirova, and Paolo Tonella. Deepmetis: Augmenting a deep learning test set to increase its mutation score. In *Proceedings of the 36th IEEE/ACM International Conference on Automated Software Engineering*, ASE '21. IEEE/ACM, 2021.

[RHJT21b] Vincenzo Riccio, Nargiz Humbatova, Gunel Jahangirova, and Paolo Tonella. DeepMetis: Augmenting a deep learning test set to increase its mutation score. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 355–367. IEEE, 2021.

[RHW86] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

[Roe99] Martin Roesch. Snort: Lightweight intrusion detection for networks. In *Proceedings of the 13th USENIX Conference on System Administration (LISA '99)*, pages 229–238. USENIX Association, 1999.

[RP23] Andrea Rossi and Fabio Pierazzi. Augmented memory replay-based continual learning approaches for network intrusion detection. In *Proceedings of NeurIPS*, pages 1–12, 2023.

[RSRD07a] Haakon Ringberg, Augustin Soule, Jennifer Rexford, and Christophe Diot. Sensitivity of pca for traffic anomaly detection. In *Proceedings of ACM SIGMETRICS 2007*, pages 109–120, San Diego, CA, USA, 2007. ACM.

[RSRD07b] Haakon Ringberg, Augustin Soule, Jennifer Rexford, and Christophe Diot. Sensitivity of pca for traffic anomaly detection. In *Proceedings of ACM SIGMETRICS*, pages 109–120. ACM, 2007.

[RSW+23] Phillip Rieger, Björn Spielvogel, Konrad Wolsing, Marc-Oliver Pahl, and Frank Kargl. ARGUS: Context-Based Detection of Stealthy IoT Infiltration Attacks. In *32nd USENIX Security Symposium*, pages 3507–3524, 2023.

[RWS+19a] Markus Ring, Sarah Wunderlich, Deniz Scheuring, Dieter Landes, and Andreas Hotho. A survey of network-based intrusion detection data sets. *Computers & Security*, 86:147–167, 2019.

[RWS+19b] Markus Ring, Sarah Wunderlich, Deniz Scheuring, Dieter Landes, and Andreas Hotho. A survey of network-based intrusion detection data sets. *Computers & Security*, 86:147–167, 2019. arXiv:1903.02460.

[RWS+19c] Michael Ring, Sarah Wunderlich, Daniel Scheuring, Dieter Landes, and Andreas Hotho. A survey of network-based intrusion detection data sets. *Computers & Security*, 2019.

[SBC+19] Tohid Shekari, Christian Bayens, Morris Cohen, Lukas Graber, and Raheem Beyah. RFDIDS: Radio frequency-based distributed intrusion detection system for the power grid. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*. Internet Society, 2019.

[SBDB02] R. Sekar, M. Bendre, D. Dhurjati, and P. Bollineni. A fast automaton-based method for detecting anomalous program behaviors. In *IEEE Symposium on Security and Privacy*, pages 144–155, 2002.

[SBF+23] Khaled Serag, Rohit Bhatia, Akram Faqih, Muslum Ozgur Ozmen, Vireshwar Kumar, Z. Berkay Celik, and Dongyan Xu. ZBCAN: A zero-byte CAN defense system. In *32nd USENIX Security Symposium (USENIX Security 23)*, Anaheim, CA, USA, 2023. USENIX Association.

[SC24] Md Hasan Shahriar and Roy H. Campbell. CANtropy: Time Series Feature Extraction–Based Intrusion Detection for Controller Area Networks. In *Proceedings of the 31st Network and Distributed System Security Symposium (NDSS)*, 2024. To appear.

[SFL+99] Salvatore Stolfo, Wei Fan, Wenke Lee, Andreas Prodromidis, and Phillip Chan. KDD Cup 1999 Data, 1999. Dataset.

[SG02] Alexander Strehl and Joydeep Ghosh. Cluster ensembles: a knowledge reuse framework for combining multiple partitions. *Journal of Machine Learning Research*, 3:583–617, 2002.

