Towards Local Neuro-Symbolic AI in Network Intrusion Detection Systems

Version of June 29, 2026

Raoul Saad

Towards Local Neuro-Symbolic AI in Network Intrusion Detection Systems

THESIS

submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER SCIENCE

by

Raoul Saad
born in Jouret Arsoun, Lebanon



Network Systems Research Group
Faculty EEMCS, Delft University of Technology
Delft, the Netherlands
www.ewi.tudelft.nl

©2026 Raoul Saad. All rights reserved.

Towards Local Neuro-Symbolic AI in Network Intrusion Detection Systems

Author: Raoul Saad
Student id: 5697689

Abstract

Network intrusion detection systems face demanding and often conflicting requirements: they must learn from the limited labelled attack data available in practice, maintain low false-alarm rates, remain interpretable enough for analysts to trust their alerts, and stay robust when their inputs are noisy or adversarially perturbed. Purely neural detectors rarely satisfy these requirements at once, while signature-based systems cannot detect behaviour they were not designed to recognise. This thesis investigates whether a neuro-symbolic approach can address these limitations jointly. Built on the neural probabilistic logic programming framework DeepProbLog, the proposed system couples a neural classifier with automatically mined symbolic rules within a single end-to-end differentiable model, so that symbolic reasoning informs learning, inference, and explanation. It is realised as packet-level, flow-level, and hybrid detectors and evaluated on several public intrusion detection datasets against neural, symbolic-only, and signature baselines. The results show that incorporating symbolic knowledge improves detection performance and reduces false alarms, most notably when labelled data is scarce, while the gain narrows once data is plentiful. The mined rules transfer across networks for attacks whose behaviour is defined broadly within the symbolic knowledge base, once their thresholds are calibrated to the target environment's own benign traffic. The coupling between the neural and symbolic components further acts as a configurable trade-off between clean-traffic precision and robustness under noise and adversarial perturbation. Finally, the symbolic layer produces faithful and stable explanations aligned with documented attack techniques, at a computational cost suited to practical deployment. Overall, this thesis demonstrates that integrating symbolic reasoning throughout a neural detection pipeline provides an effective and interpretable framework for network intrusion detection.

Thesis Committee:

Chair:	Prof. Dr. ir. F. Kuipers, Faculty EEMCS, TU Delft
Daily co-supervisors:	Dr. A. Agiollo, Faculty EEMCS, TU Delft
	Dr. E. Bardhi, Faculty EEMCS, TU Delft
Committee Member:	Dr. ir. S.E. Verwer, Faculty EEMCS, TU Delft

Preface

The road to finalising this thesis has been long and winding. I faced several challenges and had to overcome many obstacles, but I am proud to have reached this point. Before delving into the details of the work of the past year, I would like to take a moment to express my sincere gratitude to the people who have supported me throughout this journey.

First and foremost, I offer endless gratitude to my daily supervisors Andrea Agiollo and Enkeleda Bardhi. Without you, I would not have been able to complete such a thorough and comprehensive thesis in such a short time. You made the process of writing this thesis and running the experiments much easier and more enjoyable. I am grateful for your guidance, support, and encouragement throughout this journey. Your expertise and insights have been invaluable, and I have learned so much from both of you. Thank you for always offering a helping hand when I needed it and for all the encouragement throughout the thesis process. Most importantly, thank you for treating me with kindness, patience, and especially as more than just a student. You are excellent supervisors, and above all, wonderful people. I am beyond grateful for the opportunity to have worked with you.

I would also like to thank my thesis advisor, Prof. Dr. Fernando Kuipers. Despite your busy schedule, the many responsibilities you have, and the many students you supervise, you still took on the responsibility of guiding me through this thesis. I am grateful for the support and feedback you provided throughout this journey.

Thank you to my friends and colleagues going through the same journey. You made me feel seen and less alone throughout the entire process.

Lastly, many thanks to my family and friends for their unwavering support and encouragement. You made sure to be there for me during the ups and downs of this journey and the countless all-nighters I spent working on this thesis. I am grateful for your love, patience, and understanding throughout this process. Thank you for putting up with my stress and occasional mood swings, and for always being there to listen and offer support.

The journey has been long and challenging, but equally rewarding nonetheless. I can say with confidence that I am proud of what I have accomplished and grateful for the support and guidance I have received along the way.

My dear reader, thank you for taking the time to read this thesis. I hope you enjoy reading it, even if it can feel a bit heavy and long at times, and I hope you find the content informative and insightful. I am grateful for your interest in my work, and I hope that it

contributes meaningfully to this field.

Raoul Saad
Delft, the Netherlands
June 29, 2026

Contents

Preface iii

Contents v

List of Figures vii

1 Introduction 1

2 Related Work 5
2.1 Network Intrusion Detection Systems 5
2.2 Neuro-Symbolic Artificial Intelligence 14

3 Introducing the DeepProbLog framework in Network Intrusion Detection 23
3.1 DeepProbLog within network intrusion detection systems 23
3.2 Design assumptions and scope 25
3.3 Expected benefits 25
3.4 Limitations of the framework 26

4 Methodology 29
4.1 System overview 29
4.2 Symbolic knowledge extraction 33
4.3 Neuro-symbolic coupling 36
4.4 Threat model 37
4.5 Evaluation protocol 38
4.6 Validity considerations 40

5 Implementation 41
5.1 Datasets 41
5.2 Two-representation data storage 43
5.3 Neural classifiers 45
5.4 Offline rule-mining pipeline 46

5.5 Executable symbolic predicates . . . 47
5.6 Query construction and neural predicates . . . 48
5.7 Coupling architectures . . . 49
5.8 Worked example of rule evaluation . . . 51
5.9 Hybrid packet-flow execution . . . 52
5.10 Batched arithmetic-circuit evaluation . . . 52
5.11 Implementation boundaries . . . 53

6 Experiments and Results **55**
6.1 Experimental settings . . . 55
6.2 Experiments . . . 58
6.3 Ablation studies . . . 80

7 Discussion **87**
7.1 The role of the symbolic layer . . . 87
7.2 Coupling as an operational lever . . . 88
7.3 The separability boundary . . . 89
7.4 Explainability: faithful versus useful . . . 90
7.5 The price of the symbolic layer . . . 91
7.6 Generalisation and portability . . . 92
7.7 Limitations . . . 93
7.8 Future work . . . 93

8 Conclusion **95**

Bibliography **97**

A Glossary **103**

B Implementation details **105**
B.1 Packet-level feature set . . . 105
B.2 Symbolic-rule mining algorithm . . . 106

C Extended results **109**
C.1 Binary attack-specific flow results . . . 109
C.2 CIC-IDS2017 original flow-level results . . . 109
C.3 Standardised NetFlow-v3 CSE-CIC-IDS2018 flow-level results . . . 109
C.4 Standardised NetFlow-v3 CSE-CIC-IDS2018 packet-level results . . . 110
C.5 Standardised NetFlow-v3 CSE-CIC-IDS2018 hybrid-level results . . . 111
C.6 UNSW-NB15 original flow-level results . . . 112
C.7 Standardised UNSW-NB15 NetFlow-v3 results . . . 114
C.8 Hard versus soft thresholds . . . 116
C.9 Theoretical complexity analysis . . . 118

List of Figures

4.1 System overview of the proposed neuro-symbolic NIDS. 30

5.1 Arithmetic circuit for the worked DoS example. 51

6.1 Binary attack-versus-benign flow results at $k = 50$ on CIC-IDS2017 (C:) and UNSW-NB15 (U:). Shaded families have no rule and fall back to the neural path. 59
6.2 Tiered versus anchored representation. Left: per-family macro-F1 difference at $k = 50$, Right: the UNSW reconnaissance sweep. 59
6.3 Original CIC-IDS2017 flow multiclass results across training size: macro-F1 (left) and attack false-positive heatmap (right). 60
6.4 Original CIC-IDS2017 flow multiclass across training size: detection-rate (left) and exact-attack (right) bubble grids. 61
6.5 Standardised NF-v3 packet level: macro-F1 by training size (left) and attack false-positive heatmap (right). 63
6.6 Standardised NF-v3 flow level: macro-F1 by training size (left) and attack false-positive heatmap (right). 65
6.7 Standardised NF-v3 hybrid level: macro-F1 by training size (left) and attack false-positive heatmap (right). 65
6.8 Portability for the enterprise-to-IoT pair, NF-CSE-CIC-IDS2018-v3 $\rightarrow$ NF-ToN-IoT-v3, binary classification on the held-out target over five replicates. . . 69
6.9 Robustness under Gaussian noise on CIC-IDS2017 flow at $k = 500$: macro-F1 versus per-feature standard deviation (left) and attack false-positive rate heatmap (right). 70
6.10 Clean macro-F1 against noise-floor macro-F1 at $\sigma = 2.0$. 71
6.11 Macro-F1 under constrained adversarial evasion on CIC-IDS2017 flow at $k = 500$, as the perturbation budget ε grows. Only attack samples are perturbed, targeted toward benign. 72
6.12 Share of attack alerts decided by an anchor clause, a neural-gated rescue clause, or the neural path alone, for three couplings. 75
6.13 Proof-path composition of attack alerts versus training size k, guarded coupling on CIC-IDS2017 flow. 75

6.14 Agreement of the explanation across five training replicates on CIC-IDS2017 flow, for the symbolic winning clause and the Integrated Gradients and SHAP attributions on the neural baseline. 78
6.15 Per-class agreement of the symbolic winning clause across replicates, with test-sample count per class. 79
6.16 Two-stage hyperparameter selection on the validation set. Left: stage-one screen of batch size and learning rates at five epochs, placed by macro-F1 and attack false-positive rate and coloured by batch size (selected configuration starred). Right: stage-two epoch sweep over five replicates, selection score per budget (selected ten epochs in red) with detection rate. 81
6.17 The three costs of the symbolic layer, each on a different axis. Left: inference throughput versus neural-leaf count. Centre: compiled-circuit size per rung. Right: training time per epoch versus rule-clause count. 83
6.18 Inference throughput of the MLP baseline against the neuro-symbolic ladder (L0–L5) on CIC-IDS2017 flow at $k = 500$. 83

C.1 UNSW-NB15 flow multiclass results for the five couplings and the neural baseline, which share the same low macro-F1 band because the attack families overlap in flow feature space. 114
C.2 UNSW-NB15 flow: detection-rate (left) and exact-attack (right) bubble grids over the five couplings and the baselines. 116
C.3 Product-of-experts coupling against the neural baseline and the structural couplings on the original UNSW-NB15 flow data. The product-of-experts is the only variant that lifts macro-F1 above the neural baseline, by learning from the co-occurrence pattern of fired rules rather than gating on a single rule per class. 116
C.4 Standardised UNSW-NB15 NetFlow-v3 packet view: macro-F1 by training set size (left) and attack false-positive heatmap (right), neural baseline against the pipeline neuro-symbolic model. The rules edge the baseline at the smaller budgets but the macro-F1 stays low, because the rules separate attack from benign without separating the families. 117

Chapter 1

Introduction

Modern society's growing dependence on digital systems has widened its exposure to cyber-attacks, and attackers continue to develop new techniques faster than defenders can respond. These threats range from Denial-of-Service (DoS) attacks to highly targeted intrusions, with ransomware now a dominant threat and phishing a common entry point [14, 46]. The growing complexity of modern networks and the rise of AI-assisted attacks further enlarge the attack surface [14]. The cost is substantial: the average breach costs organisations around $4.44 million and takes 241 days to identify and contain, with breaches that run beyond 200 days costing roughly $1 million more [23]. Detecting an attack early is therefore critical, giving defenders the chance to respond before significant damage occurs.

The December 2019 ransomware attack on Maastricht University illustrates this [16]. Attackers gained an initial foothold through two phishing emails and, over the following weeks, moved through the university's network gathering credentials before deploying Clop ransomware on 267 servers and bringing the institution to a standstill, ultimately forcing a ransom of roughly EUR 200,000.

Detecting such attacks reliably places several demands on a detector: it must operate in real time, ideally at line rate [2]; raise few false alarms, since excessive alerts overwhelm analysts and erode trust [45]; learn effectively from the limited labelled attack data a network can provide; generalise across changing network conditions; and explain its decisions so that an analyst can act on them [28, 21]. These tasks fall to Network Intrusion Detection Systems (NIDS), which monitor traffic for malicious activity using signature-based, anomaly-based, and machine-learning techniques.

Most deployed NIDS are signature-based, such as Snort and Suricata, matching traffic against a database of known attack signatures [5]. They run at line rate and produce interpretable, low-false-positive alerts on known threats, but cannot detect novel or zero-day attacks, which bypass their signatures [26, 17].

Data-driven detectors instead learn normal and malicious traffic patterns, allowing them to flag previously unseen behaviour [1, 47]. However, they require large amounts of representative labelled data [4, 43], are prone to high false-positive rates, generalise poorly across networks and datasets [40], and are fragile under adversarial perturbation, where small input changes let malicious traffic blend in with the learned benign traffic distribution [3]. They are also largely black box, offering little insight into their decisions and thereby limiting

trust in operational settings [28, 21].

Neuro-Symbolic AI (NeSy) offers a way to address these limitations by combining neural learning with symbolic reasoning [7, 25]: the neural component learns patterns from data while the symbolic component contributes structured domain knowledge and produces explanations, improving data efficiency, robustness, and interpretability [20, 18]. In this thesis we investigate NeSy for intrusion detection in local networks using DeepProbLog, a neural probabilistic logic programming framework [31]. The proposed system couples a neural classifier with automatically mined symbolic rules over attack patterns inside a single end-to-end differentiable model. This allows it to learn from the small amount of labelled data available on a given network and to mine new rules directly from traffic rather than rely solely on expert knowledge. The system treats the coupling between the neural and symbolic components as a design surface that can be configured to trade detection performance, robustness, and explainability against one another.

Whereas prior neuro-symbolic NIDS apply symbolic knowledge at a single stage of the pipeline — typically to enrich neural features during training or to generate post-hoc explanations [8, 24] — our approach integrates symbolic reasoning across the entire pipeline: the same mined rules guide learning, decide at inference, and produce the explanation, so interpretability is faithful by construction rather than reconstructed post hoc. To the best of our knowledge, this is the first neuro-symbolic NIDS to integrate symbolic reasoning at every stage in this way.

We evaluate the proposed system on the public UNSW-NB15 and CIC-IDS2017 datasets, together with standardised NetFlow-v3 versions of CSE-CIC-IDS2018, UNSW-NB15, and ToN-IoT, across three views of network traffic: packet-based, flow-based, and hybrid. Across these settings we assess detection and false-positive performance, data efficiency under low-data training, generalisation across networks and datasets, robustness to data drift and adversarial perturbation, and the interpretability of the symbolic explanations, comparing throughout against neural, symbolic-only, and state-of-the-art NIDS baselines.

This leads to the following research questions:

- **RQ1.** How does coupling mined symbolic rules to a neural classifier affect **detection performance** and **data efficiency** relative to purely neural, symbolic-only, and signature-based approaches?
  - **RQ1.1:** How does it compare in macro-F1, attack false-positive rate, and attack-classification accuracy across the packet, flow, and hybrid views?
  - **RQ1.2:** How does detection performance scale with the amount of labelled training data, and how much labelled data does the symbolic coupling save?
- **RQ2.** To what extent does symbolic knowledge improve generalisation and robustness, and how does the neural–symbolic coupling mediate this?
  - **RQ2.1:** Does neuro-symbolic integration allow transfer to a different network topology or dataset, and under what conditions?
  - **RQ2.2:** How do the different couplings trade clean-traffic precision against degradation under noise and adversarial evasion?

- **RQ3.** Does the symbolic component yield meaningful explanations that are interpretable to a human analyst?
  - **RQ3.1:** Are the explanations faithful, stable, and human-interpretable, and how do they compare to post-hoc explanations of neural-only models?

The remainder of this thesis is organised as follows. Chapter 2 reviews related work on intrusion detection and neuro-symbolic AI, and Chapter 3 provides the necessary background. Chapter 4 presents the methodology of the proposed system and Chapter 5 its implementation. Chapter 6 reports the experimental setup and results, which are discussed in Chapter 7. Finally, Chapter 8 concludes the thesis and outlines directions for future work.

Chapter 2

Related Work

This chapter reviews related work on signature-based and anomaly-based Network Intrusion Detection Systems (NIDS), the role that explainable Artificial Intelligence (xAI) plays in machine-learning- and deep-learning-based NIDS, and the state of Neuro-Symbolic (NeSy) Artificial Intelligence in network security.

2.1 Network Intrusion Detection Systems

Network Intrusion Detection Systems occupy a central role in the defence of modern networked infrastructure. By monitoring traffic at the network perimeter and on individual hosts, an NIDS aims to identify malicious activity and alert operators before damage can propagate. At the broadest level, detection strategies fall into two families: signature-based systems, which match observed events against a database of known attack patterns, and anomaly-based systems, which flag deviations from a learned baseline of normal behaviour.

2.1.1 Signature-Based Network Intrusion Detection Systems

Despite decades of research into anomaly-based and machine learning-driven alternatives, the signature-based paradigm has remained dominant in operational deployments [43]. Understanding why this is the case and why it is ultimately insufficient is essential context for the present work.

A signature-based IDS, sometimes called a misuse detection system, operates by maintaining a curated collection of rules, each encoding the characteristics of a known attack. When a network event matches one of these rules, an alert is raised. Systems such as Snort and Suricata represent the canonical open-source instantiations of this approach and remain among the most widely deployed intrusion detection tools in production environments today [5]. Their appeal is straightforward: when a signature is accurate and the attack it describes is encountered, detection is precise, reliable, and fast. False positive rates are comparatively low when rule sets are well-tuned, and the behaviour of the system is transparent [26].

However, these properties come with a fundamental and well-understood structural limitation. Signature-based systems can only detect what they already know. As Sommer and

Paxson observe in their seminal analysis of machine learning for network intrusion detection, operational NIDS in real-world settings are almost exclusively misuse detectors precisely because anomaly-based alternatives have consistently struggled to meet the stringent operational requirements of production environments [43]. Nevertheless, the dominance of signatures does not resolve the core problem: any attack for which no rule has yet been written will pass through undetected. Zero-day exploits, novel malware variants, and attacks that deliberately modify their behaviour to avoid matching known patterns all fall outside the reach of a signature-based system, regardless of how comprehensive its rule database may be. A new signature can only be authored after an attack has been observed, analysed, and encoded, introducing an unavoidable temporal gap during which the system is blind.

The severity of this lag has been empirically documented. Asad et al. (2024) [5] conduct a retrospective analysis of Snort and Suricata using rule sets collected over a four-year period, examining how these systems perform when rules from one point in time are applied to traffic from another. Their findings confirm that there is a persistent and measurable disparity between the time an attack occurs and the time a matching rule is available, and that the configuration differences between the two tools, which use partially different rule sets and matching logic, create meaningful variation in what each system can and cannot detect. The implication is that even among the most mature and widely maintained signature-based systems, coverage is incomplete and temporally fragile.

Beyond the zero-day problem, signature-based systems face a set of operational pressures that have grown more acute as network environments have evolved. The proliferation of encrypted traffic represents a particularly significant challenge: because signature matching traditionally operates on packet payloads, encryption renders the content of traffic opaque to inspection, and many attack indicators that would otherwise trigger a rule become inaccessible without decryption infrastructure that is costly to deploy and raises its own privacy concerns [43]. Attackers have also developed a wide repertoire of evasion techniques that are specifically designed to prevent traffic from matching known signatures even when the underlying attack is functionally identical to a known threat [26]. These techniques exploit the fact that signatures encode a specific syntactic pattern rather than a semantic understanding of malicious intent, and a sufficiently minor modification to the attack's surface representation is often enough to bypass detection entirely.

The limitations of signature-based detection are not a recent discovery, nor have they gone unaddressed in the research community. Apruzzese et al. (2023) [4] observe that a striking gap exists between the academic literature on ML-based NIDS, which regularly demonstrates strong performance on benchmark datasets, and actual operational deployments, which continue to rely primarily on signature systems. Their analysis suggests that this gap is not simply a matter of conservatism or inertia, but reflects genuine difficulties in transferring research results to production: class imbalance in real traffic, the absence of ground truth labels for the majority of events, hardware constraints, and the open-world nature of operational environments all combine to make real-world deployment substantially harder than benchmark evaluation suggests. In this sense, the persistence of signature-based systems reflects not just their strengths but the unresolved challenges facing their would-be successors.

It is against this backdrop that the present work is situated. While network intrusion detection systems offer broad visibility into traffic flows and packets traversing the network, the fundamental limitations of signature-based detection remain: such systems still require prior knowledge of attack patterns, still degrade in the face of novel threats and concept drift, and still produce the alert volumes that challenge operational response. The need for a detection approach that can reason under uncertainty, generalise beyond known patterns, and integrate structured domain knowledge with learned representations without sacrificing interpretability defines precisely the research gap that motivates the exploration of neuro-symbolic AI in the sections that follow.

2.1.2 Anomaly-Based Network Intrusion Detection Systems

Anomaly-based Network Intrusion Detection Systems (NIDS) rely on machine learning (ML) or deep learning (DL) models to identify malicious network activity by learning patterns from network traffic data. These systems usually operate on features extracted from network flows or packets, which may be manually engineered or automatically learned from raw traffic data. Anomaly-based detection approaches are typically implemented using supervised, unsupervised, or semi-supervised learning techniques, depending on whether labeled attack data is available during training [32].

The performance of anomaly-based NIDS depends primarily on two factors: the architecture and learning methodology of the detection model, and the choice of appropriate datasets used for training and evaluation [36]. That performance is commonly evaluated using metrics such as accuracy, precision, recall, F1-score, and false positive rate. Among these metrics, the false positive rate is particularly critical in operational environments, as excessive false alarms can overwhelm security analysts [1].

Architectures and Learning Approaches

Ahmad et al. [1] highlight that earlier anomaly-based NIDS approaches primarily relied on traditional machine learning algorithms such as Support Vector Machines (SVM), Decision Trees, Random Forests, k-Nearest Neighbors (k-NN), and Naïve Bayes classifiers. However, deep learning techniques have gained significant popularity in recent years. According to the systematic study conducted by Ahmad et al., approximately 60% of the surveyed papers published between 2017 and 2022 relied exclusively on deep learning models, while around 20% employed hybrid architectures combining machine learning and deep learning techniques. Furthermore, deep learning models can be trained using unlabeled data in certain settings, although they typically require significantly larger training datasets compared to traditional ML-based approaches.

The increasing adoption of DL-based methods can be attributed to their ability to model complex network traffic patterns and automatically extract features from raw data, reducing the need for manual feature engineering traditionally required in machine learning-based NIDS. Common deep learning architectures used in NIDS include Convolutional Neural Networks (CNN), Recurrent Neural Networks (RNN), Long Short-Term Memory networks

(LSTM), and autoencoder-based models, which are capable of learning complex temporal and spatial patterns in network traffic [1] [32].

Architectures used in anomaly-based NIDS differ mainly in how they represent network traffic data and how they combine multiple feature sources to improve detection accuracy. Most approaches rely on either flow-based features extracted from network flows, packet-level features derived from raw packets, or hybrid representations combining both. Flow-based features are widely used due to their scalability and lower computational requirements, while packet-level analysis can capture finer-grained information embedded in payload data, leading to fewer false positives. Recent research increasingly explores hybrid approaches that combine both representations to leverage the advantages of both traffic perspectives [2, 15]. Thus, recent architectures can be grouped into models relying on a single traffic representation and models combining flow-level and packet-level information.

Single traffic models

Many modern anomaly-based NIDS rely on deep learning models trained on a single representation of network traffic, typically statistical flow features or packet-level data. These approaches aim to automatically learn complex patterns in network traffic without requiring manual feature engineering.

Lan et al. (2022) proposed MEMBER, a multi-task deep learning architecture combining a CNN, a memory-enhanced autoencoder, and a prototype-based distance learning module to improve generalization and robustness in intrusion detection. By jointly learning discriminative traffic features, normal traffic patterns, and class-separating prototypes, the model extracts complementary representations and improves performance on imbalanced datasets[30].

Beyond improving classification performance, recent work has also started to address deployment-oriented challenges such as real-time inference and data drift. Zha et al. (2025) proposed A-NIDS, an adaptive architecture combining a shallow fully connected detector with clustering-based drift detection and a stacked CTGAN generation module to mitigate catastrophic forgetting during model updates. This design allows the system to adapt to changing traffic distributions without continuously storing historical data, while maintaining real-time performance, illustrating a growing interest in adaptive and deployment-aware NIDS architectures [49].

Multi-traffic hybrid flow-packet NIDS

Qiu et al. proposed a hybrid intrusion detection framework that integrates packet-level and flow-level models using Dempster–Shafer evidence theory (DST). In their system, packet-based models analyze raw packet content to capture spatial relationships within packets, while flow-based models extract statistical features such as packet counts, timing char-

acteristics, and flow duration. The predictions produced by both models are then fused using DST to produce a final decision. This approach enables the system to exploit the fine-grained information contained in packet payloads while maintaining the scalability of flow-based analysis. Experimental results demonstrate that combining both representations significantly improves detection accuracy and detection speed compared to single-representation systems [39].

More recent research has further explored unified architectures that dynamically combine flow-level and packet-level analysis. Sudyana et al. introduced a two-stage hybrid intrusion detection framework that first performs detection using flow-based features and only invokes packet-level analysis when the flow-based prediction confidence is low. This architecture reduces computational cost while maintaining high detection performance by using flow-based detection as the first defense layer and applying packet-based inspection selectively for uncertain traffic flows. Their results show that this strategy significantly reduces computational overhead while maintaining strong detection accuracy, demonstrating the practical benefits of combining both representations in a unified system [44].

Hybrid representations have also been explored through multi-source feature learning architectures. In many modern NIDS, different feature types capture complementary aspects of network traffic behavior. For example, statistical flow features capture temporal characteristics of network activity, while packet-level features capture spatial patterns within packet payloads. Integrating these heterogeneous features allows detection models to capture both macro-level traffic behavior and micro-level packet content characteristics, leading to improved detection performance and robustness against sophisticated attacks.

Beyond architectural design, Apruzzese et al. emphasize that many machine learning-based NIDS are evaluated primarily in laboratory settings and overlook deployment factors such as computational cost, hardware constraints, dataset bias, and operational requirements. Their work highlights that practical NIDS must balance detection performance with efficiency and adaptability, which further motivates the exploration of scalable and transferable architectures [4].

Network Intrusion Datasets

Designing a high-performance anomaly-based NIDS requires not only a well-designed detection architecture but also appropriate datasets for training and evaluation. Over the years, numerous datasets have been developed to support the training and benchmarking of intrusion detection systems. Many studies still rely on legacy datasets that have long been standardized within the NIDS research community. Approximately 60% of the studies analyzed in [1] evaluated their models using the KDD Cup'99 dataset or its improved version, NSL-KDD. These are two of the oldest and most widely used datasets in the history of NIDS, featuring DoS, Probe, R2L, and U2R attacks. However, the survey highlights concerns regarding the continued reliance on these outdated datasets, which contain deprecated attack scenarios and unrealistic traffic distributions. More recent studies increasingly rely on modern datasets such as CIC-IDS2017 and UNSW-NB15, which feature a wider variety of contemporary attack types and aim to provide more realistic network traffic scenarios [1, 36].

Dataset Challenges

The broad range of datasets reviewed by Mondragon et al. (2025) highlights the absence of a universally accepted benchmark that could serve as a "gold standard" for evaluating ML/DL-based NIDS models [36]. Due to the diversity of real-world network environments and traffic patterns, it is difficult to design a single dataset capable of representing all possible deployment scenarios. Careful selection of datasets is required when evaluating a NIDS system. Furthermore, current datasets are often imbalanced, which affects the detection accuracy of models when deployed in real-world environments.

Class imbalance is a common challenge in intrusion detection datasets, where attack traffic often represents only a small fraction of the overall network traffic. This imbalance can bias machine learning models toward the majority class, leading to poor detection performance for rare attack types. Typical strategies for addressing class imbalance include resampling methods such as SMOTE, RandomOverSampler, and ADASYN, as well as cost-sensitive learning and tree- or ensemble-based methods designed to improve minority-class detection performance [1][30].

Overall, the literature shows that the effectiveness of anomaly-based NIDS depends not only on the learning architecture adopted, but also on the quality and representativeness of the datasets used for training and evaluation. Recent research highlights a clear evolution from traditional machine learning models toward deep learning architectures capable of automatically extracting complex spatial and temporal traffic patterns. At the same time, studies distinguish between systems relying on a single traffic representation and hybrid architectures combining flow-level and packet-level information to balance scalability with fine-grained detection capability. However, these architectural advances remain strongly constrained by the realism of data, class imbalance, and the difficulty of transferring learned models across different network environments and traffic topologies. Consequently, the design of robust anomaly-based NIDS requires balancing model complexity, feature representation, computational efficiency, and dataset suitability for the intended deployment scenario.

xAI in Network Intrusion Detection Systems

Machine learning (ML) and deep learning (DL) models have significantly improved the performance of anomaly-based Network Intrusion Detection Systems (NIDS) by enabling the detection of complex attack patterns in network traffic. However, Kumar et al. (2025) [28] highlight that many high-performance models operate as black-box systems, meaning that their internal decision processes remain difficult for human analysts to interpret. In addition, commercial NIDS solutions often restrict access to the underlying models and their parameters to preserve the confidentiality of the system's implementation. The black-box nature of ML and DL models constitutes one of the major challenges associated with anomaly-based NIDS. Explaining the decisions of such models by inspecting their internal parameters and architecture is inherently difficult, making it challenging to determine the causes of false alarms and misclassifications. Consequently, diagnosing model behavior and

optimizing ML/DL-based NIDS for specific deployment environments becomes difficult. This lack of transparency can reduce the trust of Security Operations Center (SOC) analysts in ML/DL-based NIDS, as the systems are often unable to justify the reasoning behind their predictions.

Moreover, deep learning models may rely on hidden correlations or artifacts present in the training data rather than meaningful network traffic characteristics, potentially leading to misleading detection performance. Therefore, explanation techniques are required to uncover such hidden patterns and allow security practitioners to better assess the reliability of ML/DL-based NIDS.

To address these challenges, researchers have explored the use of Explainable Artificial Intelligence (XAI) techniques. Hariharan et al. (2022) [21] mention that XAI methods aim to provide interpretable insights into the decision-making processes of ML/DL-based NIDS by identifying which input features influence predictions and how these features contribute to classification outcomes. These explanations allow analysts to better understand model behavior, detect misleading correlations in training data, and improve trust in automated cybersecurity systems.

Moreover, Kumar et al. (2025) [29] also suggest that explainability should be applied not only as a post-hoc analysis step, but also as an influence on the design of intrusion detection systems themselves. Systems designed with explainability in mind may produce more interpretable decision boundaries while maintaining competitive detection performance.

Kumar et al. (2025) [28] classify explainability methods into black-box and white-box explanations. Black-box explanation techniques assume a black-box context – zero knowledge about the inner workings of the underlying ML model and its associated parameters – and are invaluable when the ML and DL models are not available. White-box techniques work under the premise that we have full knowledge of all the parameters of the ML model being tested, bypassing approximation and directly computing explanations from the prediction functions.

Hariharan et al. (2022) [21] suggest that these explanations can also be categorized into global explanations, which describe the overall behavior of a model, and local explanations, which focus on explaining individual predictions. Global explanations provide insights into the general decision logic of a model, while local explanations help analysts understand why a specific network flow or packet was classified as malicious.

Several XAI methods have been applied to intrusion detection systems, including Permutation Importance, SHAP, LIME, and Contextual Importance and Utility (CIU). These black-box approaches provide different perspectives on model interpretability, ranging from feature importance rankings to instance-specific explanations [21].

Permutation Importance (PI) is a model-agnostic feature attribution method used to determine the importance of input features in a machine learning model. The method works by randomly shuffling the values of a specific feature and measuring the resulting decrease in model performance. If permuting the feature significantly reduces prediction accuracy, the feature is considered important for the model's decision-making process. Permutation

Importance provides a global interpretation of the model by ranking features according to their influence on prediction accuracy. In intrusion detection systems, this technique can be used to identify which network traffic attributes contribute most strongly to attack detection. [21]

SHapley Additive exPlanations (SHAP) is one of the most widely used XAI techniques for interpreting machine learning models. The method is based on Shapley values from cooperative game theory, where each feature is treated as a player contributing to the final prediction. The SHAP value represents the average marginal contribution of a feature across all possible combinations of input features [21]. One advantage of SHAP is that it provides both global and local explanations. At a global level, SHAP can identify the most important features influencing model predictions across the dataset. At a local level, it can explain how specific feature values contribute to the classification of an individual network traffic instance [28]. In intrusion detection systems, SHAP values can reveal which network traffic attributes trigger intrusion alerts. For example, SHAP analysis can identify the flow-level or payload-level features that contribute most strongly to the classification of network attacks [27].

Local Interpretable Model-Agnostic Explanations (LIME) is a method designed to explain individual predictions made by complex machine learning models. Instead of attempting to interpret the entire model, LIME focuses on approximating the model locally around a specific data instance. The method generates perturbed samples around the instance being analyzed and observes how the model's predictions change. These samples are then used to train a simple interpretable surrogate model that approximates the behaviour of the original complex model in that local region of the feature space [21]. The surrogate model highlights the features that most strongly influence the prediction for the analyzed instance. In intrusion detection systems, this allows analysts to understand why a specific network flow or packet was classified as malicious [28].

Contextual Importance and Utility (CIU) is another explanation method designed to interpret predictions in the context of a specific instance. The framework computes the Contextual Importance (CI) to measure how influential a feature is in the current prediction context, and the Contextual Utility (CU) that indicates whether the current value of the feature positively or negatively influences the predicted outcome. This context-dependent explanation mechanism allows analysts to better understand the role of specific features in individual predictions, which is particularly useful in cybersecurity investigations where attack patterns may depend on complex feature interactions. [21]

TRUSTEE is a global explainability method designed specifically for complex machine learning models. The technique generates interpretable decision trees that approximate the behavior of a black-box model. These decision trees serve as surrogate models that provide a simplified representation of the original model's decision logic [28]. Compared with local explanation techniques such as SHAP and LIME, TRUSTEE focuses on explaining the global behaviour of the model. This approach can reveal how the model makes decisions across the entire dataset rather than for individual instances [29]. However, studies have shown that different explainability methods may produce conflicting explanations. For example, explanations generated by SHAP and TRUSTEE may disagree on which features are most important for model predictions. This highlights the importance of evaluating multi-

ple explanation methods when interpreting machine learning models in intrusion detection systems [28].

Application of xAI in state-of-the-art NIDS

Kumar et al. (2025) analyze the interpretability of several deep learning-based NIDS models, including Kitsune, HorusEye, ENIDrift, and HELAD. The authors apply XAI methods such as SHAP and TRUSTEE to analyze how these models detect malicious traffic. They also propose an evaluation framework that compares explanations based on agreement, stability, and adversarial robustness. Their results show that different XAI techniques may produce conflicting explanations and that some models are inherently easier to interpret than others [28].

Hariharan et al. (2022) investigate several explainability methods applied to tree-based machine learning models such as Random Forest, XGBoost, and LightGBM. The study evaluates global explanation methods such as Permutation Importance and SHAP, as well as local methods such as LIME and CIU. The authors compare these techniques using evaluation metrics such as accuracy, stability, and consistency to determine the reliability of the generated explanations [21].

In a different study, Kumar et al. (2025) propose a deep learning-based intrusion detection system designed specifically to improve explainability. Unlike many existing NIDS approaches that apply XAI techniques only after model training, ExpIDS incorporates explainability as a design objective. The system is optimized to produce decision tree explanations with high fidelity using the TRUSTEE framework while remaining adaptable to network traffic distribution changes. Experimental results show that ExpIDS achieves high explanation fidelity while maintaining competitive detection performance compared to other state-of-the-art NIDS models [29].

Explainability techniques have also been applied to analyze feature contributions in hybrid intrusion detection systems. Kiflay et al. (2024) propose a multimodal NIDS that combines flow-based features and packet payload information using two random forest classifiers whose predictions are fused using a soft voting scheme. SHAP values are used to analyze the contributions of both feature types to the final model output. The analysis shows that payload features generally contribute more strongly to attack detection, while flow-based features remain necessary for detecting attacks with limited payload data [27].

Despite the progress made in integrating explainability techniques into machine learning-based intrusion detection systems, several limitations remain. Most existing approaches rely on post-hoc explanation methods, such as SHAP, LIME, or permutation importance, which are applied after the prediction has already been produced by the model. While these techniques can provide useful insights into the factors influencing a particular decision, they do not directly contribute to the decision-making process itself. Instead, they function primarily as diagnostic tools that help analysts interpret the behaviour of an already trained model rather than guiding the classification process [21] [28].

Another limitation lies in the dependence of explanation methods on the underlying machine learning model. Although many XAI techniques are described as model-agnostic, their explanations are ultimately constrained by the behaviour and biases of the models they attempt to interpret. As a result, explanations may vary significantly depending on the architecture used, and different explanation techniques may even produce conflicting interpretations for the same prediction. This inconsistency has been observed in studies comparing multiple explainability methods for intrusion detection systems [28].

Finally, most current XAI approaches are designed for interpretability rather than reasoning, meaning that they explain model predictions without enforcing logical constraints or incorporating domain knowledge about network behaviour. This limitation suggests that alternative approaches capable of combining statistical learning with symbolic reasoning may provide a more integrated solution for achieving both high detection performance and interpretable decision-making in intrusion detection systems.

2.2 Neuro-Symbolic Artificial Intelligence

Tasks in Artificial Intelligence can broadly be divided into tasks requiring high-level reasoning provided by symbolic AI and tasks requiring low-level perception, which can be achieved through connectionist AI such as machine- and deep-learning approaches. Symbolic AI, being rooted in traditional logic, knowledge representation, and probability theory, excels in domains that demand explicit, rule-governed reasoning because it can perform deductive, inductive, and abductive inference with precision. Its outputs are inherently interpretable since logical rules and symbols are readable and probability theory can easily be interpreted by humans. However, symbolic systems usually struggle when confronted with incomplete, noisy, or complex, unstructured data, as their effectiveness relies on the completeness and correctness of the underlying knowledge base. Connectionist AI takes the opposite approach by relying on deep learning approaches to learn rich representations directly from raw data, demonstrating noticeable success in perception-heavy tasks such as interpreting images, sounds, text, and speech. Nevertheless, deep learning models remain notoriously data-hungry, opaque in their decision-making due to their black-box nature, and incapable of the structured, symbolic reasoning that humans are capable of performing naturally.

As highlighted in Bhuyan et al. (2024) [7], the limitations of these two paradigms are in many ways complementary: connectionist AI offers high efficiency and strong generalization over perceptual data but suffers from limited explainability, while symbolic AI provides interpretability and logical rigor but is computationally rigid and poorly suited to learning from unstructured observations. This complementarity provided the intellectual impetus for neuro-symbolic AI (NeSy AI), a field that has been taking shape since the early 2000s and has gained substantial momentum over the last decade. The main ambition is to build systems with more human-like reasoning capabilities by combining the pattern-learning strengths of neural networks with the structured knowledge representation and logical inference of symbolic methods. As Manhaeve et al. (2021) [31] argue, truly intelligent systems must be capable of both low-level perception and high-level reasoning,

and bridging this divide remains one of the central open challenges of contemporary AI research.

A useful lens for understanding what neuro-symbolic systems are trying to achieve comes from examining four fundamental properties that distinguish them from their constituent paradigms: representation, learning, reasoning, and decision-making [7]. In terms of representation, neuro-symbolic systems must combine two fundamentally different ways of encoding knowledge. Symbolic AI represents data through discrete symbols that correspond directly to human-interpretable concepts, while connectionist AI relies on distributed representations, where concepts are encoded as patterns across continuous, high-dimensional vectors. NeSy approaches seek to leverage both, using distributed representations where flexibility and low-level perception are needed, and localist representations where interpretability and logical manipulation are required.

During the training phase, symbolic AI learns through logical deduction and inductive generalization from structured examples, often requiring significant manual knowledge engineering by building symbolic rules and knowledge bases. Neural networks, by contrast, learn from large volumes of raw data using gradient-based optimization. Neuro-symbolic systems aim to transcend both limitations: by grounding neural network outputs in symbolic representations, they can learn from fewer examples through the injection of prior symbolic knowledge, while remaining adaptable to raw perceptual inputs. For reasoning, neuro-symbolic AI blends the capabilities of symbolic AI for high-level reasoning with the approximate, data-driven inference of neural networks. Classical symbolic methods perform well in structured environments where the knowledge base is provided, but struggle with uncertainty. Neural networks handle uncertainty naturally but cannot perform explicit, rule-governed inference. The NeSy synthesis therefore enables reasoning that is simultaneously robust to noise and grounded in logical constraints.

Finally, in terms of decision-making, Bhuyan et al. [7] argue that the symbolic and neural components of a NeSy system mirror Kahneman's dual-process theory, with the neural components enabling rapid, intuitive pattern recognition analogous to System 1 thinking while the symbolic components enable deliberate, logic-driven analysis similar to System 2. Neuro-symbolic systems are designed to integrate both modes, allowing fast perceptual inference to feed into careful, structured reasoning.

A Taxonomy of Neuro-Symbolic Integration

Not all neuro-symbolic systems integrate their neural and symbolic components in the same way. Drawing on Kautz's influential taxonomy [25], Bhuyan et al. (2024) [7] identify six principal types of integration, each representing a different architectural philosophy and degree of coupling between the two paradigms.

Type 1 (Symbolic Neuro-Symbolic) systems are primarily neural in character, employing neural networks for their representational power while using symbolic reasoning to structure the knowledge space and interpret outputs. Neural vector embeddings transform input symbols into rich continuous spaces, capturing semantic relationships, while a symbolic layer imposes logical constraints and ensures interpretable outputs. Word embedding models such as word2vec [33] and GloVe [38] are representative examples.

Type 2 (Symbolic [Neuro]) systems are predominantly symbolic, treating neural networks as subroutines within a broader symbolic problem solver. The symbolic system retains overarching control, delegating specific sub-tasks such as pattern recognition or heuristic evaluation to neural modules. AlphaGo [42], which uses a neural network for heuristic guidance within a Monte Carlo Tree Search framework, is perhaps the most widely cited example of this category.

Type 3 (Neuro — Symbolic) represents a more cooperative architecture, where neural and symbolic components interact bidirectionally to improve each other's performance. Rather than one component subsuming the other, the two layers act as genuine peers: symbolic reasoning structures the inference process and generates gradient signals, while neural networks supply probabilistic groundings for logical atoms. Systems like the differentiable theorem provers of Minervini et al. [34] and DeepProbLog [31] fall squarely within this category.

Type 4 (Neuro-Symbolic $\rightarrow$ Neuro) systems incorporate symbolic knowledge directly into the architecture or training process of a neural network. Symbolic information such as logical constraints, domain ontologies, or structured relational data shapes what the network can learn, but inference at test time is performed by the neural component alone. Graph Neural Networks that incorporate external knowledge bases are a representative example.

Type 5 (NeuroSymbolic) systems encode symbolic knowledge as soft constraints within the neural network's loss function. Logic Tensor Networks (LTNs) [6] are the canonical example: they represent first-order logic formulas using fuzzy relations over real numbers, integrating them as differentiable regularization terms during training. The network internalizes symbolic knowledge through its weights, enabling approximate, gradient-compatible reasoning.

Type 6 (Neuro[Symbolic]) represents the most ambitious and as yet largely aspirational form of integration, in which a symbolic reasoning engine is embedded directly within a neural engine as a fully unified system. Kautz regards this as the most promising long-term direction for achieving human-level AI, capable of combining System 1 and System 2 cognition in a single architecture. No system currently fully realizes this vision, and it remains an open research frontier [7].

This taxonomy is not merely a conceptual exercise. It provides a practical framework for situating specific NeSy systems and for understanding what architectural trade-offs they entail. In the context of the present work, the most relevant category is Type 3, whose bidirectional coupling between neural perception and symbolic reasoning is precisely what makes systems like DeepProbLog well-suited to the demands of intrusion detection.

Deep Symbolic Learning: A Representative NeSy Approach

Before examining DeepProbLog in depth, it is instructive to briefly consider Deep Symbolic Learning (DSL), proposed by Daniele et al. (2023) [10], as it illustrates both the ambitions and the limitations of the broader NeSy landscape. DSL tackles what the authors call the Symbol Grounding Problem. This problem is the challenge of meaningfully associating abstract symbols with perceptual inputs. Most NeSy systems assume that the symbolic rules governing a task are provided a priori, using them to supervise the training of a neural

perception module. DSL departs from this assumption by simultaneously learning both the perception functions that map raw sensory inputs to discrete symbols and the symbolic rules that govern how those symbols combine, using supervision only on the end-to-end output of their composition.

While DSL is a compelling demonstration of joint symbol and rule discovery, it operates without explicit probabilistic modeling. Uncertainty in the perceptual process is handled implicitly through neural confidence values rather than through principled probabilistic semantics. For many real-world applications, and particularly for security-sensitive tasks like network intrusion detection where inputs are inherently noisy, evidence is partial, and anomaly assessment necessarily involves degrees of uncertainty, the ability to reason explicitly and rigorously over probabilistic beliefs is not a convenience but a requirement. This is precisely the gap that DeepProbLog addresses.

DeepProbLog: Neural Probabilistic Logic Programming

DeepProbLog is a neural probabilistic logic programming framework that unifies deep learning, probabilistic reasoning, and logical inference in a single, end-to-end trainable system. Manhaeve et al. (2021) [31] adopted a unique approach when introducing DeepProbLog: rather than building a new formalism from scratch, it extends ProbLog [12], an existing, well-understood probabilistic logic programming language with the minimal addition needed to incorporate neural networks. This extension takes the form of the neural predicate, a concept at the heart of the entire framework.

Grasping the fundamentals of how DeepProbLog functions requires a deeper understanding of ProbLog. Introduced by De Raedt et al. (2007) [12], ProbLog is a probabilistic extension of the paradigmatic logic programming language Prolog in which each atom in a logic program can be annotated with a probability. Formally, a probabilistic fact of the form $p :: f$ means that fact f is independently true with probability p. Given a set of probabilistic facts and a set of logical rules, ProbLog defines a distribution over possible worlds: each possible world is a consistent assignment of truth values to probabilistic facts, and the probability of a world is the product of the probabilities of its constituent facts. The success probability of a query is then the sum of the probabilities of all possible worlds in which the query is true, equivalent to performing weighted model counting over the propositional formula encoding the query.

ProbLog inference proceeds in four steps: grounding the logic program with respect to the query, rewriting it into a propositional formula, compiling that formula into an efficient arithmetic circuit (AC), and evaluating the AC to compute the query probability. The arithmetic circuit is a key structure because it encodes the logical dependencies between probabilistic facts in a form that is not only efficient to evaluate, but also differentiable – a property that becomes critical for learning.

The central innovation of DeepProbLog is the introduction of neural predicates, which replace fixed probabilistic facts with facts whose probabilities are computed dynamically by a neural network. A neural annotated disjunction (nAD) takes the form

$$nn(m, [X_1, ..., X_k], O, [y_1, ..., y_n]) :: r(X_1, ..., X_k, O)$$

where m is a neural network that defines a probability distribution over output classes $\{y_1, ..., y_n\}$ given inputs $\{X_1, ..., X_k\}$. Rather than asserting that a fact holds with a fixed probability, the system delegates the assessment of that probability to a neural network evaluated on the provided input. A convolutional network classifying an image, a recurrent network processing a sequence, or a feedforward network operating over tabular data can all serve as m in this expression. The logic program then reasons over these neural-derived probabilities to answer queries, combining perceptual uncertainty with structured logical inference. Adding this small semantic extension to ProbLog immediately enables arbitrary deep learning architectures to be incorporated into a probabilistic reasoning system.

Inference in DeepProbLog closely mirrors ProbLog inference. The key difference is that when the ground nADs are instantiated, their probability values are obtained by evaluating the neural network on the provided inputs rather than by reading off fixed parameters. Once instantiated, the resulting ground program is processed identically to a standard ProbLog program: it is compiled into an arithmetic circuit and evaluated to compute query success probabilities.

Learning in DeepProbLog is performed end-to-end, jointly optimizing both the neural network parameters and any learnable probabilistic parameters in the logic program. The mechanism exploits the differentiability of the arithmetic circuit: using the gradient semiring, an algebraic generalization of ProbLog inference, the system can simultaneously compute the success probability of a query and the gradient of that probability with respect to all learnable parameters. Gradients flow through the arithmetic circuit back to the outputs of the neural networks, at which point standard backpropagation propagates them through the network weights. This creates a single unified differentiable computation graph spanning both the logical and neural components, enabling standard gradient-based optimizers such as Adam to train the full system from end-to-end using only high-level supervision through labels on the logical queries, not on individual neural network outputs.

Manhaeve et al. (2021) [31] establish both theoretically and experimentally that DeepProbLog jointly supports four capabilities that no prior framework had realized in combination.

The first is the integration of symbolic and subsymbolic inference: the system operates simultaneously in the continuous space of neural network weights and the discrete space of logical atoms and rules, without collapsing either into the other. The second is program induction: DeepProbLog can fill in missing components of partially specified programs from input-output supervision on the complete program, generalizing to sequences longer than those seen during training in ways that specialized systems like differentiable Forth cannot. The third is probabilistic reasoning: uncertainty is a first-class citizen, and the system can explicitly model and learn the parameters of uncertain processes. In experiments with corrupted training labels, DeepProbLog with explicit noise modeling retains 73% accuracy even when 80% of labels are noisy, showing a level of robustness that a pure neural baseline cannot approach. The fourth is learning from indirect supervision: the neural components are never trained on direct perceptual labels; instead, they learn their representations as a consequence of satisfying the logical constraints of the program. In low-data regimes this

advantage is pronounced: with only 300 training examples, DeepProbLog achieves roughly three times the accuracy of a comparable CNN baseline. [31]

A further practical advantage is modularity and transferability: because neural predicates encode perceptual skills independently of the logical reasoning structure, a trained neural predicate can be reused in a logically related but different task without retraining, simply by embedding it within a new logic program.

Neuro-Symbolic AI in NIDS

The application of neuro-symbolic AI to network intrusion detection is a relatively nascent but rapidly growing research direction. While the previous sections have examined the theoretical foundations and architectural varieties of NeSy AI, this section surveys the existing body of work that has begun to operationalize these ideas in the cybersecurity domain with a particular focus on what has been attempted, what results have been achieved, and where meaningful gaps remain.

A broad framing of the challenge is offered by Grov et al. (2024) [18], who argue that a Security Operations Centre (SOC) represents one of the most natural application environments for neuro-symbolic AI. Their reasoning draws directly on the dual-process analogy discussed earlier: the monitoring phase of incident management, where ML models sift through vast streams of events to flag suspicious behaviour, corresponds to fast, System 1-style processing, while the analysis and response phases which require contextualising alerts, reasoning about attack sequences, and generating actionable decisions correspond to the deliberate, System 2-style reasoning that symbolic components provide. Neither pure neural nor pure symbolic approaches are sufficient in isolation; it is the combination that matches the operational reality of a SOC. From this perspective, Grov et al. identify a set of concrete NeSy use cases spanning the full MAPE-K lifecycle, from knowledge-aware detection to automated incident report generation. Crucially, they also conduct a proof-of-concept experiment using a Logic Tensor Network applied to a subset of the CICIDS2017 dataset to classify network flows as benign, brute force, or XSS traffic. By encoding a single domain-aware axiom stating that flows not communicating with a web server cannot constitute web attacks, the LTN-based classifier nearly doubled the precision of the baseline neural network for both XSS and brute force detection, while recall remained largely unchanged. The result, though deliberately simplified, illustrates the core NeSy proposition: even a small amount of symbolically encoded domain knowledge can produce meaningful, targeted improvements in a neural detector.

The same Logic Tensor Networks architecture is the foundation of the work by Bizzarri et al. (2024) [8], who propose a more explicitly formulated NSAI NIDS based on LTNs with a hybrid loss function. Their approach encapsulates a 1D Convolutional Neural Network within an LTN predicate that enforces a logical distinction between benign and non-benign traffic during training, adding a satisfaction loss term to the standard cross-entropy objective. The symbolic constraint operates as a regulariser that pushes the network to develop more discriminative representations of normal versus anomalous behaviour, rather than encoding deep domain knowledge per se. The system is evaluated on the CIC-

IDS2017 dataset, where five attack classes are withheld entirely from training and treated as zero-day threats at test time. The Hybrid-LTN achieves 8–12% higher binary accuracy and 7–10% higher F1-score on these unseen attacks compared to a standard 1D CNN baseline, demonstrating that logical regularisation during training improves generalisation to out-of-distribution inputs. The authors acknowledge several important limitations: scalability remains an open concern as network environments grow in complexity, and the approach does not inherently improve the explainability of individual decisions, since the symbolic component is discarded at test time and only the encapsulated neural network is used for inference. The logic, in other words, shapes the network during training but leaves no interpretable trace in the deployed model.