[Sha18] Iman et al. Sharafaldin. Toward generating a new intrusion detection dataset and intrusion traffic characterization. In *4th International Conference on Information Systems Security and Privacy*, 2018.

[SHG+15] D. Sculley, G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner, V. Chaudhary, M. Young, J.-F. Crespo, and D. Dennison. Hidden technical debt in machine learning systems. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2015.

[SHLG18a] Iman Sharafaldin, Arash Habibi Lashkari, and Ali A. Ghorbani. Toward generating a new intrusion detection dataset and intrusion traffic characterization. In *Proceedings of the 4th International Conference on Information Systems Security and Privacy (ICISSP 2018)*, pages 108–116. SCITEPRESS, January 2018.

[SHLG18b] Iman Sharafaldin, Arash Habibi Lashkari, and Ali A. Ghorbani. Toward generating a new intrusion detection dataset and intrusion traffic characterization. In *Proceedings of the 4th International Conference on Information Systems Security and Privacy (ICISSP)*, pages 108–116. SciTePress, 2018.

[SIZ+24] Methaq A. Shyaa, Noor Farizah Ibrahim, Zurinahni Zainol, Rosni Abdullah, Mohammed Anbar, and Laith Alzubaidi. Evolving cybersecurity frontiers: A comprehensive survey on concept drift and feature dynamics aware machine and deep learning in intrusion detection systems. *Engineering Applications of Artificial Intelligence*, 137:109143, November 2024.

[SKL+23] Jiwoo Shin, Hyunghoon Kim, Seyoung Lee, Wonsuk Choi, Dong Hoon Lee, and Hyo Jin Jo. RIDAS: Real-time identification of attack sources on controller area networks. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 6911–6928, Anaheim, CA, USA, 2023. USENIX Association.

[SLA12] Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems*, volume 25, pages 2951–2959, 2012.

[SLH23] Md Hasan Shahriar, Wenjing Lou, and Y Thomas Hou. Cantropy: Time series feature extraction-based intrusion detection systems for controller area networks. In *Proceedings of Symposium on Vehicles Security and Privacy (VehicleSec)*, pages 1–8, 2023.

[SLPR15] Justine Sherry, Chang Lan, Raluca A. Popa, and Sylvia Ratnasamy. Blindbox: Deep packet inspection over encrypted traffic. In *SIGCOMM '15 Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, pages 213–226, New York, NY, USA, 2015. Association for Computing Machinery. SIGCOMM '15, London, United Kingdom, August 17–21, 2015.

[SM07] Karen Scarfone and Peter Mell. Guide to intrusion detection and prevention systems (IDPS). Special Publication 800-94, National Institute of Standards and Technology (NIST), February 2007.

[SP10a] Robin Sommer and Vern Paxson. Outside the closed world: On using machine learning for network intrusion detection. In *2010 IEEE Symposium on Security and Privacy*, pages 305–316. IEEE, 2010.

[SP10b] Robin Sommer and Vern Paxson. Outside the closed world: On using machine learning for network intrusion detection. In *2010 IEEE Symposium on Security and Privacy*, pages 305–316. IEEE, 2010.

[SPST+01] Bernhard Schölkopf, John C. Platt, John Shawe-Taylor, Alexander J. Smola, and Robert C. Williamson. Estimating the support of a high-dimensional distribution. *Neural Computation*, 13(7):1443–1471, 2001.

[SVB20] Ankush Singla, Dinesh Verma, and Elisa Bertino. Preparing network intrusion detection deep learning models with minimal data using adversarial domain adaptation. In *15th ACM Asia Conference on Computer and Communications Security (ASIACCS '20)*, pages 305–317, Taipei, Taiwan, 2020. ACM.

[SWD+17] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[SY23] HyungBin Seo and MyungKeun Yoon. Generative intrusion detection and prevention on data stream. In *Proceedings of the 32nd USENIX Security Symposium (USENIX Security 23)*, pages 4319–4335, Anaheim, CA, USA, August 2023. USENIX Association.

[SYL+21] Yizheng Shen, Zeliang Yu, Kevin Leach, Wei Wang, Xiangyu Wang, and Suman Jana. Deepcrime: Mutation testing of deep learning systems based on real faults. In *Proceedings of the 43rd International Conference on Software Engineering (ICSE)*, 2021.

[SYL+25] Peishuai Sun, Xiaochun Yun, Shuhao Li, Tao Yin, Chengxiang Si, and Jiang Xie. AdvTG: An adversarial traffic generation framework to deceive DL-based malicious traffic detection models. In *Proceedings of the ACM on Web Conference 2025 (WWW 2025), Sydney, Australia, 28 Apr – 2 May 2025*, pages 3147–3159. ACM, 2025.

[TBLG09] Mohammad Tavallaee, Ebrahim Bagheri, Wei Lu, and Ali A. Ghorbani. A detailed analysis of the KDD CUP 99 data set. In *Proceedings of the 2009 IEEE Symposium on Computational Intelligence for Security and Defence Applications (CISDA)*, pages 1–6, Ottawa, Canada, 2009. IEEE.

[TDK+24] Saravanan Thirumuruganathan, Fatih Deniz, Issa Khalil, Ting Yu, Mohamed Nabeel, and Mourad Ouzzani. Detecting and mitigating sampling bias in cybersecurity with unlabeled data. In *Proceedings of the 33rd USENIX Security Symposium*, Philadelphia, PA, USA, 2024. USENIX Association.

[TKW+20] Michele Tufano, Jason Kimko, Shiya Wang, Cody Watson, Gabriele Bavota, Massimiliano Di Penta, and Denys Poshyvanyk. DeepMutation: A neural mutation

tool. In *Proceedings of the 42nd International Conference on Software Engineering: Companion Proceedings (ICSE '20)*, pages 29–32, New York, NY, USA, 2020. Association for Computing Machinery.

[TMH11] Gautam Thatte, Urbashi Mitra, and John Heidemann. Parametric methods for anomaly detection in aggregate traffic. *IEEE/ACM Transactions on Networking*, 19(2):512–525, 2011.

[UHH+24] Rafael Uetz, Marco Herzog, Louis Hackländer, Simon Schwarz, and Martin Henze. You cannot escape me: Detecting evasions of siem rules in enterprise networks. In *Proceedings of the 33rd USENIX Security Symposium (USENIX Security '24)*, pages 5179–5196, Philadelphia, PA, USA, August 2024. USENIX Association.

[uRZRX24] Mati ur Rehman, Mu Zhang, Sankardas Roy, and Xinyu Xing. FLASH: A Comprehensive Approach to Intrusion Detection via Provenance Graph Representation Learning. In *IEEE Symposium on Security and Privacy (S&P)*, 2024. To appear.

[Van19] Joaquin Vanschoren. Meta-learning. In Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren, editors, *Automated Machine Learning: Methods, Systems, Challenges*, pages 39–61. Springer, Cham, 2019.

[VD04] Jouni Viinikka and Hervé Debar. Monitoring IDS background noise using EWMA control charts and alert information. In *Recent Advances in Intrusion Detection (RAID 2004)*, volume 3224 of *Lecture Notes in Computer Science*, pages 166–187, Berlin, Heidelberg, 2004. Springer.

[VFMC23] Andrea Venturi, Matteo Ferrari, Mirco Marchetti, and Michele Colajanni. Arganids: A novel network intrusion detection system based on adversarially regularized graph autoencoder. In *Proceedings of the 38th ACM/SIGAPP Symposium on Applied Computing (SAC)*, pages 1540–1548. ACM, 2023.

[VKvE+23] Mathew Vermeer, Natalia Kadenko, Michel van Eeten, Carlos Gañán, and Simon Parkin. Alert alchemy: SOC workflows and decisions in the management of NIDS rules. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, pages 2770–2784. Association for Computing Machinery, 2023.

[VLBM08] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th International Conference on Machine Learning (ICML)*, pages 1096–1103, 2008.