A markedly different integration strategy is pursued by Kalutharage et al. (2025) [24], who combine a neural anomaly detection component with a symbolic knowledge graph in a post-hoc reasoning pipeline targeting IoT network security. Their system uses an autoencoder to detect anomalous network flows, then applies SHAP values to identify the most influential features behind each flagged instance. These feature attributions are cross-referenced against a cybersecurity knowledge graph, whose nodes represent network features and whose thresholds encode known attack behaviour, to determine whether a detected anomaly constitutes a legitimate attack. When expert knowledge is incorporated to refine the graph thresholds, the system achieves 97% overall detection accuracy and 100% accuracy in mapping confirmed attacks to MITRE ATT&CK tactics and techniques via an LLM-assisted interface. The integration of domain knowledge is demonstrably valuable: a purely data-driven knowledge graph achieves 91% accuracy on the IoT dataset used, while the expert-augmented version reaches 97%, a difference the authors attribute directly to the context-specific nature of IoT traffic and the inability of data alone to generalise across diverse device types and network configurations. The approach is thoughtful in its handling of explainability since SHAP values and CIA triad mappings give security analysts a clear account of why an anomaly was classified as an attack. Moreover, it also illustrates a structural limitation common to pipeline architectures: the neural and symbolic components are not jointly trained. The knowledge graph is built and updated separately, the symbolic reasoning is applied after the neural output has already been produced, and there is no gradient flow connecting the two. The system is, in Kautz's taxonomy, closer to a Type 2 or Type 4 architecture than a genuinely cooperative Type 3 system, and its robustness therefore depends heavily on the quality and completeness of the knowledge graph at inference time. The authors themselves note that reliance on a static knowledge graph limits the system's ability to adapt to genuinely novel threats that do not align with predefined patterns, and that the expert update process remains largely manual.

The broader landscape is surveyed by Hakim et al. (2025) [20], whose systematic review of 127 publications spanning 2019 to mid-2025 provides the most comprehensive current overview of NeSy AI in cybersecurity. They introduce a Grounding-Instructibility-Alignment (G-I-A) framework for evaluating NeSy systems, and their analysis of network intrusion detection specifically confirms several of the patterns already visible in the individual works above. LTN-based classifiers incorporating logical constraints show consistent precision improvements, multi-view approaches fusing network traffic with symbolic ontology features achieve higher F1 gains over single-view baselines, and knowledge-graph-

enhanced systems like KnowGraph demonstrate strong inductive performance on enterprise network datasets. Nevertheless, the survey emphasizes the limitations in this field. The most critical gap identified is the absence of standardised NeSy-specific benchmarks: existing intrusion detection datasets lack the structured symbolic knowledge representations necessary to evaluate hybrid reasoning, contain no annotation for multi-stage attack campaigns, and represent static snapshots rather than the evolving temporal sequences that real detection systems must handle. The survey also identifies knowledge engineering bottlenecks as a persistent practical barrier, since building and maintaining high-quality symbolic knowledge bases requires specialised cybersecurity expertise, constraining organisational adoption due to the scarcity of that expertise and the manpower required for manual maintenance of the knowledge bases.

Taken together, these works highlight both the potential and the limitations of current NeSy NIDS research. The potential can be seen through symbolic knowledge that consistently improves precision, reduces false positives, and in several cases substantially enhances robustness to unseen attacks. The limitations, however, are equally consistent. The majority of existing systems treat the neural and symbolic components as loosely coupled modules rather than as genuinely integrated, co-trained systems. Symbolic knowledge is typically injected as a fixed constraint or consulted as a post-hoc oracle, rather than participating in a principled probabilistic inference process that jointly reasons over perceptual uncertainty and domain knowledge. Datasets remain synthetic and narrow, with evaluations largely confined to single-dataset benchmarks such as CIC-IDS2017 that do not reflect the diversity of real-world host-based activity. The explainability that these systems provide is often structural rather than semantic: LTNs, for instance, discard their logical component at inference time, leaving the deployed model as opaque as the neural baseline it was built upon.

These limitations motivate the approach taken in the present work. What is needed is a framework where the neural and symbolic components are not merely adjacent but genuinely coupled: one in which probabilistic uncertainty propagates naturally between perceptual inputs and logical reasoning, in which learning and inference operate over the same unified computational structure, and in which the logic program remains a transparent, interpretable artefact even after training. This is precisely the architecture that DeepProbLog provides. By grounding logical reasoning in a principled probabilistic semantics and enabling end-to-end learning through the gradient semiring, DeepProbLog offers a path toward NeSy intrusion detection that is simultaneously more rigorous in its handling of uncertainty, more transparent in its reasoning, and more data-efficient through the incorporation of structured domain knowledge.

Chapter 3

Introducing the DeepProbLog framework in Network Intrusion Detection

This chapter presents the background on the DeepProbLog framework that is needed to understand the principles, motivation, and design behind the neuro-symbolic network intrusion detection system developed in this thesis. It explains the core concepts of the DeepProbLog framework, its role in the proposed network intrusion detection system (NIDS), and the expected benefits and limitations of using probabilistic logic for intrusion detection.

The chapter is organised as follows. Section 3.1 positions DeepProbLog within network intrusion detection and motivates the choice of framework. Section 3.2 fixes the scope and assumptions of the proposed detector. Section 3.3 states the benefits that the design is expected to deliver, and Section 3.4 records the limits of the framework.

3.1 DeepProbLog within network intrusion detection systems

Section 2.2 introduced neuro-symbolic AI as a family of approaches that combine neural learning with symbolic knowledge representation, and placed DeepProbLog within Kautz's taxonomy as a **Type 3** system, in which the neural and symbolic components interact in both directions during learning and inference [25]. This thesis uses DeepProbLog because it keeps both components active during training and inference, a property that matters for a NIDS. If symbolic knowledge is used only as a training-time regulariser and removed at test time, the deployed detector is still essentially a neural black box. If symbolic knowledge is consulted only after a neural detector has produced an anomaly score, the symbolic component can explain or filter a prediction, but it cannot shape the neural representation that produced it. DeepProbLog occupies a different position: neural predicates, probabilistic facts, and logical rules are compiled into a single differentiable computation graph. Gradients pass through the neural and symbolic parts, and the symbolic program remains part of the model at inference time. It is worth noting that this Type 3 coupling is absent from most of the NIDS literature surveyed in Section 2.2. The mechanics of DeepProbLog, ProbLog's distribution over possible worlds, the neural predicate, and the gradient semiring that makes

the arithmetic circuit differentiable, were detailed in Section 2.2; this section assumes that material and focuses on what the framework is asked to provide in a NIDS.

Selecting a neuro-symbolic framework for a NIDS came down to three requirements: (1) a close coupling that lets the symbolic and neural parts interact fully rather than run side by side; (2) a lenient symbolic layer that allows soft, non-rigid decisions instead of hard filtering; and (3) the ability to learn, discard, and prioritise candidate rules during training. DeepProbLog satisfies all three, and the choice follows directly from the limitations identified in the related work. Logic Tensor Networks, such as those used by Grov et al. [18] and Bizzarri et al. [8], encode logical constraints during training, but the symbolic component is not necessarily present once the model is deployed. Knowledge-graph approaches, such as the pipeline of Kalutharage et al. [24], preserve a symbolic layer at inference time but do not train the symbolic and neural components as one model. DeepProbLog provides the coupling required here: the classifier exposes a probability distribution through a neural predicate, while the logic program reasons over this distribution together with symbolic evidence derived from high-level rules defined on packet and flow features.

A neural predicate in the proposed system has the form

$$nn(packet_net, [P], Y, [benign, a_1, a_2, a_3, \ldots]) :: packet_class(P, Y).$$

For a packet input P, the neural network defines a distribution over traffic classes, and the predicate $packet_class(P,Y)$ holds with the probability assigned to class Y. A flow classifier is exposed in the same way. The logic program then defines rules over these neural predicates and over symbolic evidence predicates such as abnormal packet length, unusual TCP flags, high source TTL, low destination load, or protocol-specific service context. A query is answered by grounding the relevant rules, compiling the resulting formula into an arithmetic circuit, and evaluating that circuit to obtain a class probability.

The methodological reason for using a probabilistic logic program rather than a deterministic rule engine stems from the fact that intrusion evidence is rarely absolute. A high packet length, a rare service, or an unusual source TTL may support an attack hypothesis without proving it. DeepProbLog allows these pieces of evidence to remain probabilistic. Moreover, the framework allows symbolic rules to call neural networks during decision making, which is useful for complex rules that require low-level perceptual patterns. The neural classifier contributes a learned class distribution, while symbolic predicates contribute structured evidence with fixed or learnable confidence values. As a consequence, the final prediction is not a hard override from one component to the other. It is the result of a probabilistic proof in which neural and symbolic evidence can reinforce, block, or fail to support a class.

This distinction matters for interpretability. A deterministic rule engine can be easy to inspect, but it often behaves as a separate alerting system placed next to a neural classifier. In that design, disagreement between the two systems must be resolved outside the model. DeepProbLog makes the resolution policy part of the model definition. If a symbolic guard suppresses a benign prediction, or if a same-class symbolic rule boosts an attack class, that choice is visible in the logic program and can be evaluated as a separate architecture. The thesis treats those policies as part of the experimental method rather than as implementation details.

3.2 Design assumptions and scope

The proposed detector is supervised. It assumes that labelled benign and attack traffic is available for training, validation, rule scoring, and testing. This excludes fully unsupervised anomaly detection from the scope of the thesis. The reason is not that unsupervised detection is unimportant, but that the research question here concerns how labelled neural learning can be combined with symbolic attack knowledge. A supervised setting also allows the evaluation to distinguish between detecting any attack and assigning the correct attack family.

The system is feature based. It does not inspect payload contents and does not perform deep packet inspection. This choice keeps the system closer to settings where payloads may be encrypted, unavailable, or sensitive. It also means that the detector cannot rely on signatures contained only in payload bytes. The symbolic rules must therefore be written over metadata, header fields, service indicators, timing statistics, and flow summaries. This constraint is important when interpreting the results. A failure to detect a payload-only attack is not evidence that neuro-symbolic reasoning is useless; it may simply mean that the chosen feature view does not expose the relevant signal.

The thesis also assumes an offline experimental setting for evaluation. Models are trained and tested on fixed splits, even when the packet and hybrid variants are motivated by online detection. This is common in NIDS research because public datasets are distributed as static captures or feature tables. The implementation still models the temporal difference between packet and flow evidence, but the evaluation does not claim to reproduce every operational detail of a live network sensor.

3.3 Expected benefits

The first expected benefit is **interpretability**. Post-hoc explainability methods such as SHAP and LIME can describe a neural prediction after it has been made, but they do not participate in the prediction itself. In a DeepProbLog program, the symbolic rules are part of the classifier. A prediction can therefore be traced through named predicates, learned rule weights, and neural probabilities that directly contributed to the arithmetic circuit. For a security analyst, this is closer to the structure of a rule-based alert than to a feature-attribution heatmap: the explanation says which symbolic conditions supported the alert and how they interacted with the neural classifier.

The second expected benefit is **guidance in low-data settings**. Public intrusion datasets often contain many benign samples and far fewer examples of some attack families. A neural model trained with only a handful of examples per class may fail to learn regularities that are already known to security analysts. A symbolic rule can inject such regularities directly into the learning problem. For example, if a specific family is only plausible under a particular protocol or service context, the rule can restrict the hypotheses considered by the model instead of asking the neural network to rediscover the restriction from limited data. This is the same basic motivation behind the low-data advantage reported for DeepProbLog in non-security domains by Manhaeve et al. [31].

The third expected benefit is **control over false positives and false negatives**. A NIDS is rarely judged by accuracy alone. In heavily imbalanced traffic, a detector can look accurate while missing attacks, or it can detect many attacks while flooding analysts with benign alerts. The symbolic layer provides explicit places to tune this trade-off. A strict rule can require symbolic corroboration before an attack class is accepted. A more aggressive rule can suppress a benign prediction when attack-like evidence is present. A disqualifier can prevent a broad rule from firing in contexts where it is known to be unreliable. These choices are treated as experimental variants rather than hidden thresholds. The false-positive problem the symbolic layer targets here is one of the long-standing practical barriers to operational NIDS that Ahmad et al. [1] flag in their systematic review.

The fourth expected benefit is **portability**. A rule that captures general attack behaviour should be less tied to one network topology than a weight learned from one dataset. This does not mean rules transfer automatically. Protocol vocabularies, feature definitions, and class distributions differ across datasets. Still, if part of the detector is written as explicit symbolic structure, then transfer can be studied directly: the same rule can be evaluated on another dataset, or the same neural predicate can be embedded in a new program. The packet and hybrid experiments on standardised NetFlow-v3 data are included partly for this reason. This also responds, in part, to the critique of Apruzzese et al. [4], whose pragmatic assessment of machine-learning NIDS drew attention to how brittle these systems become once they are moved from one network environment to another.

The fifth expected benefit is **robustness**. Neural classifiers can be sensitive to small perturbations in input features, especially when the perturbation moves a sample across a learned decision boundary. Symbolic rules offer a different source of evidence. A protocol equality, a port condition, or a high-level conjunction of traffic properties may remain stable when individual numeric features are corrupted. The experiments therefore include noise and adversarial perturbation settings to test whether the neuro-symbolic models degrade more gracefully than their neural baselines.

3.4 Limitations of the framework

DeepProbLog also imposes practical limits. Grounding a program, compiling an arithmetic circuit, and evaluating that circuit is more expensive than a standard neural forward pass. The cost is expected to grow with the number of grounded rules, the number of neural predicate calls, and the size of the arithmetic circuit. This matters for intrusion detection because a packet-level detector may need to operate close to line rate. The symbolic program must therefore stay small enough to compile and evaluate, and the implementation must avoid unnecessary recompilation.

The framework also does not remove the knowledge-engineering problem. Learnable rule weights allow the system to down-weight weak rules, but someone still has to propose candidate predicates and candidate rule bodies, an effort that Hakim et al. [20] identify as one of the main practical barriers to neuro-symbolic adoption in cybersecurity. This thesis reduces that burden by using a feature-ranking and rule-calibration pipeline, but the final symbolic program is still constrained by the features available in the datasets and by the

attack knowledge encoded in the candidate rules.

Probabilistic logic programming also makes assumptions that are not always perfectly aligned with network traffic. Probabilistic facts are often treated as independent even when the underlying features are correlated. Flow duration, packet counts, byte counts, and rates are not independent measurements in a strict statistical sense. The system is therefore best understood as a structured probabilistic classifier rather than as a complete causal model of network traffic.

Another limitation is that symbolic rules can overfit in a different way from neural weights. A rule may look meaningful because it uses a security concept, but still be tied to a dataset artefact such as a fixed port, a testbed-specific service, or an attack tool configuration. For this reason, the methodology separates rule discovery, rule calibration, and final testing. A rule is not accepted simply because it sounds plausible. It must show useful precision, recall, or false-positive behaviour on non-test data, and its effect must be measured against neural and symbolic baselines.

The last limitation concerns label quality. Public NIDS datasets contain known artefacts, including inconsistent labels, synthetic traffic patterns, class imbalance, and sometimes incomplete documentation of how attacks were generated. The thesis does not treat the labels as perfect ground truth for real networks. They are used as the best available experimental labels for comparing the proposed models under controlled conditions. This affects the interpretation of every result: the experiments can show whether the proposed neuro-symbolic design helps on these benchmarks, but they cannot prove deployment readiness by themselves.

Chapter 4

Methodology

This chapter describes the research method used to design and evaluate the proposed neuro-symbolic network intrusion detection system, and elucidates the rationale behind the main choices. The concrete datasets, model implementation, and rule program that realise these choices are deferred to Chapter 5.

The chapter is organised as follows. Section 4.1 presents the system overview and the different views, as well as the two task formulations. Section 4.2 describes how symbolic knowledge is extracted from data. Section 4.3 defines the neuro-symbolic couplings, Section 4.4 states the threat model assumed by the robustness experiments, and Section 4.5 fixes the evaluation protocol, before a final section sets out the threats to validity. The experiments built on this method target the three research questions of the introduction: learning efficiency and performance, robustness and generalisation, and explanation.

4.1 System overview

The proposed system is built around two observations. First, packet-level and flow-level traffic views contain different information. A packet exposes local header and protocol fields such as ports, flags, length, and TTL. A flow aggregates packets that share a 5-tuple identification (i.e. source IP, destination IP, source port, destination port, and protocol) and exposes timing, directionality, packet and byte counts, and rate-based behaviour. Some attacks leave useful traces in individual packets; others are visible only after aggregation. Second, neural and symbolic components are useful for different reasons. The neural component learns nonlinear patterns from the full feature vector, while the symbolic component encodes compact, auditable traffic conditions that are meaningful to an analyst. The proposed NIDS combines both: each available view is classified by a neural model, symbolic predicates extract rule evidence from the same instance, and a DeepProbLog program combines the neural and symbolic evidence into a final class distribution.

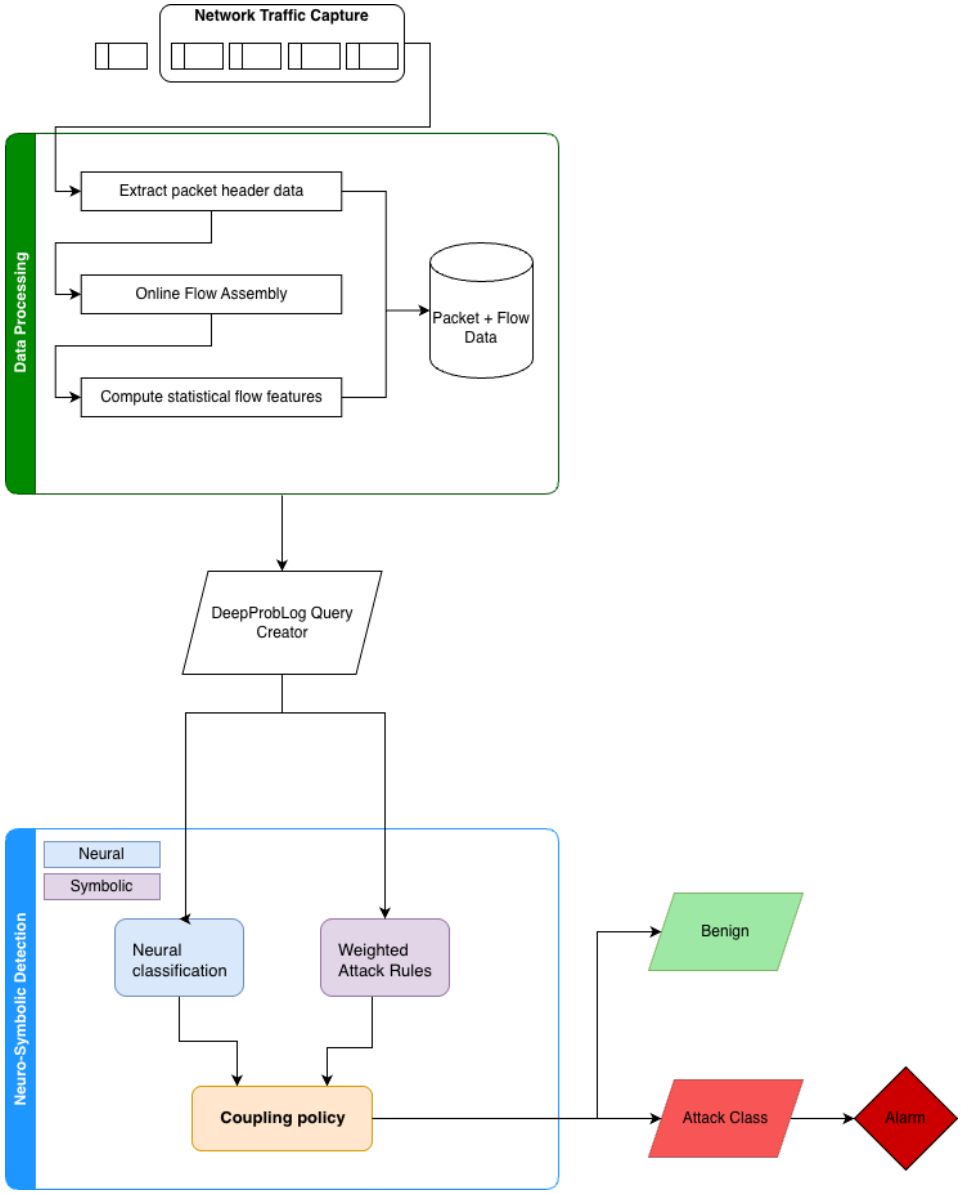


Figure 4.1: System overview of the proposed neuro-symbolic NIDS.

Figure 4.1 shows the shared structure. Data is collected from incoming network traffic and stored in both packet and flow representations: packet features are extracted from headers, while flow features are produced by assembling incoming packets into flows and aggregating their properties. The query creator then submits the data to the model. The same input instance is read in two representations: a standardised feature vector for the neural classifier and a raw feature vector for the symbolic predicates. The neural classifier produces a distribution over classes, the symbolic predicates produce graded rule evidence, and the DeepProbLog program combines the two under a chosen coupling policy into a final distribution, from which the highest-probability class is taken as the final prediction. Attack predictions are then raised as alerts.

The system is evaluated over three views in order to answer all the operational questions. Each view represents different assumptions about how much evidence is available and when.

4.1.1 Packet, flow, and hybrid views

The **packet-oriented** variant classifies individual packets. It is the closest of the three to a real-time detector, because it can decide as soon as a packet is parsed, without waiting for a flow to complete. This makes it the fastest view in principle: packet-level detection can be pushed to very high throughput and even compiled into programmable data planes that classify at line rate, as in P4-based packet NIDS [9], where a verdict is reached on the switch before the flow even exists. It also captures details that disappear after aggregation, such as flags, packet size, and protocol-stack information. The cost of this speed is coverage: many attack behaviours are not fully expressed in a single packet, since brute-force attempts, scans, and slow denial-of-service behaviour are temporal, so a packet classifier may see one fragment of the pattern but not the behaviour as a whole. The packet-level evaluation therefore has a stricter interpretation. A good packet-level result suggests that useful attack evidence is visible before aggregation, while a weak result does not mean the attack is undetectable; it may mean the relevant behaviour only appears across several packets.

The **flow-level** variant classifies aggregated sessions and trades latency for context. It cannot classify traffic before enough packets have been observed to compute the flow features, but it can express behaviours that are invisible at the packet level, such as unusually low return traffic, abnormal packet-count ratios, or timing patterns. These are the conditions analysts often use when reasoning about scans, denial-of-service, and brute-force traffic, which makes the flow representation the main setting for testing whether symbolic priors can complement a neural classifier. The flow setting also supports a clearer comparison with much of the NIDS literature, because many public datasets are released as flow tables.

The **hybrid** variant classifies a packet together with the flow to which it belongs. This is more expensive than either single-view system, but allows rules that neither a packet-only nor a flow-only detector can express: a rule can require consistency between packet and flow evidence, or use one view to guard the other. The hybrid design is evaluated because late fusion alone is not the same as neuro-symbolic coupling. A late-fusion baseline combines packet and flow neural scores after both networks have predicted independently, whereas

the hybrid DeepProbLog model combines the two views inside the logic program, so cross-view rules participate in training and inference.

4.1.2 Detection pipeline

All three variants follow the same abstract pipeline. The input is represented as a packet, a flow, or a packet-flow pair. The active neural classifier produces a class distribution, the symbolic component reads the same raw instance and produces evidence predicates, and the DeepProbLog program combines the two according to one of several coupling architectures. The class with the highest final probability is selected, and attack predictions are treated as alerts. The hybrid variant adds an assembly stage: a packet is available immediately, but its flow may not yet be mature, so the system maintains flow state while packets stream in. When the flow representation is unavailable, the flow-side classifier contributes a neutral distribution so that missing flow evidence does not corrupt the packet-level decision.

The symbolic component is intended to produce evidence predicates based on domain knowledge defined over the attack classes, so the inner workings of the symbolic rules need to be clearly specified.

Symbolic rules output evidence predicates that are graded by a continuous value between 0 and 1, representing the degree of support for a given attack class. A mined rule receives learnable parameters, which are optimised during training: a rule that is consistently useful for classification will be assigned a higher weight, while a rule that is less informative will receive a lower weight. However, a rule defined over features can only output the correct evidence predicate if the features themselves are informative for a given attack. Therefore, a trigger must be defined. Since attacks are usually characterised as abnormal behaviour, the trigger is set as a threshold over the benign distribution of the feature. A rule is triggered when the feature value reaches extreme values with respect to the benign distribution, and the threshold is set to a quantile of the benign distribution. The features and quantiles are set during candidate rule mining, which will be described in Section 4.2.2. The rule's weight is then learned during training, and the evidence predicate is computed as the product of the rule's weight and the trigger value. This allows the symbolic component to provide graded evidence for each attack class, which can be combined with the neural classifier's output in the DeepProbLog program.

4.1.3 Tasks: binary and multiclass

The thesis evaluates two task formulations. The first is binary one-versus-benign classification, where a model distinguishes one selected attack family from benign traffic. This setting isolates one family and makes it easier to inspect which features separate it from benign traffic. The second is multiclass classification, where benign and all retained attack families are predicted together. The multiclass setting is harder and closer to deployment, because the model must avoid confusing attack families with one another. The two formulations answer different questions: binary experiments ask whether symbolic evidence can help detect a specific family under controlled conditions, while multiclass experiments ask

whether the same ideas survive when families compete for probability mass. A symbolic condition that separates DoS from benign may not separate DoS from PortScan, which is why the methodology includes disqualifier predicates and exact-attack metrics rather than treating all non-benign predictions as equally correct.

The class taxonomy used in each experiment is itself a deliberate scoping choice. Related attack sub-labels are merged into broader families and very sparse classes are excluded, which narrows the claim a result can make but keeps the task learnable and the comparison clean. The thesis treats this narrowing as an experimental boundary rather than a property of the detector, and the concrete class list for each dataset is given in Section 5.1.3. The experimental unit also depends on the variant: a packet experiment scores one packet per sample, a flow experiment one flow, and a hybrid experiment a packet together with its assembled flow. A packet-level test set therefore contains many packets drawn from comparatively few flows, so a packet-level score is not directly comparable to a flow-level score on the same traffic.