[VRKK03] Giovanni Vigna, William K. Robertson, Vishal Kher, and Richard A. Kemmerer. A stateful intrusion detection system for world-wide web servers. In *Proceedings of the 19th Annual Computer Security Applications Conference (ACSAC 2003)*, pages 34–43, Las Vegas, NV, USA, 2003. IEEE Computer Society.

[VSP+17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

[Wan22] Xian Wang. ENIDrift: A fast and adaptive ensemble system for network intrusion detection under real-world drift. In *Proceedings of the Annual Computer Security Applications Conference (ACSAC '22)*, New York, NY, USA, 2022. Association for Computing Machinery.

[WCH+22] Ning Wang, Yimin Chen, Yang Hu, Wenjing Lou, and Y. Thomas Hou. FeCo: Boosting intrusion detection capability in IoT networks via contrastive learning. In *IEEE INFOCOM 2022 - IEEE Conference on Computer Communications*, pages 1409–1418, 2022.

[WCT21] Michael Weiss, Rwiddhi Chakraborty, and Paolo Tonella. A review and refinement of surprise adequacy. In *2021 IEEE/ACM Third International Workshop on Deep Learning for Testing and Testing for Deep Learning (DeepTest)*, pages 17–24. IEEE, 2021.

[WFM+24] Chao Wang, Alessandro Finamore, Pietro Michiardi, Massimo Gallo, and Dario Rossi. Data augmentation for traffic classification. In Philipp Richter, Vaibhav Bajpai, and Esteban Carisimo, editors, *Passive and Active Measurement: 25th International Conference, PAM 2024, Virtual Event, March 11–13, 2024, Proceedings, Part I*, volume 14537 of *Lecture Notes in Computer Science*, pages 159–186. Springer Nature Switzerland, 2024.

[WFSE] Cynthia Wagner, Jérôme François, Radu State, and Thomas Engel. Machine learning approach for ip-flow record anomaly detection. In *NETWORKING 2011*.

[WHC+19] Xiang Wang, Yang Hong, Harry Chang, KyoungSoo Park, Geoff Langdale, Jiayu Hu, and Heqing Zhu. Hyperscan: A fast multi-pattern regex matcher for modern cpus. In *Proceedings of the 16th USENIX Symposium on Networked Systems Design and Implementation (NSDI'19)*, pages 631–648. USENIX Association, 2019.

[WHW24] Yali Wu, Yanghu Hu, and Junhu Wang. An active learning framework using deep q-network for zero-day attack detection. *Computers & Security*, 139:103713, 2024.

[WJM22] Abdul Waleed, A. F. Jamali, and Ahsan Masood. Which open-source ids? snort, suricata or zeek. *Computer Networks*, 213:109116, 2022.

[WLZH23] Feng Wei, Hongda Li, Ziming Zhao, and Hongxin Hu. XNIDS: Explaining deep learning-based network intrusion detection systems for active intrusion responses. In *32nd USENIX Security Symposium (USENIX Security 23)*, Anaheim, CA, USA, 2023. USENIX Association.

[WP05] Arno Wagner and Bernhard Plattner. Entropy based worm and anomaly detection in fast IP networks. In *Proceedings of the 14th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE '05)*, pages 172–177, Linköping, Sweden, 2005. IEEE.

[WPS06] Ke Wang, Janak J. Parekh, and Salvatore J. Stolfo. Anagram: A content anomaly detector resistant to mimicry attack. In *Recent Advances in Intrusion Detection (RAID 2006)*, volume 4219 of *Lecture Notes in Computer Science*, pages 226–248. Springer, 2006.

[WS04] Ke Wang and Salvatore J. Stolfo. Anomalous payload-based network intrusion detection. In *Proceedings of the 7th International Symposium on Recent Advances in Intrusion Detection (RAID)*, volume 3224 of *Lecture Notes in Computer Science*, pages 203–222, Sophia Antipolis, France, 2004. Springer.

[WSL+24] Lingzhi Wang, Xiangmin Shen, Weijian Li, Zhenyuan Li, R Sekar, Han Liu, and Yan Chen. Incorporating gradients to rules: Towards lightweight, adaptive provenance-based intrusion detection. *arXiv preprint arXiv:2404.14720*, 2024.