4.2 Symbolic knowledge extraction

This section describes how symbolic knowledge is derived from data. The procedure is dataset-agnostic: it takes labelled non-test traffic and produces a ranked feature vocabulary, a set of calibrated candidate rules, and the role each rule plays. The concrete features and thresholds it produces for each dataset are reported in Chapter 5.

4.2.1 Data roles and leakage control

Each dataset is split once into a 60/20/20 train/validation/test partition, stratified by class so that every split preserves the class distribution, which gives the three roles: a training pool, a validation split, and a held-out test split. The training pool plays a double part. It trains the classifier, and it also serves as the scoring and reference pool from which symbolic features are ranked and the benign quantile thresholds are estimated, so the same non-test traffic that fits the network also mines the candidate rules. The test split is held out completely and is never used for feature ranking, benign quantile estimation, rule selection, threshold calibration, hyperparameter selection, or model selection.

Splitting is flow-stratified: all packets belonging to one flow are assigned to the same partition. A random packet-level split would leak information, because packets in the same flow are statistically dependent, so a packet from one flow in training and another from the same flow in testing would inflate the estimate of generalisation. From these three splits the experiments draw the subsets they actually use. The training subset is parametrised by $k \in \{5, 10, 25, 50, 500\}$ samples per attack class, with benign sampled as a fixed multiple of the total attack count and five independent draws per size, so the training subset is the only part that changes across the sweep. The validation and test splits are subsampled once to fixed-size subsets, and those same two subsets are reused for every training size and every replicate. Holding the evaluation data fixed while only the training subset grows is what makes it possible to measure whether symbolic knowledge helps most when the neural

model has little labelled data, and whether that advantage disappears as the baseline receives more examples.

Feature ranking and threshold estimation run on the training pool, which answers a first question: which features and thresholds look like plausible symbolic evidence for an attack. However, a feature can look selective on the same data it was mined from purely by chance, and this is most dangerous in the low-data setting, where a handful of samples can make a distribution appear far more separated than it really is. The validation split is used as a consistency check guarding against this possibility. Every mined candidate rule is measured on the validation data and its precision and false-positive behaviour are compared against the results gathered during mining on the training pool. A rule whose behaviour stays consistent across the two is treated as a genuine attack signature and kept; a rule that looked selective on the training pool but drifts on validation is discarded as a mining artefact. The validation split therefore answers a different question from the training pool, namely which of the mined rules are consistent enough to be trusted, and it is also where each surviving rule is assigned its role. Keeping discovery on the training pool and acceptance on the validation split stops a rule from being both proposed and trusted on the same evidence.

4.2.2 Feature selection for the neural and symbolic components

The neural component receives the full preprocessed feature vector, except for fields that would leak labels, timestamps, or topology identifiers. Both the neural baseline and the neuro-symbolic model receive the same neural input, otherwise any improvement could be attributed to a difference in features rather than to the system's architecture itself.

The symbolic component has the opposite constraint. A symbolic rule should be short enough to read, meaningful enough to inspect, and selective enough to avoid firing on ordinary benign traffic. The symbolic vocabulary is therefore selected by an attack-wise feature-ranking procedure. For each attack class A and each candidate feature f, the pipeline performs a one-versus-benign comparison, with samples labelled A as the positive attack sample and benign samples as the negative sample. The ranking is attack-wise because defining features differ between attack classes. Numeric and categorical features are scored with different statistics.

Numeric features. Four statistics are computed. The two-sample Kolmogorov-Smirnov statistic,

$$\mathrm{KS}(f,A) = \sup_x |F_A(x) - F_B(x)|, \tag{4.1}$$

captures the largest distributional difference between attack and benign.

Cliff's delta,

$$\mathrm{CD}(f,A) = \Pr(X > Y) - \Pr(X < Y), \quad X \sim f \mid A,\ Y \sim f \mid B, \tag{4.2}$$

gives both strength and direction; its sign later decides whether a rule should target high or low benign quantiles.

The Hellinger distance over shared histogram bins,

$$\mathrm{HD}(f,A) = \sqrt{\frac{1}{2}\sum_{i=1}^{m}\left(\sqrt{p_{A,i}} - \sqrt{p_{B,i}}\right)^2}, \tag{4.3}$$

captures distributional separation,

and a folded AUC,

$$\mathrm{AUC}(f,A) = \max\left(\Pr(f(X) > f(Y)), 1 - \Pr(f(X) > f(Y))\right), \tag{4.4}$$

treats the feature as a one-dimensional classifier and rewards separation in either direction.

Several statistics are used because each fails differently: KS detects shifts but not direction, Cliff's delta gives direction but can miss shape differences, Hellinger depends on binning, and AUC is near 0.5 when distributions differ mainly in variance. A consistent feature scores well across all four statistics, while a feature that scores high on one but low on the others is less likely to be genuinely selective. The four statistics are therefore combined into a single composite score that ranks numeric features for each attack class.

Categorical features. These are handled without assuming an ordering. The categorical Hellinger distance is computed over class-conditional category distributions, and a descriptive AUC replaces each category c with a smoothed empirical attack rate,

$$s(c) = \frac{n_A(c) + \alpha\bar{y}}{n(c) + \alpha}, \tag{4.5}$$

where $n_A(c)$ is the number of attack samples in category c, $n(c)$ the total, $\bar{y}$ the global attack rate, and $\alpha = 1$ a smoothing constant. Cramér's V is computed from the class-by-category contingency table,

$$\mathrm{CV}(f,A) = \sqrt{\frac{\chi^2/n}{\min(r-1, k-1)}}, \tag{4.6}$$

and total variation distance is $\mathrm{TVD}(f,A) = \frac{1}{2}\sum_i |p_{A,i} - p_{B,i}|$.

Composite score. The raw statistics are normalised per attack and combined as a weighted average,

$$\mathrm{score}(f,A) = \frac{\sum_{m\in\mathcal{M}(f)} w_m \hat{m}(f,A)}{\sum_{m\in\mathcal{M}(f)} w_m}, \qquad \hat{m}(f,A) = \frac{m(f,A) - \min_g m(g,A)}{\max_g m(g,A) - \min_g m(g,A) + \varepsilon}, \tag{4.7}$$

where $\hat{m}$ is the per-attack min-max normalisation of statistic m and the weights w_m are configuration values reported with the implementation (Section 5.4). KS carries the largest weight as a general measure of numeric shift, while the categorical statistics carry smaller weights because they act as supporting evidence. The output is one ranked feature list per attack class. The composite score is a screening tool, not a claim of causality: a high score means the feature is worth considering, and the later calibration step decides whether a rule built from it has acceptable precision, recall, and false-positive behaviour.

4.2.3 Candidate rule generation and calibration

Ranked features are converted into candidate rules. For a numeric feature, the sign of Cliff's delta decides whether the attack is associated with unusually high or low values, and the feature is represented by benign-quantile predicates. For a categorical feature, the most attack-associated categories become equality predicates. Candidate predicates are combined into short rule bodies favoring compact clauses with one to three evidence predicates that are easier to inspect, cheaper to ground, and less likely to describe a dataset coincidence.

The validation split calibrates each candidate by precision, recall, and benign false-positive rate, and assigns it a role from those metrics. A high-precision, low-false-positive rule may become an **anchor** that supports an attack class on its own. A lower-precision but higher-recall rule becomes a **rescue** rule that is trusted only when the neural classifier also supports the class, or when it blocks a benign prediction. A rule that is precise for the wrong class becomes a **disqualifier**. Separating rule roles allows a more nuanced approach to their application. By assigning each rule a specific role based on its performance metrics, the system can use each rule type where it is reliable.

4.3 Neuro-symbolic coupling

The experiments compare several ways of combining neural and symbolic evidence. These variants express different operational preferences about the agreement policy between the neural and symbolic components. They are bounded by two degenerate endpoints that are not couplings but baselines: a neural-only program that decides from the classifier alone, and a symbolic-only program that decides from the mined rules alone. Together with an external signature engine, these are the reference baselines the couplings are measured against, defined in Section 6.1.1; the present section concerns only the couplings that sit between the two learned endpoints.

The coupling variants between these extremes fall into three families. The first focuses on the agreement between neural and symbolic evidence.

Hard-gate A hard-gate accepts a neural class prediction only when class-compatible symbolic and neural evidence are present, so the two layers must agree and neither can solely initiate a prediction.

Same-class boost The same-class boost approach relaxes the previous requirements, keeping the neural path open and only adding probability mass when symbolic evidence supports the same class. The role of the symbolic component is to reinforce uncertain neural predictions when the defined rules are consistent with the information provided by the input.

The second family acts on the benign decision.

Guarded The **guarded** coupling suppresses a benign prediction when broad attack evidence is present, letting the symbolic layer rescue attacks the classifier would otherwise miss.

The third family is more permissive towards the symbolic layer, allowing it to make decisions where it is trustworthy.

Anchored The anchored architecture promotes high-precision rules to standalone anchors while weaker rules stay gated behind the classifier. The weaker rules still contribute, but only when the neural classifier's output supports them, so the high-precision anchors add recall without the lower-precision rules raising false alarms.

Safety-net The OR-rescue, or safety-net, is the most permissive, predicting an attack when either the classifier or a class-specific rule fires, and is included mainly to probe robustness because the symbolic path can still fire when neural confidence collapses under noise.

Product-of-Experts (PoE) The product-of-experts architecture multiplies the neural distribution with a separate rule-activation expert, so the symbolic evidence reshapes the whole posterior rather than gating it.

Evaluating these policies as separate models, rather than hiding them behind one tuned threshold, makes the precision-recall trade-off visible. Each coupling policy expresses a different operational preference, and the experiments measure whether the symbolic layer can improve detection, reduce false alarms, or stabilise predictions in low-data or noisy conditions.

4.4 Threat model

Different experiments assume different adversaries, and stating each one keeps the results interpretable. The detector is a feature-based NIDS that reads packet headers and/or flow summaries throughout. In every case, the attacker operates at the level of traffic features rather than payload content. Three settings are considered, from a passive attacker who does not adapt, through non-adversarial measurement degradation, to an active evader.

Nominal operation. The detection, efficiency, and explainability experiments assume an attacker who mounts attacks from known families that leave observable traces in packet headers or flow summaries, and who does not adapt to the detector. Traffic is generated as in the benchmark captures, with no attempt to evade detection, mislead the rules, or poison the training data. The defender has labelled benign and attack traffic for supervised training, and the test traffic is drawn from the same network and distribution as the training traffic. The move to a different network topology is treated separately as a generalisation test rather than an adversarial one. Under this model the only question is detection quality: whether the system flags attacks, assigns the correct family, and keeps benign false alarms low.

Non-adversarial degradation. The first robustness setting assumes no attacker intent at all. The measurements the detector relies on simply become unreliable, for example through packet loss, sampling, or a feature extractor that drifts from the one used at training time. This is modelled as Gaussian noise added to the test features, and it asks only how much of a model's clean performance survives when its inputs are corrupted.

Active evader. The second robustness setting assumes an attacker who controls only their own attack traffic and tries to make it pass as benign, so the perturbation is applied to attack samples and targeted toward the benign class, within a per-feature budget. The attacker cannot perturb other users' benign flows, which has a direct consequence for evaluation: benign traffic is left untouched, the attack false-positive rate is therefore fixed by construction, and the metric that moves is detection. The attacker is further constrained to features they could plausibly control. The attack-defining columns that the symbolic rules read, such as destination port, protocol, and flag counts, are frozen, so the perturbation respects the structure of network traffic rather than moving arbitrary feature values. The thesis does not model an attacker who knows the rule set and crafts traffic to satisfy or avoid specific clauses, nor one who poisons the training data; these are real threats but lie outside the scope of the experiments, which test whether the symbolic layer degrades more gracefully than a neural baseline under bounded, realistic perturbation.

4.5 Evaluation protocol

This section starts by describing the metrics used to evaluate and benchmark the models against the diverse baselines. Then, it describes the experimental protocol, including the training and evaluation procedure, the replicates, and the randomisation strategy used to answer the research questions previously set.

4.5.1 Metrics

The primary metrics are macro-F1 and attack false-positive rate. Macro-F1 gives equal weight to each class and is the headline score because the test sets are benign-heavy, so a model can reach high accuracy while missing most attacks. Accuracy is reported but not used as the main criterion for the same reason. The secondary metrics separate three outcomes that accuracy would merge. Let A be the set of attack classes and B benign. The detection rate, exact-attack rate, and attack false-positive rate are

$$\mathrm{DR} = \frac{|\{i : y_i \in A \wedge \hat{y}_i \in A\}|}{|\{i : y_i \in A\}|}, \quad \mathrm{EAR} = \frac{|\{i : y_i \in A \wedge \hat{y}_i = y_i\}|}{|\{i : y_i \in A\}|}, \quad \mathrm{AFPR} = \frac{|\{i : y_i = B \wedge \hat{y}_i \in A\}|}{|\{i : y_i = B\}|}.$$

Detection rate measures whether attacks are detected at all, exact-attack rate whether they are assigned to the correct family, and attack false-positive rate how often benign traffic is flagged. A model that fails to separate attacks has failed exact-attack classification but may still be useful as an alerting system, so the confusion matrix is read both as a full multiclass problem and with attack families collapsed into one super-class.

Weighted-F1 is also reported as a secondary score. Unlike macro-F1 it weights each class by its prevalence,

$$\mathrm{wF1} = \sum_c \frac{n_c}{N} \mathrm{F1}_c$$

where n_c is the support of class c, N the total number of test samples, and $\mathrm{F1}_c$ the per-class F1; reading it alongside macro-F1 separates performance on the abundant classes from performance on the rare attack families. Inference time, memory use, and arithmetic-circuit size are also recorded, because a neuro-symbolic NIDS is only useful if the arithmetic circuit can be evaluated at a practical cost.

The explainability experiments use three further metrics, all read from the same arithmetic circuit that produces the predictions. Let $\mathcal{S}_A$ be the set of raised attack alerts. *Coverage* is the share of those alerts whose winning proof path is a named clause rather than the open neural path,

$$\mathrm{Cov} = \frac{|\{i \in \mathcal{S}_A : \text{a named clause decides } \hat{y}_i\}|}{|\mathcal{S}_A|}.$$

Fidelity of a clause c is its test precision when it decides an alert, that is, the fraction of the alerts it decides that are correct,

$$\mathrm{Fid}(c) = \frac{|\{i : c \text{ decides } \hat{y}_i \wedge \hat{y}_i = y_i\}|}{|\{i : c \text{ decides } \hat{y}_i\}|}.$$

Stability measures whether the explanation of a sample survives retraining. With R replicates and $e_r(i)$ the explanation of sample i under replicate r, it is the fraction of samples on which all replicates agree,

$$\mathrm{Stab} = \frac{1}{|\mathcal{T}|} \sum_{i \in \mathcal{T}} \mathbf{1}[e_1(i) = e_2(i) = \cdots = e_R(i)],$$

over the test set $\mathcal{T}$. For the symbolic models the explanation $e_r(i)$ is the winning clause; for the neural baseline, whose explanations come from SHAP and Integrated Gradients, it is the single most important feature (or, in a looser variant, the top-three feature set).

4.5.2 Protocol

All models are evaluated on the same fixed held-out test subset with the metric definitions above, so that differences reflect the model rather than the measurement. Every reported number is an average over replicates rather than a single run. At each training size k, five independent training subsets are drawn, each tied to its own seed so that the subset sampling, weight initialisation, batch order, and dropout are fixed and reproducible (the seeding is detailed in Section 5.3), while the validation and test subsets stay fixed across all replicates and sizes. The model is re-initialised and retrained from scratch on each subset, and results are reported as the mean and standard deviation across the five. The mean describes expected behaviour over random training subsets, while the standard deviation shows sensitivity to the subset draw, which is more informative than a single best run.

The robustness and explainability questions are evaluated on top of this protocol. Robustness is tested under the threat model of Section 4.4 with two settings: Gaussian noise scaled to each feature's standard deviation, and a constrained adversarial search that perturbs only mutable attack features toward benign. In both, the perturbation is applied to the held-out test subset as an evaluation condition rather than a training source, so the result measures how the trained detector behaves when its inputs shift rather than how well it can learn from augmented data. Explainability is evaluated separately: for selected models, the proof path that wins the arithmetic-circuit evaluation is extracted and compared across training replicates, measuring whether alerts are supported by named clauses, how precise those clauses are on the test set, and whether rule-based explanations are more stable than post-hoc neural attributions.

4.6 Validity considerations

The split, metric, and replicate choices described above guard against the obvious failure modes, but each leaves a residual threat that those guards cannot reach, and this section states what remains.

For internal validity, the leakage controls of Section 4.2.1 keep test labels out of training and rule selection, but they cannot tell a genuine attack signature from a benchmark artefact that happens to hold across every split of the same capture. A rule built on a testbed-specific port, a capture-time field, or an attack-tool fingerprint passes the validation consistency check and still generalises no further than the benchmark it was mined on. The thesis can only limit this by preferring rules whose conditions describe attack behaviour rather than environment, and by reporting where a clause is environment-specific rather than behavioural.

For construct validity, the metrics of Section 4.5.1 stand in for operational usefulness but do not equal it. Macro-F1 and attack false-positive rate proxy the analyst's burden, yet no offline metric captures the true cost asymmetry between a missed multi-stage campaign and a single benign alert, and the coverage, fidelity, and stability scores measure properties of an explanation without establishing that an analyst would actually find it useful. The interpretability claims are therefore claims about faithfulness and stability, not about measured human benefit. To mitigate this, the symbolic rules are matched against known knowledge bases such as the MITRE ATT&CK framework, and the explanations are read as evidence about whether the model is using known attack signatures rather than as a claim that an analyst would find them useful in practice.

Chapter 5

Implementation

The methodology chapter defined the design of the proposed NIDS: the packet, flow, and hybrid variants; the data split strategy; the feature-ranking method; the rule-calibration procedure; and the coupling variants. This chapter describes how those choices are realised in an executable system, and on which concrete datasets. The goal is reproducibility: a reader should be able to reconstruct the system from the data schema, the model components, the rule language, and the execution procedure described here.

The implementation is built around DeepProbLog [31], which extends ProbLog [12] with neural predicates and supplies the neuro-symbolic interface, grounding, and arithmetic-circuit evaluation. The system built for this thesis adds the machinery that intrusion detection needs around it, and two constraints hold throughout: the neural baseline is a standalone network trained outside DeepProbLog, while the neuro-symbolic models embed that same network, with the same architecture and input features, as a neural predicate inside the logic program, so any difference between them is attributable to neuro-symbolic integration, and the symbolic predicates keep learnable confidences but stay tied to named feature conditions, so the model remains inspectable. The chapter follows the data through the system, from the datasets and their two-representation storage, through the neural classifier and the rule-mining pipeline, to the DeepProbLog query, the coupling programs, and the execution engine.

5.1 Datasets

The thesis uses two dataset families. The original CIC-IDS2017 [41] and UNSW-NB15 [37] datasets are used for flow-level experiments because they ship with flow records and are widely used in NIDS research, which makes the flow results directly comparable to prior work. For packet and hybrid experiments, the thesis uses the standardised NetFlow-v3 versions of CSE-CIC-IDS2018 and UNSW-NB15 [40], which share a uniform feature schema across captures and can be connected more cleanly to packet-derived evidence. For the portability experiments, the thesis employs the standardised NetFlow-v3 versions of CSE-CIC-IDS2018 and ToN-IoT [40] to test portability both across datasets and across topologies. Using more than one family is deliberate: a result on a single benchmark may reflect

its topology, traffic generator, or feature extractor, and the families differ in feature schema and class distribution. When a result is reported for one family, it is interpreted within that family unless a separate transfer experiment supports a broader claim.

5.1.1 Original flow datasets

CIC-IDS2017 contains benign traffic and several attack families, with fine-grained raw labels such as DoS Hulk, DoS GoldenEye, FTP-Patator, and SSH-Patator. For the experiments, related variants are merged into broader families: DoS variants under *DoS* (with DDoS folded into DoS in the CICIDS flow setting), Patator variants under *Brute-Force*, and web-attack variants under *Web Attack*. Very sparse classes such as Heartbleed are excluded from the main evaluation. The reported taxonomy is benign plus the families that remain sufficiently represented and separable, namely Bot, Brute-Force, DoS, PortScan, and Web Attack. UNSW-NB15 contains benign traffic and nine attack categories: Analysis, Backdoor, DoS, Exploits, Fuzzers, Generic, Reconnaissance, Shellcode, and Worms. Its files carry label spelling inconsistencies, whitespace artefacts, missing values, and a long tail of rare protocol strings, all handled before splitting so that every model sees a consistent representation.

5.1.2 Standardised NetFlow-v3 datasets

The packet and hybrid systems use NetFlow-v3 data because the feature schema is standardised across datasets, which matters for two reasons: the hybrid system reconstructs flow evidence from packet streams, which is far easier when the target schema is consistent and documented, and the portability experiments require that a rule written for one dataset can be grounded on another without changing the meaning of its feature names or the dimensionality of the feature space. Additionally, the original datasets contain features that are direct outputs of the flow extraction tools used during dataset creation, making it infeasible to estimate and compute those features from the packet-header data received from each network packet input. By contrast, the standardised NetFlow-v3 datasets provide a fully reproducible, uniform feature schema that can be reconstructed from packet-header features, albeit with some uncertainty for a few features.

Two standardised captures are used. NF-CSE-CIC-IDS2018-v3 is the enterprise capture that backs the packet and hybrid experiments, and NF-ToN-IoT-v3, the NetFlow-v3 version of the ToN-IoT testbed [40], provides an IoT/IIoT network that differs from the CIC captures in topology, devices, and attack families. The latter is used only for the cross-topology portability experiment of Section 6.2.2, where a rule grounded on one dataset is evaluated on the other; sharing the NetFlow-v3 schema is precisely what makes that transfer well defined.

The standardised setting also has a limit. NF-CSE-CIC-IDS2018-v3 is not the same capture as CIC-IDS2017, although both come from the Canadian Institute for Cybersecurity testbed family, so the packet and hybrid results are read as results on a related standardised family rather than a packet-level reconstruction of the exact CIC-IDS2017 experiments.

Moreover, the feature space offered by the standardised NetFlow-v3 datasets is a subset of the original flow features, offering a more limited view of the traffic.

5.1.3 Class taxonomy and preprocessing

The label space is fixed before training. Raw labels are normalised by trimming whitespace, resolving spelling variants, and mapping tool-specific sublabels into broader families. This reduces label granularity, so the thesis reports results at the level of the mapped taxonomy: a result on the merged DoS class measures the broader family and does not claim to distinguish all DoS subtypes. Classes too sparse to support low-data training, validation, and testing are excluded from the main multiclass setting, and a removed class is treated as a boundary of the experiment rather than a failure of the detector.

Rows with undefined or infinite numerical values produced by invalid rate or ratio computations are removed rather than imputed, because imputing a rate for a zero-duration flow would create artificial structure in exactly the region where the measurement is undefined, and short flows are not evenly distributed across classes. Protocol values are consolidated where a dataset carries a long tail of rare strings: the most relevant values are retained and the rest grouped into an *"other"* category, which also makes the symbolic vocabulary less dataset-specific. Categorical encoders are fitted on the training subset only, without looking at test labels, and rare categories are grouped when their frequency is too low to support stable learning or rule scoring. Label columns, binary label helpers, raw identifiers, and fields that directly encode the target are removed from the classifier input; IP addresses and timestamps are retained internally only for flow reconstruction and leakage control, not exposed as behavioural features, so the model cannot learn the identity of a testbed host instead of the structure of an attack.

For packet-level and hybrid experiments, raw packets are labelled by matching them to labelled flow records through the 5-tuple and a timestamp constraint, and packets that cannot be matched are excluded. The packet feature set deliberately excludes payload bytes, which keeps the detector applicable to encrypted traffic and avoids the privacy cost of deep packet inspection, at the price of not detecting attacks whose only signature is in payload content. The original datasets are divided with a stratified 60/20/20 train/validation/test split; packet-level data is grouped into flows before the split so that all packets of a flow stay in one partition, as required by the flow-stratification rule of Section 4.2.1.

5.2 Two-representation data storage

Every instance is stored in two forms, because the neural and symbolic components need different representations of the same traffic. The neural form is the dense, standardised vector consumed by the multilayer perceptron; the symbolic form is the raw feature vector read by the threshold and equality predicates. The two are derived from the same configured feature list but are not numerically identical after preprocessing, and keeping both available is what lets the same instance feed a learner that needs scaled inputs and a rule layer that needs concrete thresholds.

Concretely, each dataset instance carries an index, a label, a neural tensor, and a raw tensor. The neural tensor is dense and numeric. The raw tensor preserves the original feature positions, and a feature map links human-readable feature names to tensor positions, so a predicate can be defined by name during program construction and resolved to an efficient tensor lookup during inference. The label is stored twice as well: as an integer class index for the loss, and as a symbolic class name for the DeepProbLog query.

Scaling is applied inside the neural network rather than destructively in the dataset object. During the forward pass, numerical features are standardised using statistics fitted on the training subset, while the symbolic predicates read the unscaled tensor. This detail is essential. A rule that checks whether a destination port equals 8080, or whether a flow duration exceeds a benign quantile, cannot operate on a standardised value, and it is precisely because the raw representation is never touched by neural scaling that symbolic explanations stay readable: a predicate compares against a cutpoint or category in the original unit rather than against a normalised number. The preprocessing state is fitted only on non-test data and applied unchanged to validation and test samples. Standardisation stores a mean and standard deviation per numerical feature; categorical encoding stores a value-to-code mapping. Unknown categories are mapped to a reserved code unless the symbolic predicate can use the original raw value, in which case a raw-value fallback is kept. This fallback matters in the low-data regime, where a $k = 5$ training subset cannot be expected to contain every port, protocol, or service value that appears in validation or testing. Missing numerical values are replaced with zero before neural inference as a guard against undefined gradients, but a missing value does not satisfy a threshold or equality predicate, which prevents the placeholder from becoming accidental symbolic evidence.

5.2.1 Packet and flow feature vectors

The packet vector contains header and protocol information available at parsing time: IP protocol, TTL, differentiated-services field, frame and IP length, transport ports, TCP flags, window size and header length, UDP length, inter-packet time delta, sequence and acknowledgement numbers, ICMP and DNS fields, FTP response code, and a protocol-stack field that preserves application hints such as DNS, FTP, HTTP, and TLS. Fields that would require a completed flow are kept out of the packet-only vector. The flow vector contains aggregate behaviour: duration, forward and backward packet and byte counts, packet-length and inter-arrival-time summaries, rates, state and service fields, and TCP flag summaries. The exact fields differ between the original and standardised datasets, but in both the neural model receives the full non-leaking vector while symbolic predicates read selected interpretable features from the same raw representation. For the hybrid system an instance is a pair (P,F); the packet P is always available, while the flow F may be a sentinel early in a flow's life, in which case the flow-side network returns a neutral distribution so that absence of flow evidence supports no class. The neural input dimension is derived from the final feature count after exclusions and encoding rather than hard-coded, so the same implementation adapts across datasets with different schemas.