[WTvS+22] Konrad Wolsing, Lea Thiemt, Christian van Sloun, Eric Wagner, Klaus Wehrle, and Martin Henze. Can industrial intrusion detection be SIMPLE? In *Computer Security – ESORICS 2022*, pages 441–461. Springer, 2022.

[WWL+25] Konrad Wolsing, Eric Wagner, Luisa Lux, Klaus Wehrle, and Martin Henze. Gecos replacing experts: generalizable and comprehensible industrial intrusion detection. In *Proceedings of the 34th USENIX Conference on Security Symposium*, SEC '25, USA, 2025. USENIX Association.

[WWQ+24] Duo Wu, Xianda Wang, Yaqi Qiao, Zhi Wang, Junchen Jiang, Shuguang Cui, and Fangxin Wang. Netllm: Adapting large language models for networking. In *Proceedings of the ACM SIGCOMM 2024 Conference*, ACM SIGCOMM '24, pages 661–678, New York, NY, USA, 2024. Association for Computing Machinery.

[WYFPW24] Minxiao Wang, Ning Yang, Nicolas J. Forcade-Perkins, and Ning Weng. Progen: Projection-based adversarial attack generation against network intrusion detection. *IEEE Transactions on Information Forensics and Security*, 19:5476–5491, 2024.

[WYW24] Minxiao Wang, Ning Yang, and Ning Weng. K-getnid: Knowledge-guided graphs for early and transferable network intrusion detection. *IEEE Transactions on Information Forensics and Security*, 2024.

[WZG+25] Chenglong Wang, Pujia Zheng, Jiaping Gui, Cunqing Hua, and Wajih Ul Hassan. R+R: From claims to crashes: A systematic re-evaluation of graph-based network intrusion detection systems. In *Annual Computer Security Applications Conference (ACSAC)*, 2025.

[WZM+21] Zhongjie Wang, Shitong Zhu, Keyu Man, Pengxiong Zhu, Yu Hao, Zhiyun Qian, Srikanth V. Krishnamurthy, Tom La Porta, and Michael J. De Lucia. THEMIS: Ambiguity-aware network intrusion detection based on symbolic model comparison. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security (CCS '21)*, pages 3384–3400, New York, NY, USA, 2021. Association for Computing Machinery.

[WZS02] Haining Wang, Danlu Zhang, and Kang G. Shin. Detecting syn flooding attacks. In *Proceedings.Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 3, pages 1530–1539, 2002.

[XLW24] Yuxin Xu, Haojing Li, and Jian Wang. ReCDA: Concept drift adaptation with representation enhancement for network intrusion detection. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 2114–2124, 2024.

[XMAR21] Shengzhe Xu, Manish Marwah, Martin Arlitt, and Naren Ramakrishnan. Stan: Synthetic network traffic generation with generative neural models. In *International Workshop on Deployable Machine Learning for Security Defense*, pages 3–29. Springer, 2021.

[XMT+24] Bruno Missi Xavier, Magnos Martinello, Célio Trois, Brenno de Mello Alencar, and Ricardo Araújo Rios. Fast learning enabled by In-network drift detection. In *Proceedings of the 8th Asia–Pacific Workshop on Networking (APNet '24)*, pages 129–134, Sydney, Australia, August 2024. Association for Computing Machinery.

[XSCIV19] Lei Xu, Maria Skoularidou, Alfredo Cuesta-Infante, and Kalyan Veeramachaneni. Modeling tabular data using conditional GAN. In *Advances in Neural Information Processing Systems 32 (NeurIPS 2019)*, pages 7333–7343, 2019.

[XSD20] Congyuan Xu, Jizhong Shen, and Xin Du. A method of few-shot network intrusion detection based on meta-learning framework. *IEEE Transactions on Information Forensics and Security*, 15:3540–3552, 2020.

[XZN23] Qingfu Xie, Xiangliang Zhang, and Yacin Nadji. Rosetta: Enabling robust tls-encrypted traffic classification in adverse environments. In *32nd USENIX Security Symposium*, pages 1915–1932, 2023.

[YC01] Nong Ye and Qiang Chen. An anomaly detection technique based on a chi–square statistic for detecting intrusions into information systems. *Quality and Reliability Engineering International*, 17(2):105–112, 2001.

[YCA+18] Wenchao Yu, Wei Cheng, Charu C. Aggarwal, Kai Zhang, Haifeng Chen, and Wei Wang. Netwalk: A flexible deep embedding approach for anomaly detection in dynamic networks. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '18, page 2672–2681, New York, NY, USA, 2018. Association for Computing Machinery.

[YdHE16] Ömer Yüksel, Jerry den Hartog, and Sandro Etalle. Reading between the fields: Practical, effective intrusion detection for industrial control systems. In *Proceedings of the 31st Annual ACM Symposium on Applied Computing (SAC '16)*, pages 2063–2068, New York, NY, USA, 2016. Association for Computing Machinery.

[YES21] Aviv Yehezkel, Eyal Elyashiv, and Or Soffer. Network anomaly detection using transfer learning based on auto-encoders loss normalization. In *14th ACM Workshop on Artificial Intelligence and Security (AISec '21)*, pages 127–138. ACM, 2021.

[YHS23] Xue Yang, Enda Howley, and Micheal Schukat. Adt: Agent-based dynamic thresholding for anomaly detection. In *Adaptive Learning Agents Workshop @ AAMAS 2023*, 2023. arXiv : 2312.01488.

[YHY+20] Zeyu Yang, Liang He, David K.Y. Yau, Jiming Chen, Peng Cheng, and Linkang Du. PLC-Sleuth: Detecting and localizing PLC intrusions using control invariants. In *23rd International Symposium on Research in Attacks, Intrusions and Defenses (RAID '20)*, pages 333–348. USENIX Association, 2020.

[YLJ+22] Yucheng Yin, Zinan Lin, Minhao Jin, Giulia Fanti, and Vyas Sekar. Practical gan-based synthetic ip header trace generation using netshare. In *Proceedings of the ACM SIGCOMM 2022 Conference*, SIGCOMM '22, page 458–472, New York, NY, USA, 2022. Association for Computing Machinery.

[YLJ+25] Xu Yu, Yan Lu, Feng Jiang, Qiang Hu, Junwei Du, and Dunwei Gong. A cross-domain intrusion detection method based on nonlinear augmented explicit features. *IEEE Transactions on Network and Service Management*, 22(1):187–200, February 2025.

[YPW23] Yuanyuan Yuan, Qi Pang, and Shuai Wang. Revisiting neuron coverage for dnn testing: A layer-wise and distribution-aware criterion. In *Proc. 45th Int. Conf. on Software Engineering (ICSE)*, 2023. Extended version: arXiv:2112.01955.

[YVC03] Nong Ye, Sean Vilbert, and Qiang Chen. Computer intrusion detection through EWMA for autocorrelated and uncorrelated data. *IEEE Transactions on Reliability*, 52(1):75–82, March 2003.

[YXCG21] Yuexin Yang, Zhiyuan Xu, Peng Cheng, and Javad Ghaderi. Cade: Detecting and explaining concept drift samples for security analytics. In *Proceedings of the 30th USENIX Security Symposium*, pages 1–18, 2021.

[YXX+23] Fan Yang, Jiacen Xu, Chunlin Xiong, Zhou Li, and Kehuan Zhang. PROGRAPHER: An anomaly detection system based on provenance graph embedding. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 4355–4372, Anaheim, CA, August 2023. USENIX Association.

[YZFH17a] Chuanlong Yin, Yuefei Zhu, Jinlong Fei, and Xin-Zheng He. A deep learning approach for intrusion detection using recurrent neural networks. *IEEE Access*, 2017.

[YZFH17b] Chuanlong Yin, Yuefei Zhu, Jinlong Fei, and Xinzheng He. A deep learning approach for intrusion detection using recurrent neural networks. *IEEE Access*, 5:21954–21961, 2017.