The detailed features extracted from the packet header are listed in Appendix B.1.

5.3 Neural classifiers

All neural classifiers are feed-forward multilayer perceptrons. The packet and flow classifiers share the same trunk of three hidden layers with 256, 128, and 64 units, which keeps the two views directly comparable. Each hidden layer is followed by a ReLU activation and dropout with probability 0.3; the final layer maps to the class count and a softmax produces a categorical distribution. The same architecture serves the neural baseline and the neuro-symbolic models, which is essential for attribution: a larger or more specialised network would mix the contribution of the neuro-symbolic integration with architecture engineering. The MLP is intentionally simple, expressive enough to learn nonlinear feature interactions while small enough to train across many low-data replicates.

Randomness is controlled per replicate so that the reported mean and standard deviation are reproducible. The replicate number seeds the training-subset sampling, the neural weight initialisation, the minibatch ordering, and dropout, so each of the five draws at a given k is independent but the whole collection can be regenerated from a known set of seeds. The neural baseline and its neuro-symbolic counterparts share this seeding and the same data preparation, and differ only in whether the symbolic query program is present.

Setting	**Value**
Optimiser	Adam
Hidden layers	256, 128, 64
Dropout	0.3
Neural learning rate (lr_{nn})	3×10^{-3}
Rule-weight learning rate (lr_{dpl})	3×10^{-3}
Soft-tier learning rate (lr_{tier})	1×10^{-3}
Epochs	10
Training batch size	8
Evaluation batch size	8192
Soft-tier steepness (initial)	50
Softmax temperature	annealed $1.0 \rightarrow 0.5$

Table 5.1: Fixed training configuration for the neuro-symbolic models, held constant across all k and replicates.

The classifier is exposed to the logic program through a neural annotated disjunction. For a flow classifier this is written as

$$nn(flow_net, [F], C, [c_1, \ldots, c_n]) :: flow_class(F, C),$$

where F is the flow tensor, C the class variable, and $[c_1, \ldots, c_n]$ the ordered class list; a packet classifier is exposed the same way. The logic program then reasons over the full distribution. For the standalone neural baseline the loss is cross-entropy between the predicted and target class, computed without any logic program; for a DeepProbLog model the objective is the negative log-likelihood of the target query, and gradients update the neural parameters and any learnable rule weights. Table 5.1 lists the fixed training configuration, selected once by

the ablation in Chapter 6 and then held constant across all k and replicates so that no result depends on per-run tuning.

The neural-only MLP baseline uses the same optimiser, neural learning rate, and batch size, but is trained for up to 50 epochs rather than 10, since it has no probabilistic-logic parameters to learn and benefits from longer neural training.

5.4 Offline rule-mining pipeline

Symbolic predicates can be manually written by an expert or derived by consulting an extensive knowledge base, but this is not how the models developed in this thesis are constructed. The predicates are produced by an offline pipeline that turns labelled non-test data into a DeepProbLog program through four stages: scoring, calibration, program assembly, and a soft rewrite. The pipeline runs entirely on the training pool, which doubles as the scoring and reference pool, and the validation split, so the test set never participates in rule construction. The full procedure, with the concrete thresholds used in the experiments, is given as Algorithm 1 in Appendix B; the rest of this section describes how each stage is implemented, and the statistics it uses were defined in Section 4.2.2.

Scoring. The first stage ranks features per attack family. For every attack-versus-benign pair it computes the numeric and categorical statistics, normalises them per attack, and combines them into the composite score, emitting one ranked feature list per attack together with the most attack-associated categorical values. The default composite weights are $w_{\mathrm{KS}} = 0.30$, $w_{\mathrm{CD}} = w_{\mathrm{HD}} = w_{\mathrm{AUC}} = 0.20$, and $w_{\mathrm{CV}} = w_{\mathrm{TVD}} = 0.05$, and the top $K = 12$ features per attack are retained. Each ranked feature carries a predicate hint recording which tier or threshold predicate represents it, so later stages do not re-derive the predicate form. For numeric features the hint also stores the benign-quantile cutpoints at 0.80, 0.90, 0.97, and 0.995 (and their lower-tail mirrors), computed once from the benign traffic in the training pool, so that a predicate can compare a raw value to a fixed cutpoint at inference time without recomputing the reference distribution.

Calibration. The second stage forms short rule bodies from the top-ranked features and evaluates each candidate on the validation split, recording precision, recall, and benign false-positive rate. It compares training and validation behaviour and discards rules whose precision or false-positive rate drifts too far between the two, which removes rules that look selective only on the mining data. Each surviving rule is then assigned a role from its validation metrics, following Section 4.2.3: a rule with precision ≥ 0.90 and false-positive rate ≤ 0.015 becomes an anchor, a rule with recall ≥ 0.35 and false-positive rate ≤ 0.050 becomes a neural-gated rescue, and a rule that is precise (precision ≥ 0.90, false-positive rate ≤ 0.020) for a rival class becomes a disqualifier.

Program assembly. The third stage assembles the surviving rules into a DeepProbLog program. Each single-feature condition becomes a named evidence atom such as *e_sttl_high*, related evidence atoms are grouped into per-class pockets, and the pockets are wrapped into

learnable anchor and rescue predicates whose initial confidences are taken from calibration. The predicate hints from scoring are copied verbatim, so the thresholds and categories in the final program match the ones that were calibrated, with no hand-transcription drift, and the query head is attached according to the configured coupling architecture.

Soft rewrite. The assembled program is written with hard threshold and equality externals. A final rewrite replaces each hard external with the soft module of Section 5.5, and splits any rule mentioning several feature thresholds into single-external helper predicates, so that one cacheable arithmetic circuit can serve every sample. The same logic and the same pockets are preserved, but the program becomes a single stable circuit instead of one that recompiles per sample, which is what connects the mining pipeline to the batched execution path described later.

5.5 Executable symbolic predicates

An evidence predicate is a small logical statement about one feature. Numeric predicates compare a raw value to benign-derived thresholds; categorical predicates test equality with a selected category. For a numeric feature f and instance x, with $v_f(x)$ the raw value and B_f the sorted benign reference, the empirical benign rank is

$$q_f(x) = \frac{|\{b \in B_f : b \leq v_f(x)\}|}{|B_f|}$$

. In the hard setting the upper-tail tiers are defined by fixed benign-quantile ranges, *very_low* at $0.80 \leq q_f < 0.90$, *low* at $0.90 \leq q_f < 0.97$, *medium* at $0.97 \leq q_f < 0.995$, and *high* at $q_f \geq 0.995$, with lower-tail predicates defined symmetrically. The tier names denote evidence strength, not feature magnitude: a feature may support an attack because it is unusually low.

Hard predicates are easy to inspect, but they create a performance problem. Due to their binary output, different samples fire different predicates, so the structure of the compiled circuit varies from sample to sample under DeepProbLog pruning, and a cached circuit becomes less reusable since each sample may activate a different subset of predicates, leading to unique circuits. The soft architecture replaces each hard external with a differentiable module while preserving the logical structure, so the program still contains readable predicates such as $e_sttl_high(X)$ but their truth value is supplied by a small module. The benign-quantile thresholds are first converted into feature-space cutpoints, which define intervals for the different quantiles. For interval $i = [l_i, u_i]$ the soft membership is

$$m_i(x) = \sigma(k(v_f(x) - l_i)) \cdot \sigma(k(u_i - v_f(x))),$$

where σ is the sigmoid and k a learnable steepness. Memberships are normalised with a temperature-controlled softmax,

$$s_i(x) = \frac{\exp(\log(m_i(x))/\tau)}{\sum_j \exp(\log(m_j(x))/\tau)},$$

so early training keeps uncertainty near boundaries and later training sharpens the assignment. A straight-through estimator lets the forward pass behave like a discrete tier assignment while the backward pass still receives gradients: with $h_i(x)$ the one-hot tier selected forward, the surrogate is $\tilde{s}_i(x) = h_i(x) + s_i(x) - \text{stopgrad}(s_i(x))$. Each tier has a learnable confidence $\sigma(a_i)$, and the probability that the evidence holds is

$$P(e_f(x) = true) = \sum_i \tilde{s}_i(x)\,\sigma(a_i),$$

with a_i initialised so that $\sigma(a_i)$ equals the corresponding hard-rule confidence, for example 0.90 for the strongest tier. Categorical equality is softened by comparing the encoded value to the target c as $P(feat_is(f,c,x)) = \sigma(k(0.5 - |v_f(x) - c|))$, which approaches one when the values match. The implementation uses inclusive threshold semantics, matching the greater-than-or-equal comparison used during calibration, because many benign and attack samples can sit exactly on the same quantile cutpoint and a mismatch at the boundary would make calibration and inference disagree.

The soft form changes how a failed condition is handled. Instead of pruning a proof when a threshold fails, the module returns graded support, so branches that hard predicates would prune stay active but contribute little when their evidence probability is low. This keeps the circuit structure stable across samples and lets the model learn from near-boundary samples that hard predicates would ignore, which is especially valuable in the low-data setting. The implementation keeps the distinction between a predicate's *membership*, which tier the current sample belongs to, and its *confidence*, how reliable that tier is as evidence for the rule; a sample can clearly belong to the strongest tier while the feature is still a weak predictor for the class, and the learnable confidence captures that second uncertainty. Hard execution is retained as a diagnostic mode, because if a rule does not fire its failure can be traced to one threshold or equality check, which helps validate that a rule was translated correctly from the calibration stage.

To make the mechanism concrete, consider the TCP header-length predicate $e_hdr_high(X)$ on a packet with raw value $v_f(x) = 40$ bytes. Suppose calibration placed the high-evidence interval at values at or above the benign 0.995 quantile, a cutpoint of 32 bytes. The raw value sits well inside that interval, so its soft membership $m_{high}(x) = \sigma(k(40-32)) \cdot \sigma(k(u_{high} - 40))$ is close to 1 while the lower tiers are near zero, and the straight-through estimator selects the high tier in the forward pass. With the high-tier confidence learned at $\sigma(a_{high}) = 0.80$, the evidence probability is $P(e_hdr_high(x)) = \sum_i \tilde{s}_i(x)\,\sigma(a_i) \approx 1 \times 0.80 = 0.80$. A categorical predicate behaves the same way: for a packet whose protocol-stack field encodes HTTP, the soft equality $\sigma(k(0.5 - |v_f(x) - c|))$ against the HTTP target c returns about 0.99 because the encoded values coincide. These two predicate values are the ones carried into the full proof of Section 5.8.

5.6 Query construction and neural predicates

This section explains how a single instance becomes a DeepProbLog query that uses both components. It is the bridge between the neural classifier and the symbolic predicates, and

it is the same bridge for every coupling variant. The neural classifier enters the program as a neural predicate that binds the network output to a probabilistic atom,

$$nn(multiclass_net, [X], Y, [c_1, \ldots, c_m]) :: class_pred(X, Y),$$

so that $class_pred(X, c_j)$ is an atom whose probability is the softmax mass the network assigns to class c_j.

An instance is not copied into the logic program as ground feature facts. It is passed by reference. Each sample is named by a tensor term $tensor(view, i)$, where *view* identifies the traffic representation and i is the row index in the active dataset, and the query substitutes this term for a placeholder before asking for a label:

$$classify(X, Label), \quad X = tensor(view, i).$$

Both sides of the program dereference the same handle. The neural predicate feeds the indexed tensor through the network to obtain *class_pred*, while the symbolic predicates read the raw, unscaled feature row at the same index through foreign predicates such as $feat_is$ and $feat_ge_q$. Passing the instance by reference, rather than as ground facts, is exactly what lets the neural side see a standardised vector and the symbolic side see raw values for the one instance, realising the two-representation storage of Section 5.2 inside the query. The functor in the query head selects the coupling policy: the default head is $classify/2$, but the configuration can name a different functor, so swapping the coupling architecture is a matter of pointing the query at a different definition of the head rather than rewriting the data pipeline. During training, one query is built per instance for its true label and the loss is the negative log-probability the program assigns to it; during evaluation the instance is queried once per candidate class and the class with the highest returned probability is the prediction. In both cases ProbLog grounds the query into an arithmetic circuit whose leaves are the neural atom probabilities and the symbolic evidence probabilities, and returns the probability of the queried label.

5.7 Coupling architectures

Each coupling defined in Section 4.3 is realised as a different definition of the query head, while everything else in the pipeline stays fixed. Within DeepProbLog, the symbolic-only baseline removes the neural classifier as the primary class source and tests whether the rules can label traffic by themselves, which is useful for diagnosing rule quality. This symbolic baseline is the most basic form of the DeepProbLog program, as no coupling is really needed since it only relies on a single component.

For the evaluated NIDS models, the different coupling architectures are implemented as follows.

The **hard-gate** architecture requires agreement between the neural prediction and class-specific symbolic evidence,

$$classify(X, C) :- class_pred(X, C),\ symbolic_C(X).$$

making it the strictest coupling: if either component fails, the class is not supported.

On the other hand, the **same-class boost** architecture relaxes this policy by keeping the neural path open and adding a second clause

$$classify(X,C) :- class_pred(X,C).$$
$$classify(X,C) :- class_pred(X,C),\ symbolic_C(X).$$

This leaves the neural network as the main classifier, while the symbolic component reinforces it only when the neural predictions are uncertain and clear attack signals are detected through the rules.

The **guarded** architecture suppresses benign predictions through negation-as-failure,

$$classify(X,benign) :- class_pred(X,benign),\ \backslash+ attack_signal(X),$$

This behaves similarly to the **hard-gate** architecture while allowing the guard statement to rescue attacks the classifier would otherwise label benign when a clear attack signal is detected by the symbolic component.

The **anchored** architecture splits rules by reliability, assigning roles to the symbolic evidence and promoting high-precision rules to standalone anchors with no neural agreement required,

$$classify(X,C) :- symbolic_C_anchor(X),$$
$$classify(X,C) :- class_pred(X,C),\ symbolic_C_rescue(X).$$

while weaker rules stay gated and a benign guard protects benign.

The **safety-net** architecture, also known as the OR-rescue, is the most permissive, supporting an attack when either path fires,

$$classify(X,C) :- class_pred(X,C); symbolic_C(X),$$

giving both components equal authority when it comes to classifying incoming traffic. Given that the two components are modelled as alternative proofs combined by a sum node, the rule can still raise an attack when the neural distribution collapses, making this coupling policy promising in environments polluted by noise and in adversarial experiments.

All couplings expose the same external interface: given an instance and a candidate class, the system returns a probability for the query. This shared interface is what lets the training loop and metric collector treat every variant uniformly, with the differences confined to the head definition. When a symbolic path and a neural path both support a class, DeepProbLog combines their proof probabilities through the compiled circuit, which is why adding a symbolic rule is not the same as adding a post-processing if-statement: the symbolic path becomes part of the probabilistic explanation for the class, and if it carries a learnable probability the model can adjust its influence during training.

5.8 Worked example of rule evaluation

Consider a packet classified by a hard-gate DoS rule,

$$classify(X, dos) :- class_pred(X, dos),\ symbolic_dos(X),$$

with

$$symbolic_dos(X) :- e_http(X),\ e_hdr_high(X).$$

Suppose the packet has an HTTP protocol-stack value and a TCP header length of 40 bytes. The categorical predicate $e_http(X)$ compares the encoded protocol-stack feature with the HTTP category. Suppose the soft equality module returns 0.99. The header length lies in the strongest benign-tail tier, whose learned confidence is 0.80, so $e_hdr_high(X) \approx 0.80$. The symbolic conjunction is a product,

$$P(symbolic_dos(X)) = 0.99 \times 0.80 \approx 0.79,$$

and if the neural classifier assigns

$$P(class_pred(X, dos)) = 0.70,$$

the hard-gate proof has probability $0.70 \times 0.79 \approx 0.55$. The same query is evaluated for all classes and the largest final probability is selected.

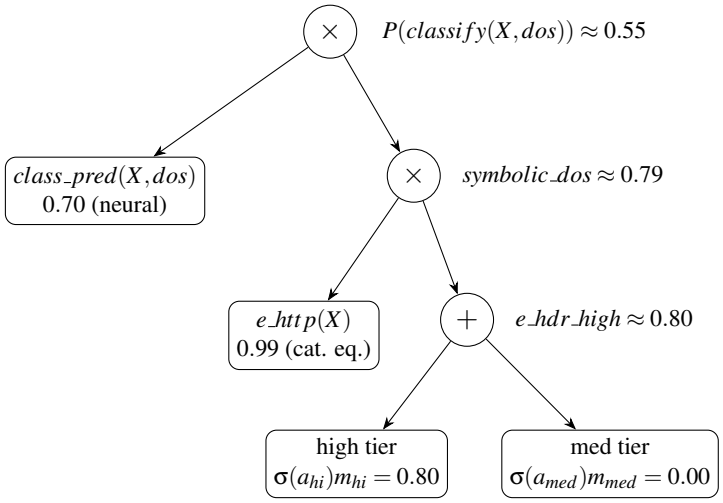


Figure 5.1: Arithmetic circuit for the worked DoS example.

Figure 5.1 shows the corresponding arithmetic circuit. Product nodes implement conjunctions, the sum node aggregates the soft evidence tiers, and the leaves carry neural probabilities or learned tier confidences. Because each value is attached to a named predicate, the same structure that produces the prediction also explains it, with no separate post-hoc model: a rule body with three evidence predicates is a product over three probabilities,

alternative bodies become several proof paths combined by sum nodes, and a guard or disqualifier introduces negated evidence. During explanation the implementation records the winning proof path, so an alert refers to predicates and probabilities that actually participated in the class query.

5.9 Hybrid packet-flow execution

The packet-only and flow-only systems consume one representation at a time. The hybrid system must synchronise two representations of the same traffic from a packet-based input stream: each packet is classified immediately by the packet path and also used to update the state of its flow. Flows are keyed by the 5-tuple, and as packets arrive the implementation updates packet and byte counts, direction-specific summaries, inter-arrival-time statistics, flag counts, and protocol fields. Once a flow has enough packets to produce a valid feature vector, the flow path becomes available. The implementation supports two ways to obtain the flow vector behind the same interface. In online assembly, the flow features are computed incrementally from the packet stream, which is the deployment-like setting; in offline matching, a packet is associated with an existing labelled flow record through its 5-tuple and timestamp, which is used for controlled evaluation because public datasets often provide labelled flows but not the complete packet stream needed to reconstruct every feature exactly. If a packet has no available flow representation, the flow tensor is marked missing and a small detector returns a uniform flow-side distribution. The neutral distribution is uniform rather than a benign prior: a benign prior would make missing flow evidence support benign and hide packet-level attack evidence early in a flow, whereas a uniform distribution says only that the flow side is unavailable and should not favour any class, leaving the packet path responsible for early decisions.

While both offline matching and online assembly are implemented, the experiments conducted in Chapter 6 use online assembly for the hybrid system to simulate a deployment scenario and highlight the impact of increasing training data and the resulting precision of the flow reconstruction on the results.

5.10 Batched arithmetic-circuit evaluation

A naive DeepProbLog evaluation would compile or evaluate a separate circuit per query, which is too slow for test sets of tens of thousands of instances queried over several classes each. The implementation avoids this by separating circuit structure from leaf values. The program is compiled once against placeholder terms for the current packet and, in the hybrid case, the current flow; at evaluation time the placeholders are substituted with the tensors for the current batch, so the logical structure is unchanged and the compiled circuit is reused. The soft-rule architecture is what makes this effective: because threshold predicates return probabilities rather than pruning branches, the circuit structure is stable across samples, and the same circuit can be evaluated with different leaf values. The batched evaluator then represents each leaf as a vector of probabilities, one per sample, and walks the circuit once for the whole batch, with product nodes multiplying elementwise, sum nodes adding, and

negation nodes computing one minus the vector. A per-instance mode is kept for debugging; when batched and per-instance results disagree it is used to localise the discrepancy to broadcasting, class ordering, predicate evaluation, or circuit reuse. The evaluator records average inference time per sample, steady-state time after warm-up, peak memory, and circuit size, which feed the complexity analysis in the results chapter.

5.11 Implementation boundaries

Several choices are intentionally left outside the implementation. The system does not implement payload inspection, online model updating, automatic response actions, or a full analyst interface; its output is designed for experimental evaluation and explanation analysis, not operational deployment. When a neuro-symbolic model predicts an attack, the recorded proof path shows whether the decision came from the neural path, symbolic corroboration, a benign guard, or a disqualified rule, which makes it possible to compare two models with similar scores but different reasoning, and to identify which predicate is responsible when a coupling raises more false positives than the baseline. The implementation also does not claim that the learned symbolic probabilities are universal rule reliabilities: a learned confidence is tied to the dataset, taxonomy, split, and coupling of its experiment and should be re-estimated before the rule is used elsewhere. These boundaries keep the work focused on its question, whether DeepProbLog can combine neural and symbolic evidence for NIDS tasks under controlled conditions, while still keeping deployment constraints visible by measuring inference cost, handling missing flow evidence, avoiding payload features, and preserving rule-level explanations.

Chapter 6

Experiments and Results

This chapter reports the experiments and their results. Its structure follows the three research questions stated in the introduction. Section 6.1 fixes the settings that every later result depends on: the datasets, the training-size sweep, the replicate protocol, the metrics, and the execution environment. The three result sections then take one research question each. Section 6.2.1 measures detection performance and how it changes with the amount of labelled training data. Section 6.2.2 asks whether the symbolic rules help the models generalise to a different dataset and hold up under perturbed inputs. Section 6.2.3 examines the explanations the symbolic layer produces and compares them with post-hoc attributions on the neural baseline. A final section collects the ablations that isolate the effect of individual design choices. In every result section the neuro-symbolic models are compared against the neural baseline, and where it is informative against a symbolic-only model and a signature-based detector.

6.1 Experimental settings

All experiments share the same setup: each model is trained on a small per-class subset of one dataset, re-initialised from scratch for every replicate, and evaluated on a fixed held-out test set with macro-F1 and attack false-positive rate as the headline metrics. The split strategy, training protocol, and metric definitions are defined in Section 4.5.1, and the datasets in Section 5.1; this section records the choices the result tables depend on and the conventions used to report them, so the results can be read as a self-contained reference.

The result tables depend on one convention worth restating here. Training is conducted on subsets parametrised by $k \in \{5, 10, 25, 50, 500\}$, where each attack class contributes k samples and benign is sampled as a fixed multiple of the total attack count, so a subset is balanced between benign and general attack mass while k controls the attack evidence the model sees per class. For every k, five independent training subsets are drawn and the model is re-initialised and trained from scratch on each, while the validation and test sets stay fixed across replicates and across all values of k.

6.1.1 Baselines

The neuro-symbolic models are compared against three reference baselines, each chosen to isolate a different part of the system.

Neural Baseline The neural baseline is the multilayer perceptron described in Section 5.3, trained on the same features as the neuro-symbolic models. This baseline shares the neural component with the neuro-symbolic models and only drops the symbolic query program, so the gap between the two isolates the symbolic layer and the DeepProbLog integration.

Symbolic Baseline The symbolic-only baseline is the opposite endpoint: it removes the neural classifier and labels traffic from the mined rules alone, which measures the quality of the rules on their own and, by comparison with the full models, the contribution the neural component adds.

Signature Baseline The signature baseline is external to the learned models. It runs a rule-matching engine over the same traffic and maps its alerts into the class taxonomy, representing a traditional signature IDS. It has no training step, so comparing against it tests whether the proposed models add value over an off-the-shelf engine rather than only over a neural classifier. It is built with Suricata and the Emerging Threats ruleset; a ruleset tuned to these specific captures could raise its scores, but the stock configuration is kept as a representative off-the-shelf point of comparison.

Aside from the three reference baselines, the neuro-symbolic models are also compared to state-of-the-art (SOTA) baselines. The SOTA baselines are models from recent literature that are widely recognised in the domain or perform well on the datasets used in this thesis.

The SOTA baselines are organised by traffic view, matching the three levels at which the neuro-symbolic system is evaluated. Each is run on the same splits, training sizes and replicate protocol as the corresponding neuro-symbolic model, so the only difference is the detector. Where no public implementation is released, the model is reimplemented faithfully from its paper.

Kitsune (flow level) Kitsune [35] is a widely cited unsupervised detector built from an ensemble of autoencoders (KitNET): it trains on benign traffic only and flags a sample when its reconstruction error rises above a threshold. Its native front end, AfterImage, derives around one hundred incremental statistics from raw packet captures. The faithful pipeline is reproduced here: AfterImage extracts features from the original CIC-IDS2017 captures, carved so that the packets correspond to exactly the same flows used by every other model, KitNET is trained on the benign packets, and its per-packet anomaly scores are aggregated to a flow decision.

KnowML (flow level) KnowML [19] is a recent network intrusion detector whose contribution is a feature extractor that produces a richer set of per-connection statistics from raw captures. Two variants are evaluated, both on the same carved CIC-IDS2017 captures. The first pairs the KnowML features with the same unsupervised KitNET detector used for Kitsune; the second pairs them with the supervised multilayer perceptron of the neural baseline. The two variants separate the contribution of the feature extractor from that of the detector by coupling the same feature extraction technique to different detection heads.

CNN-WDLSTM (packet level) CNN-WDLSTM [22] is a hybrid deep model in which a convolutional network extracts features and a weight-dropped LSTM, which applies dropout to the recurrent connections to resist overfitting, models the dependencies among them before a softmax classifier. No official implementation is released, so it is reimplemented and run per packet on the same packet-header features as the packet-level neuro-symbolic models, with no flow assembly and no payload bytes. It is worth noting that the original paper evaluates it at the flow level on statistical flow features, while this thesis adapts it to the packet level.

SPHBI (packet level) SPHBI [13] represents a single packet by packing the relevant fields of its raw TCP/IP header, bit by bit, into a small fixed-size binary image, which a lightweight convolutional network then classifies, avoiding any explicit feature engineering. It is reimplemented following the paper's architecture and run on the header fields available in the packet view, with the few header bytes absent from the dataset zero-filled.

DST-IDS (hybrid level) DST-IDS [39] fuses a packet-level and a flow-level classifier using Dempster–Shafer evidence theory rather than a simple average: each classifier's output is turned into a body of evidence that reserves an explicit mass for uncertainty, set from the classifier's own error rate, and the two are combined with Dempster's rule so that agreement is reinforced and a confident branch dominates an uncertain one. The fusion rule, which is presented as model-agnostic, transferable, and applicable on any pair of classifiers, is transplanted onto the packet and flow classifiers used by the hybrid neuro-symbolic system, so the comparison isolates the fusion strategy.