[ZCW24] Liang Zhao, Mei Chen, and Hao Wang. AOC-IDS: An autonomous online framework with contrastive learning for intrusion detection. In *Proceedings of the IEEE INFOCOM*, pages 1–10, 2024.

[ZMM21] Eshetu M. Zeleke, Henok M. Melaku, and Fasil G. Mengistu. Efficient intrusion detection system for sdn orchestrated internet of things. *Journal of Computer Networks and Communications*, 2021:1–15, 2021. Recursive-feature-elimination shows best accuracy with the same 14 flow features.

[ZMSC21] Tianyi Zhou, Lei Ma, Yijun Sun, and Zhenyu Chen. An empirical study on surprise adequacy refinements for deep neural network testing. *IEEE Transactions on Software Engineering*, 2021.

[ZSA+20] Zhipeng Zhao, Hugo Sadok, Nirav Atre, James C. Hoe, Vyas Sekar, and Justine Sherry. Achieving 100 gbps intrusion prevention on a single server. In *Proceedings of the 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI '20)*, pages 1083–1099. USENIX Association, 2020.

[ZSM+22] Bolor-Erdene Zolbayar, Ryan Sheatsley, Patrick D. McDaniel, Michael J. Weisman, Sencun Zhu, Shitong Zhu, and Srikanth V. Krishnamurthy. Generating practical adversarial network traffic flows using NIDSGAN. *CoRR*, abs/2203.06694, 2022.

Résumé

Les systèmes de détection d'intrusion réseau (Network Intrusion Detection Systems, NIDS) basés sur l'apprentissage automatique obtiennent souvent d'excellentes performances sur des jeux de données publics. Pourtant, ces résultats reposent fréquemment sur un schéma d'usage relativement statique (entraîner une fois, évaluer une fois, puis déployer), alors que les réseaux réels évoluent en continu : les applications changent, les usages se transforment et les attaquants adaptent leurs stratégies. Cette thèse part de ce décalage entre conditions expérimentales et conditions opérationnelles et s'intéresse à deux formes de robustesse souvent sous-estimées : (i) la robustesse de la configuration des NIDS (hyperparamètres et seuils de décision) quand le contexte de déploiement change, et (ii) la robustesse de l'évaluation avant déploiement, c'est-à-dire la capacité à tester un NIDS sur des jeux de test réellement exigeants et révélateurs de ses faiblesses.

La première contribution propose un cadre de méta-apprentissage pour automatiser la configuration des NIDS. L'idée centrale est de reformuler la configuration comme un problème supervisé au niveau méta : à partir de descripteurs compacts du contexte opérationnel (méta-caractéristiques), un méta-modèle apprend à prédire une configuration proche de l'optimum. Le coût de calcul est ainsi découplé en deux étapes complémentaires : une phase hors-ligne, qui consolide l'expérience issue d'épisodes d'optimisation passés, et une phase en ligne légère, qui infère en une seule étape les hyperparamètres et le seuil de décision adaptés à un nouveau contexte. Un enjeu important, dans le cas des détecteurs d'anomalies, est la rareté (voire l'absence) de labels d'attaque : la thèse montre qu'il est possible d'inférer conjointement hyperparamètres et seuil en s'appuyant uniquement sur du trafic bénin, ce qui correspond à des scénarios réalistes de type « zero-day ».

La seconde contribution traite la robustesse de l'évaluation et propose TATA (Test-set Assessment and Targeted Augmentation), un cadre agnostique du modèle NIDS visant à diagnostiquer la qualité d'un jeu de test et à l'enrichir lorsqu'il ne stresse pas suffisamment le détecteur. TATA construit d'abord une représentation latente structurée des données (via un autoencodeur contrastif) afin d'analyser le jeu de test dans le même espace que le jeu d'entraînement. Trois métriques centrées sur les données sont définies pour quantifier la qualité du test : la diversité (couverture de l'espace des caractéristiques), la proximité (proximité des frontières de décision) et la distribution (couverture de l'ensemble des frontières). Ces métriques guident ensuite un agent d'apprentissage par renforcement connecté à un banc d'essai réseau, qui configure des générateurs afin de produire du trafic nouveau, réaliste et conforme au protocole, ciblant spécifiquement les zones sous-couvertes ou critiques identifiées lors de l'évaluation.