6.1.2 Execution environment.

All evaluations were conducted on a **MacBook Air, Apple M3 Chip, 16 GB unified memory**, running **macOS Sonoma 14.6**. The code for all the experiments was executed in the same environment, ensuring consistency across the results. The specific versions of the software libraries used for the implementation and evaluation of the neuro-symbolic models, as well as the baselines, are the following:

- Python 3.13.11
- PyTorch 2.9.1
- DeepProbLog 2.0.6

- ProbLog 2.2.9

Inference time and memory are measured on this machine and reported with the cost results. The numbers should be read relative to this environment and hardware.

6.2 Experiments

This section is dedicated to the experiments conducted in order to answer the three research questions proposed in the introduction. Section 6.2.1 investigates the impact of employing a neuro-symbolic approach on the performance of a network intrusion detection system (NIDS), as well as the efficiency of the developed system in terms of the amount of labelled data required during training in order to achieve this performance. Section 6.2.2 evaluates the generalisation capabilities of the developed system by testing it on a different dataset and under perturbed inputs. Lastly, Section 6.2.3 examines the explanations produced by the symbolic layer and compares them with post-hoc attributions on the neural baseline.

6.2.1 Detection performance & Efficiency

This section is dedicated to answering the first research question proposed in Chapter 1, which investigates the impact of employing a neuro-symbolic approach on the performance of a NIDS, as well as the efficiency of the developed system in terms of the amount of labelled data required during training in order to achieve this performance.

Binary attack-specific flow experiments

Before the full multiclass system is evaluated, a simpler question has to be settled: do the mined rules carry any real detection signal, or do they only become useful once a neural classifier is already doing most of the work? The binary setting isolates this question. Each attack family is evaluated as its own attack-versus-benign problem, so the rule set for that family is judged without competition from the other attack classes.

For each family, a reduced version of the experiment protocol is conducted: three systems are trained and compared on the same training-size sweep, $k \in \{5, 10, 25, 50\}$, with three replicates per value. The neural baseline is trained and compared against two neuro-symbolic detectors that share the **anchored** coupling architecture described in Section 5.7. They differ in how the candidate rule generation expresses numerical evidence. The **anchored** variant turns each numerical feature into a single crisp threshold placed at a benign quantile. The **tiered** variant splits a numerical feature into four ordered bands, from very low to high, and lets the DeepProbLog program learn a separate weight for each band, so it can settle on the part of the range that actually separates the attack instead of committing to one cut in advance.

The test split used remains faithful to the original dataset's distribution, which leaves most families as rare-attack problems. The full per-family sweep is reported in Appendix C.1; the figures here keep only the readings that answer the two questions of this section.

Figure 6.1 gives the result at $k = 50$. On the ten families where the pipeline produced at least one rule, a neuro-symbolic detector reaches the higher or equal macro-F1 on eight,

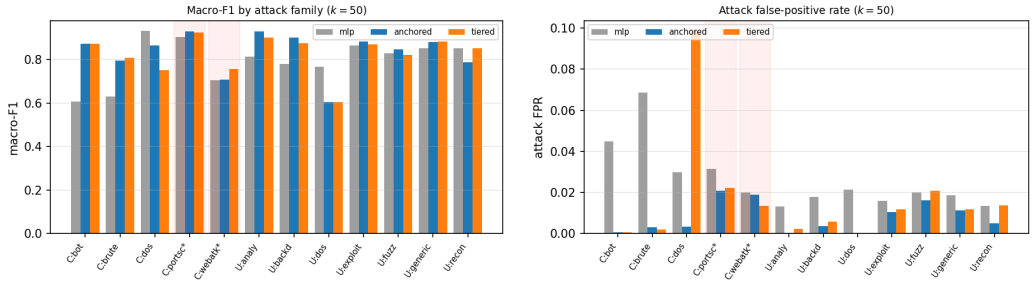


Figure 6.1: Binary attack-versus-benign flow results at $k = 50$ on CIC-IDS2017 (C:) and UNSW-NB15 (U:). Shaded families have no rule and fall back to the neural path.

and the second panel shows why. The neural baseline keeps a high detection rate on every family, but it leaks benign flows into the attack class, and under the rare-attack prior those false alarms cut its precision sharply. The mined rules fire rarely and almost only on real attacks, so they hold the attack false-positive rate close to zero. Bot is the clearest case: the baseline reaches a macro-F1 of 0.609 at a false-positive rate of 0.045, while the anchored detector reaches 0.873 on the destination-port rule alone with a false-positive rate of 0.001. The protocol anchors on UNSW-NB15 behave the same way, lifting analysis from 0.814 to 0.931 and backdoor from 0.779 to 0.903. The candidate rule generation therefore produces family-specific evidence with real classification power, and that evidence is strong enough to be useful from the first handful of training samples.

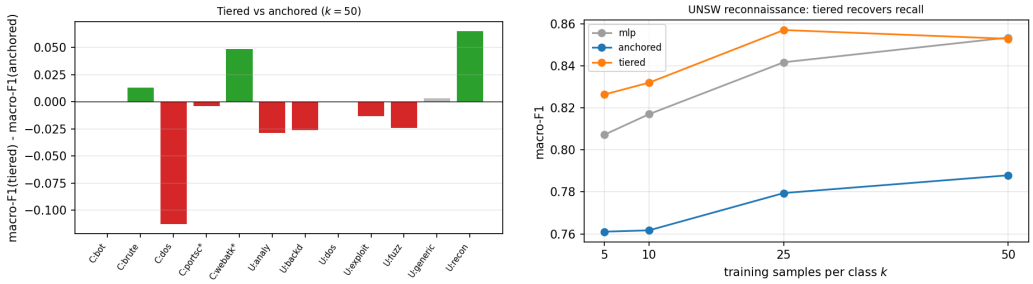


Figure 6.2: Tiered versus anchored representation. Left: per-family macro-F1 difference at $k = 50$, Right: the UNSW reconnaissance sweep.

As Figure 6.2 shows, there is no clear-cut winner between the anchored and tiered representations. The tiered ladder is revealed to have the upper hand on three attack families, while the anchored representation with a single threshold on features wins on six. Neither representation dominates, so the choice can be made per family. The tiered representation is more flexible, but it also has more parameters to learn, so it can overfit when the training size is small. The anchored representation is simpler and more robust, but it can miss attacks that fall outside the single threshold. The right panel of Figure 6.2 shows an example where the tiered ladder recovers the recall that a single crisp threshold misses: the reconnaissance

family on UNSW-NB15 has a bimodal distribution on its packet-per-flow feature, and the anchored detector only catches the low mode while the tiered ladder captures both.

Across the full sweep the anchored detector is the best system on 25 of the 40 family-and-budget cells that carry rules, against 8 for the neural baseline and 7 for the tiered variant. The binary experiment supports a measured claim. The mined rules carry genuine family-specific signal, they suppress false alarms well enough to beat a neural baseline on most families with very little data, and the tiered representation extends that reach to attacks defined by a band rather than a tail. They are not uniformly safe as a drop-in attack path, which is why the multiclass experiments that follow keep the open, same-class boost, guarded, and hard-gate couplings separate instead of treating the symbolic layer as a single fixed add-on.

CIC-IDS2017 Original flow multiclass results

The full multiclass system is evaluated first on the original CIC-IDS2017 flow dataset, which is the setting most directly comparable to prior work. Table C.2 reports every coupling against the three baselines across the training-size sweep, Figure 6.3 visualizes the results as a macro-F1 curve and an attack false-positive heatmap, and Figure 6.4 shows the detection-rate and exact-attack metrics as bubble grids.

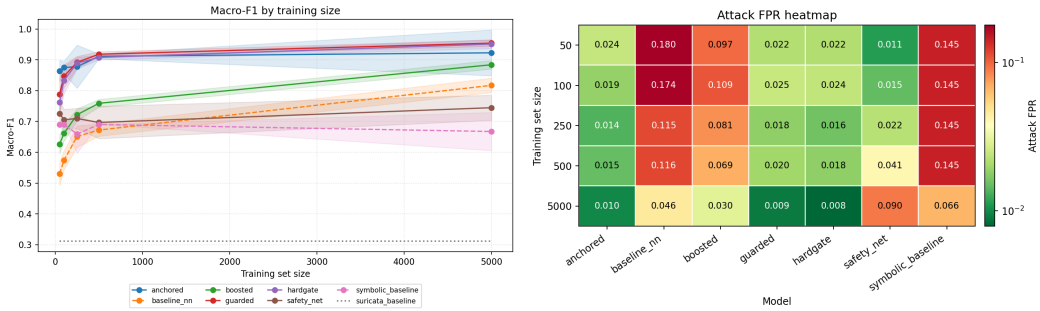


Figure 6.3: Original CIC-IDS2017 flow multiclass results across training size: macro-F1 (left) and attack false-positive heatmap (right).

The results in Figure 6.3 reveal that the headline is the gap at small k. With only fifty samples per class the neural baseline reaches a macro-F1 of 0.672, while the guarded coupling strategy reaches 0.903 and the anchored and hard-gate coupling strategies are close behind.

The gap stems from the data each component needs: at low k the neural classifier sees few samples per family, while the mined rules already fire on stable benign-calibrated evidence and only use the training samples to adjust their weights. As k grows the neural baseline reaches 0.817 while the best couplings hold near 0.95, and the symbolic baseline stays flat across sizes, so neither component alone matches the coupled system. The guarded and anchored couplings already pass the baseline's maximal score at $k = 10$, a **50x** training-volume efficiency.

The attack false-positive rate shows a second benefit: most couplings stay near 0.02 and drop to 0.008 for the hard-gate, while the neural baseline starts at 0.18 at $k = 5$ and

only reaches 0.04. Under the gated policies a sample must match a symbolic rule before being labelled an attack, which gates suspicious-looking benign flows the neural component would otherwise flag.

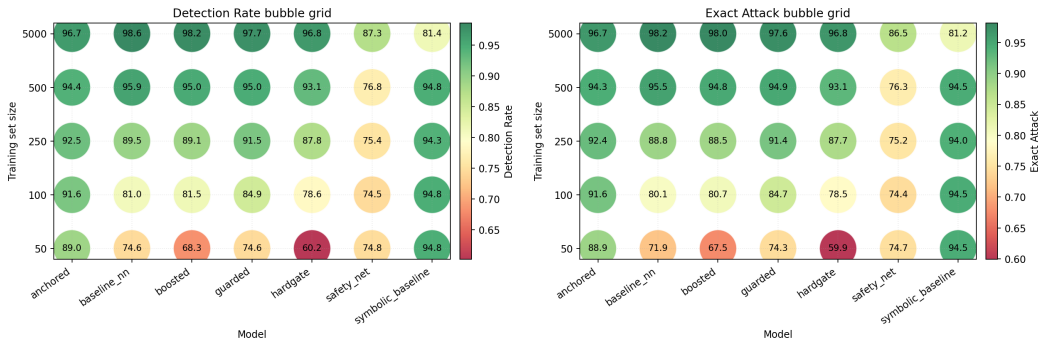


Figure 6.4: Original CIC-IDS2017 flow multiclass across training size: detection-rate (left) and exact-attack (right) bubble grids.

Figure 6.4 shows the detection-rate and exact-attack bubble grids. The symbolic layer recovers and correctly classifies most attacks even at low k, where the neural baseline struggles; the gated couplings keep this recovery while holding a low false-positive rate, whereas an ungated symbolic path over-flags benign traffic through its broad rescue rules. At $k = 500$ the neural classifier catches up on detection and exact-attack rate and slightly surpasses the models, and the symbolic baseline dips there, consistent with overfitting on its learned weights. The neuro-symbolic models still hold the higher macro-F1 at $k = 500$.

The couplings separate into a clear order. The guarded and hard-gate couplings are the strongest and most stable; the anchored coupling matches them at low k and trails slightly at high k through its larger variance. The same-class boost coupling sits between the baseline and the gated couplings, since it leaves the neural path open and only adds symbolic mass when a rule matches. The safety-net coupling is weakest on clean traffic, its disjunctive rule capping macro-F1 near 0.79. The symbolic-only and Suricata baselines confirm that neither component alone suffices, with the signature engine reaching only 0.311 on these untuned captures.

Beyond the in-house baselines, the flow detectors are compared against two published state-of-the-art systems, **Kitsune** [35] and **KnowML** [19]; Section 6.1.1 describes how each is implemented.

The flow comparison is conducted on the original CIC-IDS2017 flow data of Section 6.2.1 at the largest training size, $k = 500$, on the binary attack-versus-benign axis. The unsupervised detectors cannot attribute an attack to a family, so the six-class neuro-symbolic outputs are collapsed to attack-versus-benign and every system is scored identically; detection rate is attack recall and attack-FPR is the fraction of benign flows flagged as attacks.

Each detector is given a validation-tuned decision threshold, reporting the unsupervised systems at their most favourable. Kitsune appears in two configurations — its faithful AfterImage pipeline on the raw captures, carved to the same flows, and a flow-feature variant fed the same features as every other model — and KnowML's feature extractor is paired both with the unsupervised KitNET and with the supervised neural-baseline MLP, which isolates the contribution of the features from that of the detector.

Table 6.1: State-of-the-art flow comparison on CIC-IDS2017 at $k = 500$, binary attack-versus-benign, mean ± standard deviation over five replicates. The six-class neuro-symbolic outputs are collapsed to the attack-versus-benign axis on which the state-of-the-art detectors operate. Best per column in bold.

System	Macro-F1	Detection rate	Attack-FPR
Neural baseline	0.945±.010	**0.986**±.012	0.046±.011
Symbolic baseline	0.867±.014	0.814±.070	0.066±.040
Anchored	0.978±.005	0.967±.008	0.010±.002
Same-class boost	0.961±.004	0.982±.015	0.030±.001
Guarded	**0.982**±.002	0.977±.002	0.009±.001
Hard-gate	0.981±.002	0.968±.010	**0.008**±.003
Safety-net	0.865±.008	0.873±.053	0.090±.027
KnowML + MLP (supervised)	0.927±.063	0.909	0.054
KnowML + KitNET (unsupervised)	0.811±.014	0.778	0.155
Kitsune + AfterImage	0.792±.026	0.743	0.129
Kitsune + flow features	0.721±.006	0.402±.017	0.030±.007

Table 6.1 collects the comparison. The neuro-symbolic couplings lead: the guarded coupling reaches a binary macro-F1 of 0.982 at an attack-FPR of 0.009, with hard-gate and anchored within a point. The supervised models follow — the neural baseline at 0.945 and KnowML+MLP at 0.927 — while every unsupervised detector sits far below, from KnowML+KitNET at 0.811 down to the flow-feature Kitsune at 0.721, none clearing 0.82 and the best of them recovering only 0.778 of attacks, compared with above 0.96 for the gated couplings. Two readings follow. The faithful AfterImage pipeline (0.792) beats the flow-feature adaptation (0.721), so feeding Kitsune its native packet statistics helps and the earlier adaptation was, if anything, conservative. More tellingly, the same KnowML features reach 0.927 with a supervised head but only 0.811 with KitNET.

Standardised NF-v3 CSE-CIC-IDS2018 results

The standardised NetFlow-v3 representation of CSE-CIC-IDS2018 lets the same system be evaluated at the packet, flow, and hybrid levels on one feature schema, whose flow features can be approximately rebuilt from packet headers. This makes the packet view, which classifies in real time without waiting for flow assembly, directly comparable to the flow view. The full packet-header feature list is given in Appendix B.1, Table B.1. The **webattack** family is dropped from all three views because its PCAPs are corrupted, and **infiltration** is dropped as a multi-stage attack outside the rule-construction scope.

The hybrid level is a fusion comparison between (i) a neural late-fusion baseline, (ii) a symbolic-only model, (iii) a single neuro-symbolic hybrid model, since the coupling policy axis is a single-view construct that does not transfer to a hybrid system of the two views. Its online assembler groups packets into flows, computes flow features once enough packets arrive, and consistently updates these features with new packet arrivals.

Packet level analysis The symbolic layer gives its largest improvement at the packet level, where a single packet carries the least evidence. Figure 6.5 shows the neural baseline at macro-F1 0.632 and FPR 0.075 at $k = 50$, while the guarded coupling reaches 0.924 and cuts the FPR roughly ninefold to 0.008. The symbolic baseline also exceeds the neural baseline at every size while trailing the guarded coupling, so the rules carry real packet-level signal that the network is slower to learn on its own at low data.

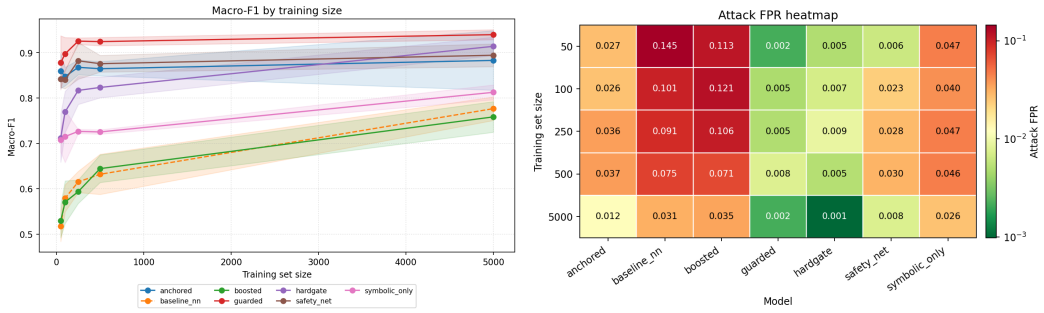


Figure 6.5: Standardised NF-v3 packet level: macro-F1 by training size (left) and attack false-positive heatmap (right).

The packet view is also compared against two state-of-the-art (SOTA) models. The first packet-level SOTA is **CNN-WDLSTM** [22], and the second model is **SPHBI** [13]. Their implementation is detailed in Section 6.1.1.

Both are supervised multiclass classifiers, evaluated on the same standardised NF-v3 splits, training-size sweep, and five replicates as above. As data-hungry deep models, they are trained for up to fifty epochs like the other baselines, while SPHBI follows the longer 100-epoch schedule proposed in the original paper. The neuro-symbolic systems stay at their converged ten-epoch budget.

Table 6.2 reports both packet-level baselines against the guarded and anchored couplings and the neural baseline, with macro-F1, detection rate and attack false-positive rate at every size. CNN-WDLSTM is a competent state of the art model: it overtakes the plain neural baseline at almost every size, reaching 0.818 against 0.776 at $k = 500$. Yet it is data-hungry seeing that it only scores 0.580 at $k = 5$ even after fifty epochs, and the guarded coupling exceeds it at every training size, by the widest margin in the low-data regime with 0.878 against 0.580 at $k = 5$, and 0.924 against 0.742 at $k = 50$, and at a far lower false-positive rate (0.002 against 0.109 at $k = 5$).

SPHBI is the header-only exception, and unlike every other model in this chapter it reads no payload, no flow assembly and no handcrafted statistics: it packs the TCP/IP header of

Table 6.2: Packet-level state-of-the-art comparison on standardised NF-v3 across the training-size sweep over five replicates. Each size reports macro-F1 (MF1) as mean ± standard deviation, with detection rate (DR) and attack false-positive rate (FPR) as means. CNN-WDLSTM is the best of up to fifty epochs and SPHBI is trained to convergence; the neuro-symbolic couplings and the neural baseline are at ten epochs. For each metric the best system per k is in bold (highest MF1 and DR, lowest FPR).

	k=5			k=10			k=25			k=50			k=500		
System	MF1	DR	FPR	MF1	DR	FPR	MF1	DR	FPR	MF1	DR	FPR	MF1	DR	FPR
Neural baseline	0.518±.034	0.898	0.145	0.579±.030	0.930	0.101	0.615±.023	0.955	0.091	0.632±.044	0.948	0.075	0.776±.026	0.992	0.031
Guarded (NeSy)	**0.878**±.059	0.894	**0.002**	**0.897**±.036	0.915	**0.005**	**0.925**±.007	0.933	**0.005**	**0.924**±.009	0.947	**0.008**	0.940±.011	0.951	0.002
Anchored (NeSy)	0.860±.038	0.931	0.027	0.847±.020	0.946	0.026	0.868±.016	0.973	0.036	0.864±.016	0.973	0.037	0.883±.066	0.992	0.012
CNN-WDLSTM (SOTA)	0.580±.062	0.945	0.109	0.598±.045	0.950	0.097	0.681±.016	0.970	0.056	0.742±.030	0.970	0.034	0.818±.016	0.986	0.008
SPHBI (SOTA)	0.430±.023	**0.947**	0.113	0.551±.059	**0.973**	0.091	0.678±.097	**0.993**	0.041	0.896±.019	**0.992**	0.012	**0.985**±.004	**0.998**	**0.001**

a single packet bit-for-bit into a 12×12 binary image classified by a small convolutional network, the closest of all baselines to the raw header features the couplings read. It is the one place a published baseline overtakes the neuro-symbolic system: at $k = 500$ it reaches a macro-F1 of 0.985, with a detection rate of 0.998 and an attack false-positive rate of 0.001, above the guarded coupling at 0.940 and CNN-WDLSTM at 0.818. The ranking inverts as labels grow scarce, because a convolutional model has many filters to fit over the header image and a handful of packets per class is not enough: at $k = 50$ SPHBI (0.896) already trails the guarded coupling (0.924), and at $k = 5$ it collapses to 0.430, below even the plain neural baseline at 0.518. The crossover sits between $k = 50$ and $k = 500$. This is the chapter's data-efficiency argument seen from the other side: a strong header-only CNN can beat the neuro-symbolic system, but only in the data-rich regime that operational low-label deployments rarely enjoy.

Flow level analysis The flow level shows a smaller gap, because the standardised flow features already let the neural baseline reach 0.881 at $k = 50$; the gated couplings edge ahead to 0.895 and cut the false-positive rate from 0.014 to 0.003. Figure 6.6 shows the gated couplings just above the baseline on macro-F1, converging near 0.925 at $k = 500$, and lowest on benign error at every size. The flow features give the neural classifier enough to learn from, so the symbolic gain here is mainly the consistent reduction in false alarms, and the baseline catches up by $k = 500$.

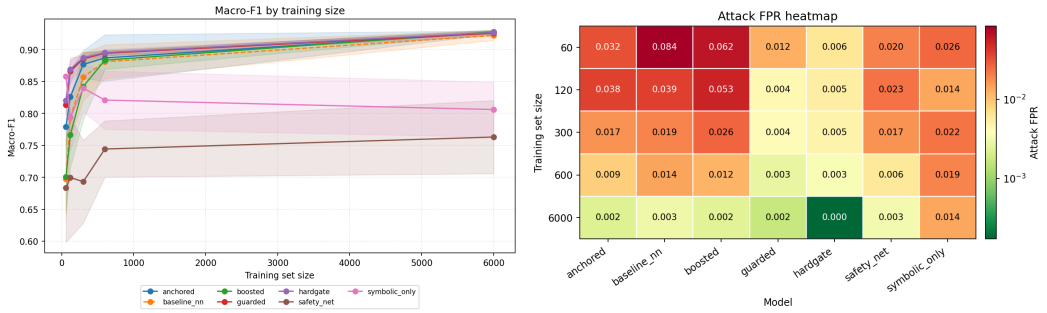


Figure 6.6: Standardised NF-v3 flow level: macro-F1 by training size (left) and attack false-positive heatmap (right).

Hybrid level analysis The hybrid level gives the largest low-data gap: Figure 6.7 shows the neural late-fusion baseline at 0.355 against the neuro-symbolic hybrid's 0.734 at $k = 5$, and 0.620 against 0.869 at $k = 50$. The cross-view rules carry the model when neither single-view network has enough data to fuse, and the hybrid ceiling at $k = 500$, 0.971, exceeds either single view. Part of the rise with k also comes from more accurate flow construction, since the online assembler refines each flow's features as more packets arrive.

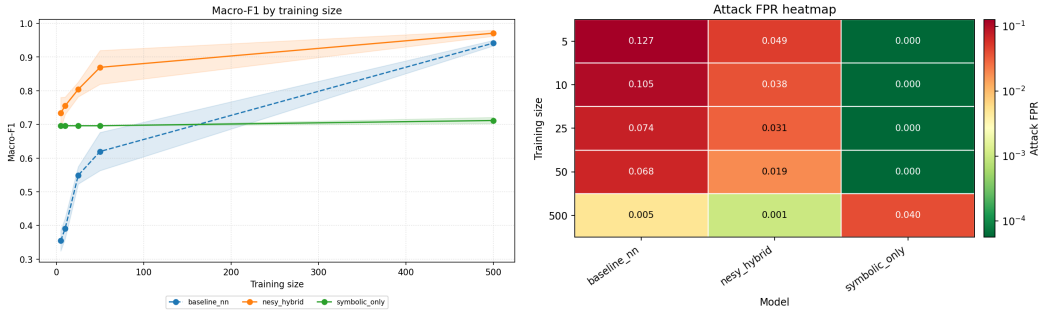


Figure 6.7: Standardised NF-v3 hybrid level: macro-F1 by training size (left) and attack false-positive heatmap (right).

The hybrid-level SOTA is **DST-IDS** [39], which fuses an independent packet detector and flow detector at the decision level with Dempster-Shafer evidence theory as described in Section 6.1.1. We transplant this fusion mechanism — the paper's actual contribution, which the authors explicitly state may sit on top of any packet and flow models — onto the same packet and flow classifiers used by the neuro-symbolic hybrid, so that only the fusion operator changes; we do not reproduce their original Random-Forest-plus-payload-CNN instantiation or their early-detection protocol, which target a different setting.

Table 6.3 reports the result. DST-IDS is a markedly stronger fusion than the naive product baseline at low data (0.520 against 0.405 at $k = 5$; 0.724 against 0.665 at $k = 25$), so the neuro-symbolic hybrid is again measured against a credible state of the art. Even so, the neuro-symbolic hybrid exceeds DST-IDS at every training size — 0.734 against 0.520

at $k = 5$, 0.869 against 0.752 at $k = 50$, and 0.971 against 0.927 at $k = 500$ — and at a lower false-positive rate. The cross-view symbolic rules carry the hybrid model when neither single-view network has enough data to fuse, which a purely statistical fusion rule such as Dempster-Shafer cannot replicate.

Table 6.3: Hybrid-level state-of-the-art comparison on standardised NF-v3 across the training-size sweep over five replicates. Each size reports macro-F1 (MF1) as mean ± standard deviation, with detection rate (DR) and attack false-positive rate (FPR) as means. DST-IDS and the neural product fusion are the best of up to fifty epochs; the neuro-symbolic hybrid is at ten epochs. For each metric the best system per k is in bold (highest MF1 and DR, lowest FPR).

	k=5			k=10			k=25			k=50			k=500		
System	MF1	DR	FPR	MF1	DR	FPR	MF1	DR	FPR	MF1	DR	FPR	MF1	DR	FPR
NeSy hybrid	**0.734**$\pm$.047	**0.944**	**0.049**	**0.755**$\pm$.026	**0.957**	**0.038**	**0.804**$\pm$.022	**0.968**	**0.031**	**0.869**$\pm$.050	**0.977**	0.019	**0.971**$\pm$.009	0.980	**0.001**
DST-IDS (SOTA)	0.520$\pm$.062	0.800	0.125	0.624$\pm$.077	0.803	0.071	0.724$\pm$.044	0.965	0.050	0.752$\pm$.100	0.805	0.037	0.927$\pm$.016	**0.993**	0.011
Neural product	0.405$\pm$.040	0.287	0.114	0.446$\pm$.046	0.308	0.088	0.665$\pm$.059	0.578	0.045	0.767$\pm$.024	0.546	**0.017**	0.949$\pm$.007	**0.993**	0.003

View summary and comparison The packet view classifies fast but its granular header features miss multi-packet signatures such as (D)DoS, so it has the lowest ceiling and benefits most from the symbolic layer. The flow view adds aggregated features yet does not improve on the packet view, because the standardised NF-v3 schema strips much of the flow detail the original dataset carries. The hybrid view combines the multi-packet patterns visible in flows with the granularity of packet headers, and reaches the highest macro-F1 of the three. The hybrid system is likewise compared against a published fusion state of the art.

Table 6.4: Standardised NetFlow-v3 CSE-CIC-IDS2018: cross-view comparison of the strongest neuro-symbolic system per view across all training sizes k, mean over five replicates.

	k=5			k=10			k=25			k=50			k=500		
view	MF1	DR	FPR	MF1	DR	FPR	MF1	DR	FPR	MF1	DR	FPR	MF1	DR	FPR
packet (guarded)	**0.878**	0.894	**0.002**	**0.897**	0.915	**0.005**	**0.925**	0.933	**0.005**	**0.924**	0.947	0.008	0.940	0.951	0.002
flow (hard-gate)	0.820	0.896	0.006	0.869	0.927	**0.005**	0.887	0.962	**0.005**	0.895	0.968	**0.003**	0.927	**0.986**	**0.000**
hybrid (NeSy)	0.734	**0.944**	0.049	0.755	**0.957**	0.038	0.804	**0.968**	0.031	0.869	**0.977**	0.019	**0.971**	0.980	0.001

The three standardised views are drawn together for direct comparison. Table 6.4 takes the strongest neuro-symbolic system in each view (the guarded coupling for packet, the hard-gate coupling for flow, and the neuro-symbolic hybrid for the fusion view) and reports macro-F1, detection rate, and attack false-positive rate at every training size. The packet view leads on macro-F1 across the data-scarce regime, from 0.878 at $k = 5$ to 0.924 at $k = 50$, and holds the lowest false-positive rate at the smaller budgets; the flow view tracks just behind and tightens to the lowest false-positive rate once data grows; and the hybrid view, weakest on macro-F1 when labels are scarce but strongest on detection rate throughout, overtakes on macro-F1 once the budget is large, reaching 0.971 at $k = 500$.

A limitation: class overlap on UNSW-NB15

The approach proves consistent across datasets and views, with one exception: the original UNSW-NB15 flow dataset exposes a limitation rooted in the data rather than the method. Every system stays in the same low macro-F1 band: the neural baseline reaches only 0.543 at $k = 50$ and 0.600 at $k = 500$, the mined symbolic layer hovers near 0.38, and the couplings track the baseline rather than lifting it. The detection rate is high across all systems and training sizes while the exact-attack rate stays low, capping at 64.3% for the anchored coupling: the systems detect the attacks but cannot name the family. The cause is heavy class overlap in the flow feature space: analysis, backdoor, exploits and the generic category share large regions, so a rule precise for one fires for the others and the per-family confusion is systematic; the evidence needed to separate the families is simply not present in the features, which neither the rules nor the neural classifier can manufacture. The same low band recurs on the standardised NetFlow-v3 version across the packet, flow and hybrid views, for the same reason. The full per-coupling sweep, the macro-F1 and false-positive figures, and the detection-rate and exact-attack bubble grids are collected in Appendix C.6 for the original flow data and Appendix C.7 for the standardised NetFlow-v3 version.

Summary

The first research question aims to measure the impact of neuro-symbolic reasoning on network intrusion detection in terms of performance as well as training efficiency.

RQ.1: Performance & Efficiency

Performance Neuro-symbolic reasoning improves network intrusion detection performance on datasets where the attack families are separable, by using symbolic rules mined from the data to guide the neural classifier at training and inference time.

Efficiency Neuro-symbolic reasoning is more efficient than purely neural approaches, achieving higher macro-F1 scores with fewer training samples.

Limitations The symbolic layer is effective when the attack families are separable in the feature space, but it cannot resolve confusion in the data itself, such as family overlap, as seen in the UNSW-NB15 dataset.

6.2.2 Generalisability & Robustness

The second research question asks whether neuro-symbolic reasoning allows better generalisability of the network intrusion model across datasets and network topologies, and whether the symbolic component increases the model's robustness and resistance to noise and adversarial attacks that manipulate the input features. This section answers that question through experiments that test the model's behaviour in a cross-dataset setting and its robustness to noise and adversarial attacks.

Portability and fine-tuning experiments

The portability question asks whether the symbolic rules carry detection knowledge that survives a move to a different dataset and network topology, or whether they only encode artefacts of the dataset they were mined on. Conversely, the experiment also validates whether rules mined on the destination network can help a neural system trained on a previous dataset adapt to the new environment. The standardised NetFlow-v3 representation makes the questions testable, because every dataset in that family is described by the same feature columns. A rule written for one dataset can therefore be grounded on another without any change to its predicates. Schema compatibility, however, is not the same as distribution compatibility, and the experiments in this section separate the two.

Datasets The experiment consists of a cross-dataset transfer, where a model is trained on one dataset and then evaluated on another. The datasets used are the enterprise and IoT datasets, NF-CSE-CIC-IDS2018-v3 and NF-ToN-IoT-v3, which differ in topology, capture conditions, and attack families. The transfer is run in both directions, from enterprise to IoT and from IoT to enterprise. The evaluation is done on the held-out test set of the target dataset, while training and rule extraction occur on separate train and validation sets.

Fine-tuning The fine-tuning follows a two-stage schedule rather than retraining the whole network on the small target sample. In the first stage the feature extractor is frozen and only the classifier head and the soft thresholds of the symbolic tiers are updated, which lets the model fit the new benign-to-attack balance without disturbing the representation learned on the source. In the second stage the whole network is unfrozen and trained at a learning rate an order of magnitude smaller, so that the shared features drift towards the target rather than being overwritten by a handful of examples. The symbolic thresholds continue to adapt in both stages. The neural baselines are adapted from the same source snapshots at the same reduced learning rate, so that any gap between them and the neuro-symbolic model reflects the rules rather than a different adaptation budget.

The comparison is conducted across four systems:

- the **oracle**, which is a neural baseline trained and evaluated directly on the target dataset;
- the **neural transfer**, which is a neural baseline trained on the source dataset and then transferred to the target dataset, either zero-shot or fine-tuned on a small number of target samples;
- the **regularised neural transfer**, which is the same as the neural transfer model but with weight decay as a regularisation control; and
- the **neuro-symbolic transfer**, which is the neuro-symbolic model trained on the source dataset and then transferred to the target dataset, either zero-shot or fine-tuned on a small number of target samples. Two versions of this model exist: the

first version defines broad rules that are general and can be applied to both datasets (**Transfer-NeSy**), while the second version defines local rules for each dataset and only transfers the neural part (**NeSy local-rules**).

The training size on the source dataset is fixed at $k = 500$, while the target budget is swept from $k = 5$ to $k = 50$ to test the effect of fine-tuning on a smaller number of target samples.

The comparisons reveal the impact of the neuro-symbolic approach on model portability across datasets by showing whether the symbolic rules carry knowledge that survives a move to a different dataset, or whether they only encode artefacts of the dataset they were mined on, and whether local rules defined on each dataset help a neural component adapt to its new environment. The comparisons also show whether the neuro-symbolic model exceeds the plain and regularised neural transfer, which would indicate that the neuro-symbolic integration, not incidental regularisation, is responsible for any advantage. Moreover, comparing the neuro-symbolic model to the oracle shows whether the symbolic component allows the transferred neuro-symbolic model to approach the performance of a model trained directly on the target dataset.

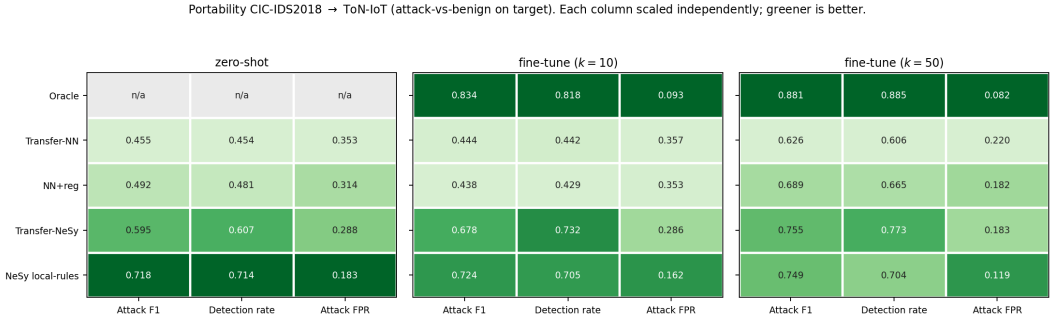


Figure 6.8: Portability for the enterprise-to-IoT pair, NF-CSE-CIC-IDS2018-v3 → NF-ToN-IoT-v3, binary classification on the held-out target over five replicates.

Figure 6.8 shows the enterprise-to-IoT transfer, and it carries two neuro-symbolic variants. Transfer-NeSy ports the broad rules together with the neural weights, while NeSy local-rules keeps the rules local to each dataset and transfers only the neural component. Both beat the neural transfer at every setting. Transfer-NeSy leads it by about fourteen attack-F1 points zero-shot and by between thirteen and twenty-three after fine-tuning, while holding a clearly higher detection rate and a lower false-positive rate (0.183 against 0.220 at k=50). The claim that transfer approaches the target-trained oracle is still not met in full: the oracle reaches an attack F1 of 0.88 at k=50 against Transfer-NeSy's 0.76.

The two variants differ in where the symbolic knowledge is fixed, and that choice shapes the trade-off. NeSy local-rules holds the lower attack false-positive rate at every budget, 0.183 against 0.288 zero-shot and 0.119 against 0.183 at k=50, and it leads on attack F1 when labels are scarcest, 0.718 against 0.595 zero-shot and 0.724 against 0.678 at k=10.

By k=50 the two are level on attack F1, 0.749 against 0.755, but Transfer-NeSy keeps the higher detection rate, 0.773 against 0.704. The reason is that a rule mined on the target's own traffic is referenced to that traffic and fires precisely there, whereas a broad rule written to hold on both datasets is calibrated to neither and over-fires on the target. Either way the gap to the oracle at k=50 is about thirteen attack-F1 points.

Robustness under Gaussian noise

The second part of the research question focuses on the robustness of the models when the test input is degraded, making it drift from the distribution learned during training. Gaussian noise is added to the test features, scaled per feature in units of that feature's standard deviation, at levels $\sigma \in \{0.05, 0.1, 0.25, 0.5, 1.0, 2.0\}$, with $\sigma = 0$ as the clean reference. Only the test set is perturbed; training is unchanged. This isolates how much of a model's clean performance survives once the measurements it relies on become unreliable. All models in this section are trained on the original CIC-IDS2017 dataset at $k = 500$.

The robustness curves feature the different coupling architectures studied in Section 6.2.1, as well as the **tiered variant** of the anchored coupling architecture, which relies on multiple threshold levels for the numerical rules, as opposed to its original counterpart that employs only the best threshold selected on the training pool and verified on the validation set. Additionally, a **categorical** variant that relies purely on rules defined on the unperturbed categorical features is added to show how much of the robustness comes from the symbolic component's ability to encode categorical anchors that are unaffected by noise or adversarial perturbation.

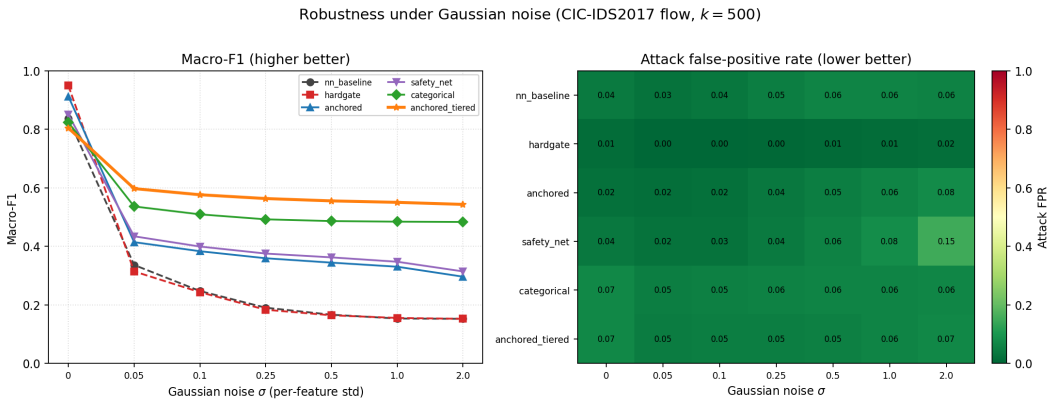


Figure 6.9: Robustness under Gaussian noise on CIC-IDS2017 flow at $k = 500$: macro-F1 versus per-feature standard deviation (left) and attack false-positive rate heatmap (right).

Figure 6.9 (left) shows the result: the models can clearly be separated into three groups. The macro-F1 of the neural baseline and the hard-gate model collapses as soon as noise is added, dropping nearly to 0.15 at $\sigma = 2.0$, which is 16% to 18% of their clean value. This effect is a consequence of the models' hard dependency on the neural component's

output. The hard-gate model, although starting with the highest macro-F1 of 0.95 on the clean test, is affected by its coupling policy, which requires both the neural and symbolic components to agree on an attack decision. This makes it more vulnerable to noise as the neural component's performance degrades, an effect clearly shown by the degradation of the neural baseline itself.

The softer variants, such as the anchored and the safety_net approaches that allow the symbolic component more freedom in the final decision, hold a much higher floor. Both models retain approximately 0.35 macro-F1 at $\sigma = 2.0$, which is about forty percent of their clean value.

The last group is composed of the anchored-tiered and categorical models. The anchored-tiered model keeps 0.543 macro-F1 at the strongest noise level, 68% of its clean score, and it is the best model at every noisy operating point. The results are a direct consequence of the tiered structure, which allows the model to capture a wider band around the optimal threshold and assign it a probabilistic weight, so that when noise pushes a sample across the optimal threshold, it can still be captured by a nearby one with a lower weight, instead of being missed entirely as in the single-threshold anchored model. The categorical model, which relies purely on rules defined on the unperturbed categorical features, holds a macro-F1 of approximately 0.50 at $\sigma = 2.0$ because the categorical features are unaffected by the noise, but it loses the patterns provided by the numerical features.

The false-positive view in Figure 6.9 (right) shows that all models raise more false alarms as noise grows, but the increase is insignificant for most models. As a consequence, the drops in macro-F1 are mostly due to a drop in recall rather than a rise in false positives, which is consistent with the fact that noise makes the model miss more attacks by pushing them across the decision boundary.

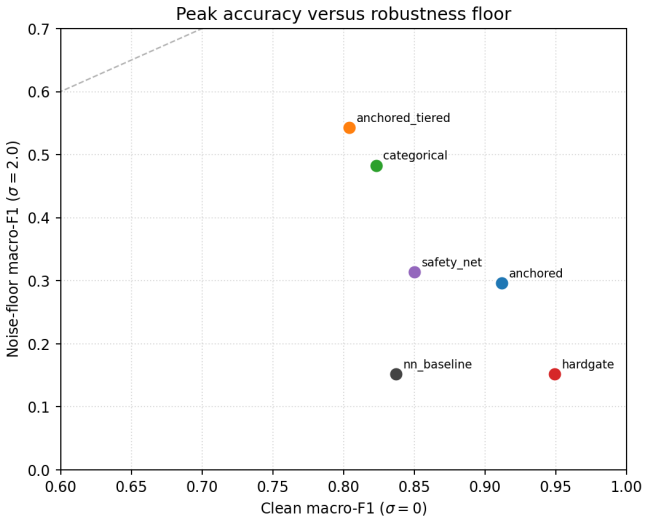


Figure 6.10: Clean macro-F1 against noise-floor macro-F1 at $\sigma = 2.0$.

Figure 6.10 plots clean macro-F1 against the noise floor at $\sigma = 2.0$. The hard-gate model sits bottom-right (highest clean, near-lowest floor) and the anchored-tiered model sits top (moderate clean, highest floor), so clean accuracy and robustness rank in roughly opposite order. The false-positive view in Figure 6.9 (right) is consistent: the hard-gate benign veto holds the lowest attack false-positive rate across the noise range, while the rescue-based models raise more false alarms as noise grows.

Robustness under constrained adversarial search

The adversarial setting replaces random noise with a search that actively tries to flip the model's decision. The threat model is a realistic evader: an attacker controls only their own attack traffic and tries to make it pass as benign, so a projected-gradient attack perturbs only the attack samples, targeted toward the benign class, within a budget ε measured in per-feature standard-deviation units. Benign traffic is left untouched, because an attacker cannot perturb other users' flows; the attack false-positive rate is therefore unaffected by construction and the metric that moves is detection. Only features an attacker could plausibly control are made mutable, while the attack-defining columns (destination port, protocol, flag counts) are frozen, so the perturbation respects the constraints of network traffic rather than moving arbitrary columns. The budgets are $\varepsilon \in \{0.05, 0.1, 0.25, 0.5, 1.0, 2.0\}$.

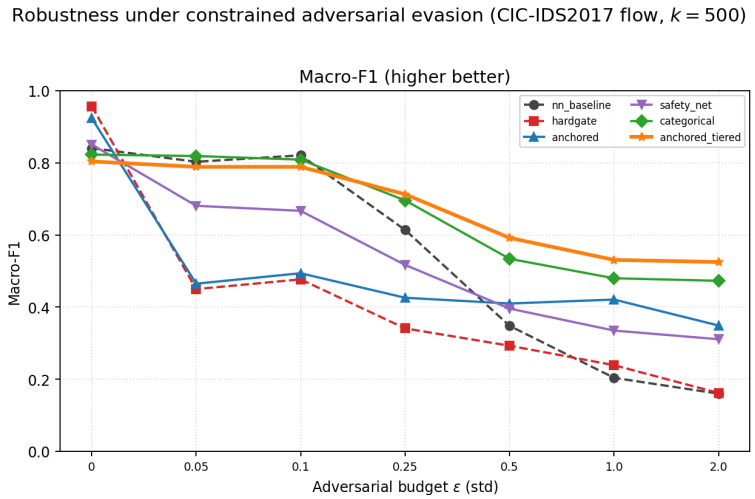


Figure 6.11: Macro-F1 under constrained adversarial evasion on CIC-IDS2017 flow at $k = 500$, as the perturbation budget ε grows. Only attack samples are perturbed, targeted toward benign.

Figure 6.11 shows that the ordering changes with the budget, and that the coupling decides who survives. At the smallest budgets the neural baseline is still among the strongest, keeping 0.82 macro-F1 at $\varepsilon = 0.1$, because a weak perturbation barely moves the decision boundary. The strict-gate couplings collapse first and fastest: the hard-gate and anchored models fall from 0.96 and 0.92 to around 0.46 as soon as $\varepsilon = 0.05$, because an attack de-

cision needs the neural prediction and the rule to agree, so once the evasion flips either one of them, the whole attack path dies. As the budget grows the neural baseline collapses too, from 0.84 to 0.160 at $\varepsilon = 2.0$, nineteen percent of its clean value, and the hard-gate model bottoms out at the same 0.162. The looser neuro-symbolic coupling variants hold a far higher floor, because both their categorical anchors and neural components can label an attack on their own without a required agreement between both: where one component fails, the other manages to detect the attack.

The categorical model keeps 0.473 macro-F1 at $\varepsilon = 2.0$, 57% of its clean score, because the symbolic component relies on features that the adversary cannot modify. The anchored-tiered model is the most robust, keeping 0.525 at the largest budget, 65% of its clean score, and it leads every other model once the budget passes the midpoint.

Because the evasion attack only moves attack samples toward benign, it is a pure detection-evasion threat: it lowers recall on the attack families rather than raising false alarms on benign traffic. The symbolic models that rely on a single threshold per feature end up collapsing once the adversary moves these numerical values in the opposite direction of the threshold, while the neuro-symbolic models relying on a looser-coupling or multiple thresholds end up resisting the perturbation for the same reasons that they resist noise. An attack decision still has to satisfy a rule, and the frozen categorical anchors on destination port and flag counts cannot be perturbed, so they keep firing on the evaded attacks while the neural boundary alone is flipped.

Summary

The second research question aims to highlight the impact of neuro-symbolic reasoning on network intrusion detection's capability to generalise across datasets and network topologies, as well as its robustness to noisy traffic and evasion attacks.

RQ.2: Generalisation & Robustness

Generalisation Neuro-symbolic reasoning helps the model generalise better across topologies through rule-based guidance if the rules are built carefully to be applicable on the target topology. However, it cannot compete with a model trained locally on the new topology.

Robustness Neuro-symbolic reasoning renders the model more robust to noise and adversarial attacks, as the symbolic layer provides a stable structure and rules that describe attack patterns and are not as influenced by the original distribution as purely neural models that learn their distributional assumptions from the training data. The coupling policy also plays a major role in the robustness of the model, with architectures that are more permissive being more robust to noise and adversarial attacks, while more restrictive architectures that require agreement between the neural and symbolic components are more fragile.

6.2.3 Explainability

The third research question asks whether a neuro-symbolic approach can provide explanations for its decisions that are faithful to the model's actual reasoning process, and whether these explanations are useful for a human analyst to understand and trust the model's decisions.

This section is dedicated to answering this question through an analysis of the symbolic proof paths that lead to attack decisions in the neuro-symbolic model, and how they can be interpreted in terms of known attack techniques.

The explanations examined in this section are read directly from the arithmetic circuit that produces the decisions. The motivation behind this choice stems from the fact that an explanation is only meaningful if it corresponds to the computation that actually labelled the traffic. The explanations are therefore faithful by construction, not approximations whose agreement with the model has to be assumed.

Symbolic proof-path coverage

Coverage is reported as the share of attack alerts that can be attributed to a named clause. This section examines the coverage of the symbolic rules across the different coupling strategies defined. The goal is to show to what extent the symbolic layer contributes to attack decisions, and whether it does so in a way that is consistent with the coupling design.

Datasets The coverage is measured on both the original CIC-IDS2017 flow dataset and the standardised NF-CSE-CIC-IDS2018-v3 packet-level dataset. The packet pipeline coupling is of particular interest, because it leaves an open path for the neural classifier to label an attack without any rule at all, so it tests whether the symbolic layer is actually contributing to the decisions.

Coupling policies In the guarded coupling, every attack alert is a conjunction of the neural prediction and a rule, so all attack alerts are attributed to a clause by construction.

The more informative cases are the couplings that do not force this. In the anchored coupling about three quarters of attack alerts are decided by an anchor clause that fires on its own, without requiring the neural classifier to agree, and the remaining quarter are neural-gated rescues. The packet pipeline on NF-CSE-CIC-IDS2018-v3 is the most interesting test, because it keeps an open neural path that could in principle label an attack with no rule at all: across the attack alerts that open path wins none, so every alert is still decided by a rule even when the architecture does not demand it.

Figure 6.12 shows the three couplings, and the absence of any unexplained share supports the claim. Additionally, this result holds on multiple datasets and traffic views, which indicates that the coverage property is a feature of the neuro-symbolic design of the system rather than an artefact of one dataset.

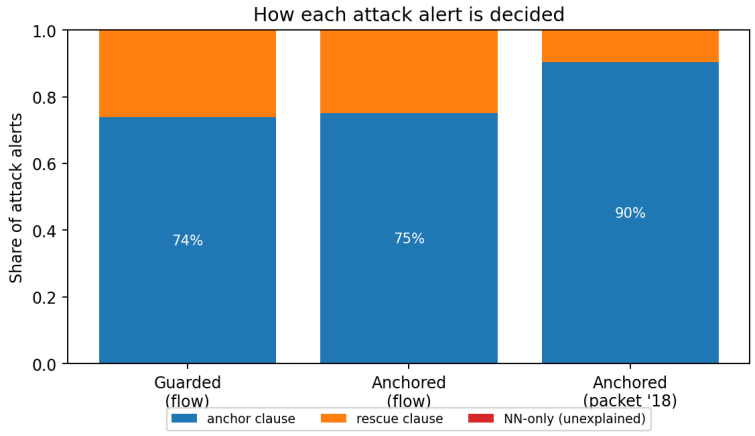


Figure 6.12: Share of attack alerts decided by an anchor clause, a neural-gated rescue clause, or the neural path alone, for three couplings.

The composition of the proof paths shifts with the amount of training data, as Figure 6.13 shows across the full sweep. At $k = 5$ anchors decide 84.5% of attack alerts and rescues the rest, whereas at $k = 500$ the split moves to 46.3% anchors and 53.7% rescues. The reason is that with more attack evidence the model leans more on the neural-gated rescue clauses, which combine the classifier with weaker rules, since the neural component becomes more influential and reliable. Meanwhile, the high-precision anchors carry proportionally fewer of the alerts.