Les contributions sont validées expérimentalement sur des NIDS orientés flux, en s'appuyant notamment sur les jeux de données IDS2017 et IDS2018. Les résultats montrent que la configuration par méta-apprentissage retrouve des réglages de performance comparables à une optimisation exhaustive, tout en réduisant fortement le coût et le temps de reconfiguration, et confirment la faisabilité pratique de l'inférence de seuils à partir de trafic bénin. En parallèle, les métriques de TATA se révèlent corrélées à la difficulté de détection, et l'augmentation ciblée pilotée par apprentissage par renforcement s'avère plus efficace que des stratégies d'augmentation non adaptatives pour mettre en évidence des faiblesses avant déploiement.

Mots-clés : détection d'intrusion réseau, apprentissage automatique, méta-apprentissage, optimisation d'hyperparamètres, seuil de décision, qualité des jeux de test, augmentation ciblée,

apprentissage par renforcement, IDS2017, IDS2018.

Abstract

Machine learning-based Network Intrusion Detection Systems (NIDS) often achieve excellent performance on public datasets. However, these results frequently rely on a relatively static usage paradigm (train once, evaluate once, deploy), whereas real-world networks continuously evolve: applications change, usage patterns shift, and attackers adapt their strategies. This thesis addresses the discrepancy between experimental and operational conditions, focusing on two frequently underestimated forms of robustness: (i) the robustness of NIDS configuration (hyperparameters and decision thresholds) when the deployment context changes, and (ii) the robustness of pre-deployment evaluation, namely the ability to test a NIDS on genuinely demanding datasets that reveal its vulnerabilities.

The first contribution proposes a meta-learning framework to automate NIDS configuration. The core concept is to reformulate configuration as a supervised problem at the meta-level: using compact descriptors of the operational context (meta-features), a meta-model learns to predict a near-optimal configuration. The computational cost is thus decoupled into two complementary stages: an offline phase that consolidates experience from previous optimization episodes, and a lightweight online phase that infers the hyperparameters and decision threshold adapted to a new context in a single step. A significant challenge in anomaly detection is the scarcity (or even absence) of attack labels. This thesis demonstrates that it is possible to jointly infer hyperparameters and thresholds relying solely on benign traffic, corresponding to realistic zero-day scenarios.

The second contribution addresses evaluation robustness by proposing TATA (Test-set Assessment and Targeted Augmentation), a NIDS model-agnostic framework designed to diagnose test set quality and augment it when it fails to adequately stress the detector. TATA initially constructs a structured latent representation of the data via a contrastive autoencoder to analyze the test set within the same feature space as the training set. Three data-centric metrics are defined to quantify test quality: diversity (feature space coverage), proximity (closeness to decision boundaries), and distribution (coverage across all boundaries). These metrics subsequently guide a reinforcement learning agent connected to a network testbed, which configures generators to produce novel, realistic, and protocol-compliant traffic that specifically targets under-covered or critical regions identified during the evaluation.

These contributions are experimentally validated on flow-oriented NIDS, specifically utilizing the IDS2017 and IDS2018 datasets. The results demonstrate that meta-learning-based configuration recovers performance settings comparable to exhaustive optimization while significantly reducing reconfiguration time and computational cost. Furthermore, they confirm the practical feasibility of threshold inference using only benign traffic. Concurrently, the TATA metrics prove to be correlated with detection difficulty, and targeted augmentation driven by reinforcement learning is shown to be more effective than non-adaptive augmentation strategies at exposing pre-deployment vulnerabilities.

Keywords: Network intrusion detection, machine learning, meta-learning, hyperparameter optimization, decision threshold, test set quality, targeted augmentation, reinforcement learning, IDS2017, IDS2018.