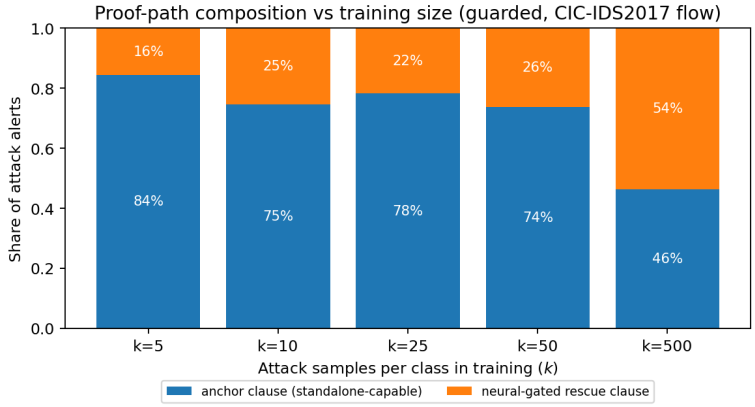


Figure 6.13: Proof-path composition of attack alerts versus training size k, guarded coupling on CIC-IDS2017 flow.

Fidelity of symbolic clauses

The deciding clauses are also accurate on held-out test traffic. Table 6.5 reports, for each clause at $k = 50$, the number of alerts it decides, its test precision (the fidelity of the explanation when it fires), and the recall of the attack class it serves. Anchor clauses reach precisions between 0.90 and 1.00, with the denial-of-service, port-scan, and web-attack anchors at 0.993, 0.997, and 0.990; the rescue clauses range from 0.59 to 0.97, and the benign gate reaches 0.990. Recall stays high across the classes these clauses cover (0.80–0.99), so the rules are not buying precision by abstaining: a clause that decides an alert is usually correct when it fires, and the clauses together still catch most of the attacks present. The explanation is therefore not only present but trustworthy.

Table 6.5: Per-clause fidelity and per-class recall for the guarded coupling on CIC-IDS2017 flow for $k = 50$.

Clause	Type	Alerts triggered	Test precision (fidelity)	Class recall
benign gate	gate	15,625	0.990	0.987
bot anchor	anchor	82	0.902	0.796
bot rescue	rescue	27	0.593	
bruteforce rescue	rescue	245	0.784	0.985
dos anchor	anchor	2,440	0.993	0.964
dos rescue	rescue	287	0.693	
portscan anchor	anchor	605	0.997	0.981
portscan rescue	rescue	588	0.968	
webattack anchor	anchor	101	0.990	0.877

Interpretability

A vital property of these explanations is that they can be interpreted in the vocabulary used by analysts. Table 6.6 grounds each anchored clause of the guarded neuro-symbolic model in a plain-language reading and maps it to a technique in the MITRE ATT&CK framework, the standard catalogue of adversary behaviour.

Every deciding clause corresponds to a named technique: the bot anchor to web-based command and control, the brute-force rescues to password guessing against FTP and SSH, the denial-of-service anchors to endpoint and service-exhaustion floods, the port-scan anchor to network service discovery, and the web-attack anchor to the exploitation of a public-facing application.

These clauses were not authored by hand. They are the output of the automated mining pipeline of Section 5.4, which ranks features and calibrates thresholds from training statistics and verifies them on validation, yet the conditions it recovered line up with documented techniques, so the mined rules are interpretable in the terms an operator already uses. Two caveats hold: the rescue clauses carry lower fidelity by design (0.59 to 0.97), and the port-

Table 6.6: Champion symbolic clauses (guarded coupling, CIC-IDS2017 flow) mapped to MITRE ATT&CK techniques. **Fidelity:** clause's precision when deciding an attack

Clause	Conditions (analyst reading)	MITRE technique	Fidelity
bot anchor	dst port 8080, fwd-window tier (scripted web-proxy check-in)	T1071.001 Web C2	0.90
bot rescue	low source-port + small-payload tiers (periodic C2 beacon)	T1071 / T1573	0.59
brute-force rescue	dst port 21/22, min-seg tier (FTP/SSH credential guessing)	T1110.001 Password Guessing	0.78
dos anchor	dst port 80, backward-length-variance tier (HTTP flood)	T1499 Endpoint DoS	0.99
dos anchor	dst port 80, FIN set (connection-churn teardown)	T1499.002 Service Exhaustion	0.99
dos anchor	dst port 80, ACK only, sub-5μs inter-arrival (machine-rate flood)	T1499	0.99
portscan anchor	high backward-packet-rate + default fwd-window tiers (SYN scan)	T1046 Service Discovery	1.00
portscan rescue	tiny packets, near-zero duration (single probe)	T1046	0.97
webattack anchor	regular fwd inter-arrival tiers, minimal payload (scripted tooling)	T1190 Exploit Public App	0.99
benign gate	neural benign vote and no anchor fires	—	0.99

based conditions are specific to these captures while the flag, timing, and size conditions describe the attack shape that carries to other settings.

Moreover, while the mined rules have the potential to map to attack descriptions in known knowledge bases, there is no guarantee that the pipeline will always achieve this, because the computed statistics depend on the dataset used. Hand-written rules still have an advantage in interpretability, as they are written by an expert analyst and are therefore intended to be human-interpretable from the start.

Explanation stability across replicates

A second requirement for a useful explanation is stability across different training runs: the reason a rule produces a prediction should not change if the model is trained on a different sample of the same data. Stability is measured by training five replicates, holding the test set fixed, and checking how often the explanation of a sample agrees across the replicates. For the symbolic models agreement means the same winning clause; for the neural baseline, where the SHAP and Integrated Gradients attributions are used, agreement refers to the same most-important features.

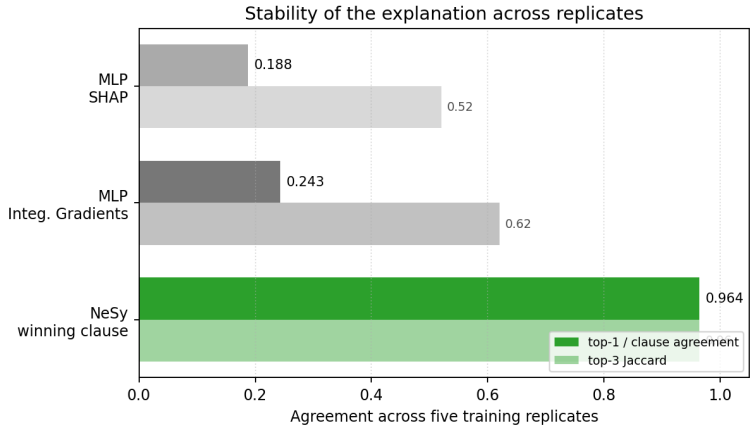


Figure 6.14: Agreement of the explanation across five training replicates on CIC-IDS2017 flow, for the symbolic winning clause and the Integrated Gradients and SHAP attributions on the neural baseline.

Figure 6.14 reports the result. The symbolic winning clause agrees across replicates on 0.964 of samples, and on the samples where the five replicates also agree on the label, the clause agreement rises to about 0.999. The post-hoc attributions on the neural baseline are far less stable: Integrated Gradients selects the same top feature on 0.243 of samples and SHAP on 0.188, and even the looser measure of a shared top-three set reaches only 0.62 and 0.52. The motivation behind including both attribution methods for the neural baseline stems from the fact that instability in a single method could be a property of that method, whereas the same instability in two widely used methods points to the neural model itself: the feature a gradient-based method credits depends on which of several correlated features a given replicate happened to rely on, while the symbolic clause is tied to attack characteristics that do not move between replicates.

Stability is not uniform across classes, as Figure 6.15 shows. The web-attack, benign, and port-scan explanations are the most stable, at 1.00, 0.99, and 0.98, while the bot class is the least stable at 0.54. The bot result is consistent with its small support of 167 test samples and with the fact that bot alerts sit between an anchor and a rescue clause, so a small change in the trained weights is enough to move which of the two decides a given alert. The classes with stable, high-precision anchors are exactly the classes whose explanations are most stable.

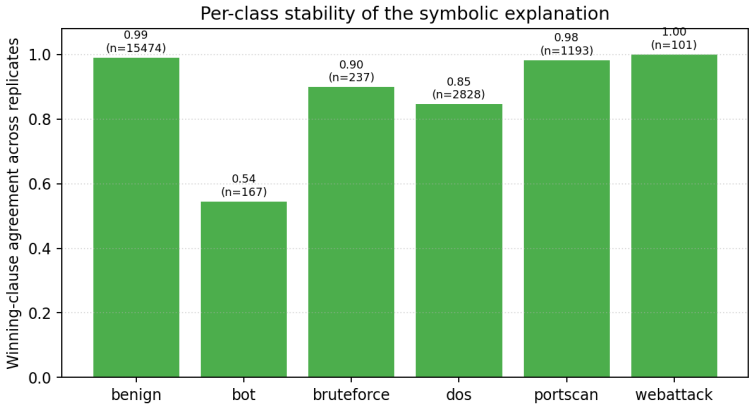


Figure 6.15: Per-class agreement of the symbolic winning clause across replicates, with test-sample count per class.

Attribution contrast with neural explanations

Table 6.7: Symbolic explanation against post-hoc neural attribution on the same CIC-IDS2017 flow samples. "pred" is the neural baseline's assigned class.

Sample (true)	Symbolic deciding clause (NeSy)	IG top-1 (NeSy pred / NN pred)	SHAP top-1
bot	bot anchor: dst port = 8080, fwd init-window tier	Destination Port, unstable (bot / benign)	Source Port, unstable
bruteforce	bruteforce rescue: dst port = 21, min-seg-size tier	Fwd PSH Flags, unstable (bf / benign)	Fwd PSH Flags, unstable
dos	dos anchor: dst port = 80, bwd-pkt-len-std tier	Packet Length Variance, stable (dos / dos)	Packet Length Variance, stable
portscan	portscan anchor: bwd-packets/s tier, fwd init-window tier	Destination Port, unstable (ps / ps)	Total Bwd Packets, unstable

The stability numbers are made concrete by individual case studies, in which the symbolic explanation of a correctly detected attack is placed next to the Integrated Gradients and SHAP explanations of the neural baseline on the same sample. Table 6.7 gives four of them. Two differences recur. First, the symbolic explanation names the exact conditions that fired, such as a destination port of 8080 together with an initialisation window in a learned tier, whereas the attribution returns a feature ranking whose top entry changes from one replicate to the next. Second, and more seriously, the attribution often explains a different decision from the one the neural baseline actually made: for several attack samples that the neuro-symbolic model labelled correctly, the neural baseline predicted benign, so its attribution is an explanation of a missed detection rather than of a catch. The symbolic explanation does not have this failure mode, because it is the proof of the decision rather

than a commentary on it.

The denial-of-service case shows the same pattern from the other direction. For this attack the neural baseline both predicts the attack and gives a stable attribution, packet-length variance, which the symbolic explanation echoes through its packet-length condition. When the neural decision is itself well grounded, the two explanations agree; the contrast appears precisely on the samples where the neural decision is shaky, which are also the samples where an analyst most needs a dependable reason.

Summary

The third research question asks what the symbolic layer contributes to the interpretability of the detector, whether those contributions are interpretable by analysts, and whether the evidence is consistent across the three subsections.

RQ.3: Explainability

Reliability The symbolic layer provides rules that are directly read from the classification circuit. The explanations are reliable because every attack alert is attributed to a named clause, including under a coupling whose architecture would allow unexplained alerts. Moreover, they are accurate because they maintain high precision on the test data.

Stability The symbolic layer provides explanations that are stable across retraining, agreeing on the large majority of samples. In contrast, the post-hoc attributions on the neural baseline change their leading feature on most samples for models trained on different subsets of the same dataset.

Interpretability The symbolic component is meaningful to a practitioner, because each mined clause corresponds to a named MITRE ATT&CK technique, and the fact that these clauses came from the automated rule-generation pipeline rather than from hand authoring shows that the pipeline recovers signatures that can be linked back to prior expert knowledge.

6.3 Ablation studies

This section reports the ablation studies conducted to answer the remaining questions about system decisions and performance.

6.3.1 Hyperparameter selection

The neuro-symbolic model adds two learning rates where a plain network has one: alongside the neural rate, a separate rate trains the probabilistic parameters of the logic program and a third trains the soft-threshold tier networks, on top of the usual batch size and epoch count. A full joint grid over all of these is too expensive, so selection runs in two stages on

the validation set. Every candidate is scored by the validation macro-F1, penalised by half the attack false-positive rate and a quarter of its standard deviation across replicates, which rewards detection without rewarding false alarms or run-to-run instability.

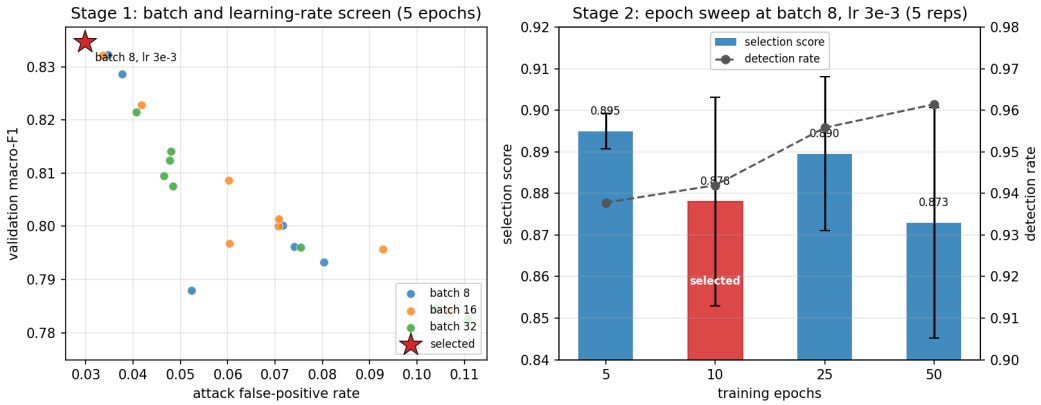


Figure 6.16: Two-stage hyperparameter selection on the validation set. Left: stage-one screen of batch size and learning rates at five epochs, placed by macro-F1 and attack false-positive rate and coloured by batch size (selected configuration starred). Right: stage-two epoch sweep over five replicates, selection score per budget (selected ten epochs in red) with detection rate.

Table 6.8: Stage one of the hyperparameter search: five strongest batch and learning-rate configurations at five epochs, one validation replicate. η_{nn}, η_{dpl}, η_{tier} are the neural, probabilistic, and tier learning rates; score is macro-F1 $-0.5\cdot$FPR $-0.25\cdot$std.

Rank	Batch	η_{nn}	η_{dpl}	η_{tier}	Macro-F1	Attack-FPR	Score
1	8	0.003	0.003	0.001	0.835	0.030	0.820
2	16	0.001	0.003	0.001	0.832	0.034	0.815
3	8	0.001	0.001	0.001	0.832	0.035	0.815
4	8	0.001	0.003	0.001	0.829	0.038	0.810
5	16	0.001	0.003	0.003	0.823	0.042	0.802

The first stage fixes five epochs and screens the batch size against the three learning rates at a single replicate (Table 6.8, Figure 6.16 left). A batch size of eight with the higher neural rate of 0.003 wins on both axes, reaching the best macro-F1 at the lowest false-positive rate.

Table 6.9: Stage two of the hyperparameter search: epoch sweep at the stage-one configuration (batch 8, $\eta_{nn} = 0.003$, $\eta_{dpl} = 0.003$, $\eta_{tier} = 0.001$), five validation replicates.

Epochs	Macro-F1	Attack-FPR	Detection rate	Score
5	0.906 ± 0.004	0.021	0.938	0.895
10 (selected)	0.895 ± 0.025	0.021	0.942	0.878
25	0.904 ± 0.018	0.020	0.956	0.890
50	0.891 ± 0.028	0.022	0.961	0.873

The second stage fixes the winning batch and learning rates and sweeps the epoch count at five replicates, judging training length against run-to-run variance (Table 6.9, Figure 6.16 right). The epoch count barely matters: the four budgets all reach a macro-F1 between 0.89 and 0.91 at almost the same false-positive rate, within one standard deviation of one another, and paired t-tests against ten epochs find no significant difference (five epochs $p \approx 0.44$, twenty-five $p \approx 0.60$, fifty $p \approx 0.81$). Detection rate is the only quantity that moves, rising from 0.94 at five epochs to 0.96 at fifty as longer training recovers a few more attacks. Ten epochs is selected as a short, clearly converged budget, with the understanding that any value in this range performs equivalently. The full configuration is a batch size of eight, a neural rate of 0.003, probabilistic and tier rates of 0.003 and 0.001, and ten training epochs. The choice of ten epochs was motivated by the heavy training duration of a neuro-symbolic model, favouring fewer epochs due to the time constraints and number of experiments conducted. Furthermore, the differences between the four epoch counts are statistically insignificant and the model has a high probability of converging at ten epochs.

6.3.2 Temporal and Spatial Complexity

The cost of the symbolic layer is measured on a ladder of six programs on CIC-IDS2017 flow at $k = 500$, from a rule-free neural classifier (L0) to the full anchor-tiered program (L5), with architecture, training budget, and hardware fixed and three replicates per rung.

Table 6.10: Complexity ladder on CIC-IDS2017 flow at $k = 500$, from a rule-free neural classifier (L0) to the full anchor-tiered program (L5), three replicates per rung.

Rung	Leaves	SDD edges	Throughput (pred/s)	Latency (ms/flow)	Train (s/epoch)
L0 neural-only	1	267	44,097	0.024	1.9
L1 single anchor	3	333	17,112	0.059	3.0
L2 gated	8	538	7,681	0.133	5.5
L3 gated + numeric	10	484	8,162	0.123	5.3
L4 tiered	10	484	8,111	0.123	8.4
L5 anchor-tiered	15	461	6,520	0.156	20.1

Table 6.10 reports the ladder and Figure 6.17 separates the three costs. The main finding is that inference, memory, and training are each governed by a different quantity, and none is a proxy for the others.

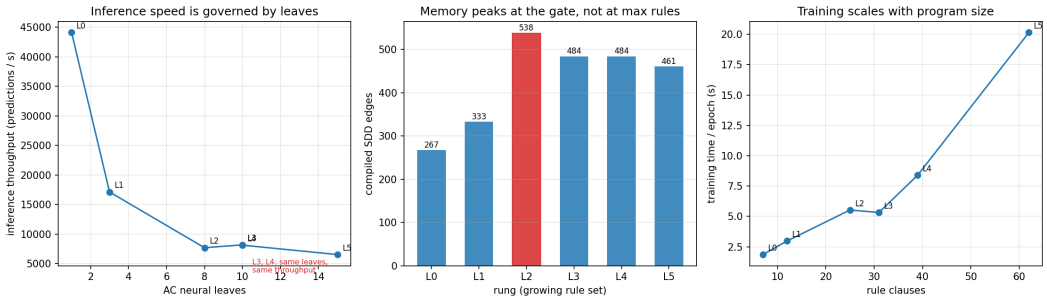


Figure 6.17: The three costs of the symbolic layer, each on a different axis. Left: inference throughput versus neural-leaf count. Centre: compiled-circuit size per rung. Right: training time per epoch versus rule-clause count.

Inference throughput tracks the neural-leaf count, not the clause count, falling roughly sevenfold from 44,000 predictions per second with no rules to about 6,500 at the full program. L3 and L4 share ten leaves and run at the same speed (8,162 against 8,111 predictions per second) even though L4 has eight more clauses, because a clause that adds no leaf is compiled away and costs nothing at inference. Each leaf is one network evaluation, so the leaf count is the right summary of inference cost. In absolute terms the per-flow latency rises from 0.024 ms with no rules to 0.156 ms at the full program, and even the heaviest program serves 6,500 flows per second, adequate for flow-level deployment in offline settings.

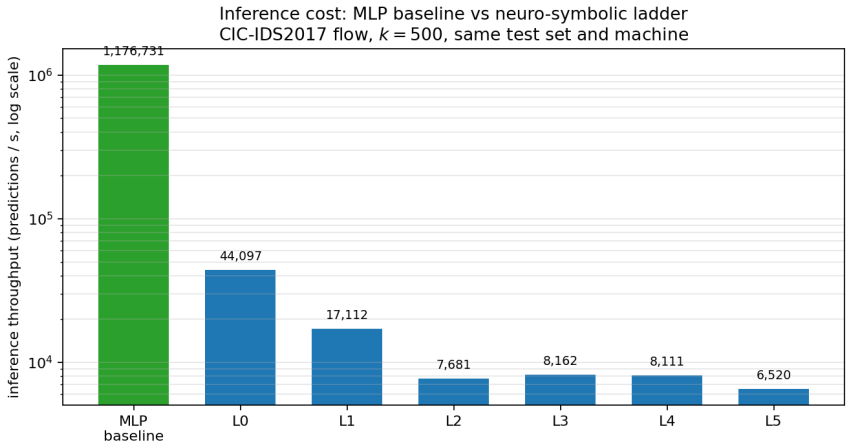


Figure 6.18: Inference throughput of the MLP baseline against the neuro-symbolic ladder (L0–L5) on CIC-IDS2017 flow at $k = 500$.

Figure 6.18 sets this overhead against the neural baseline on the same test set and machine. The plain MLP runs at about 1.18 million predictions per second, roughly twenty-seven times the rule-free circuit (L0) and 180 times the full program (L5). The gap is the cost of evaluating a compiled circuit instead of one forward pass.

Inference, Training, and Spatial Complexity Analysis

Memory is better described by the compiled-circuit size than by the process resident set. The resident set is dominated by the interpreter and the shared network, so it barely moves across the ladder and does not track the rule set. The circuit itself does track it, but not monotonically: it peaks at L2, the benign-negation gate, at 538 edges, while the richer tiered rungs compile smaller, down to 461 at L5. The gate entangles the formula and is the memory hotspot, whereas the tier and numeric structure added above it is absorbed by knowledge compilation.

Training time grows fastest, from under two seconds per epoch to twenty, scaling with the number of soft-tier predicates, each a small trainable sub-network. L3 and L4 share the same circuit and inference cost, yet L4 trains in 8.4 seconds against L3's 5.3 because it adds seven tier predicates, and L5's twenty-eight tier predicates account for its twenty-second epochs.

These behaviours reduce to three quantities: the neural-leaf count L, the compiled-circuit size in edges E, and the number of soft-tier predicates P. Inference evaluates the whole circuit, but each leaf is one forward pass of the shared network and those passes dominate the arithmetic, so per-flow cost is governed by L rather than E ($0.030 + 0.0094\,L$ ms, $R^2 = 0.90$). This is why L3 and L4 share a speed: they share ten leaves, and L4's extra clauses add only cheap arithmetic. Training adds one small trainable sub-network per soft predicate, so an epoch costs $\Theta(P)$ beyond the circuit and rises linearly ($3.5 + 0.59\,P$ s, $R^2 = 0.97$). Compilation is the one axis with no benign law: the circuit size is worst-case exponential in the treewidth of the program's interaction graph, and the non-monotone peak at the benign-negation gate is that sensitivity surfacing while the program is still small enough to avoid the blow-up. Because three different parts of the program set the three costs, a coupling can be cheap to serve and slow to train, or heavy in clauses and light in memory, instead of paying on every axis at once.

A final caveat: the absolute timings are specific to the machine of Section 6.1, so it is the relative scaling across the ladder, not the millisecond figures, that carries to other hardware.

Complexity study on the coupling architectures

The ladder isolates what each structural element costs by building specific neuro-symbolic models designed for this experiment, but the deployed couplings of Section 6.2.1 are the programs that are used throughout the experiments. Table 6.11 reports the five couplings and the symbolic-only baseline at $k = 500$ on the same axes. Their structural cost barely moves: every coupling compiles to about 22 neural leaves (twenty-one for symbolic-only) and serves between 1,500 and 2,300 flows per second, under a factor of two apart, even as the compiled size ranges fourfold from 247 to 981 edges. Detection quality moves far more, from a macro-F1 of 0.954 for the guarded coupling to 0.667 for symbolic-only. Inference cost, fixed by the leaf count, is therefore almost flat while accuracy varies by nearly thirty points, and no cost axis orders the couplings by accuracy: symbolic-only compiles the smallest circuit yet scores worst, and anchored at 0.923 undercuts guarded at 0.954 on every axis at once. Complexity buys nothing on its own; what separates 0.95 from 0.67 is which rules fire, not how many leaves or edges the program carries. The fourfold spread in edges

again leaves throughput within a factor of two, which confirms the leaf-bound reading on an independent family of programs; the absolute rates sit below the ladder's only because these full programs carry heavier per-leaf networks.

Table 6.11: Cost of the deployed couplings of Section 6.2.1 on CIC-IDS2017 flow at $k = 500$, on the cost axes of Table 6.10 and the same machine, ordered by macro-F1. Leaves is the compiled circuit's neural-leaf count, edges its size, throughput the predictions per second, and latency the steady-state inference time per flow; macro-F1 is the five-replicate detection score of Section 6.2.1.

Coupling	Leaves	SDD edges	Throughput (pred/s)	Latency (ms/flow)	Macro-F1
Guarded	22	900	1,945	0.517	0.954
Hard-gate	22	941	1,505	0.681	0.951
Anchored	22	772	2,183	0.461	0.923
Same-class boost	22	981	2,206	0.455	0.884
Safety-net	22	964	2,236	0.448	0.744
Symbolic-only	21	247	2,316	0.439	0.667

Soft vs Hard thresholds Every rung above uses soft thresholds rather than the crisp alternative; Appendix C.8 isolates this choice on a single fixed program and reports that the soft representation trains an order of magnitude faster and serves roughly three hundred times more flows per second, because it compiles one cacheable circuit instead of recompiling per flow.

Chapter 7

Discussion

This chapter interprets the results of Chapter 6. It explains why the patterns observed across the three research questions appear, draws together findings that only become meaningful when read across experiments, relates them to prior work, and states plainly what the results do not establish. The chapter closes with a consolidated account of the limitations of the approach and the directions they open for future work.

7.1 The role of the symbolic layer

Read across the detection and efficiency experiments, the contribution of the symbolic layer concentrates exactly where the neural classifier is weakest. Three conditions bring it out: a small labelled budget, a thin traffic view, and the false-positive axis. Where the neural baseline already has enough evidence to learn the families on its own, the symbolic layer adds little; where it does not, the symbolic layer carries the model.

The mechanism behind this stems from how the two components depend on data. The neural classifier needs enough samples of each attack family to learn its decision boundary, so in the low-data regime it is fed only a handful of examples per class and has little to learn from. The mined rules do not share this dependence. They fire on benign-calibrated evidence whose thresholds are estimated once from the stable benign distribution, so a rule is already selective before the model has seen many attacks, and the training subset only adjusts its weight once the rule itself is in place. The symbolic evidence is therefore available from the first few samples, while the neural evidence is still scarce. This is what produces the large gaps at small k. Additionally, the gated couplings pass the baseline's best score already at $k = 10$, a roughly fiftyfold reduction in the labelled data needed to match it. The effect is sharpest in the packet view, where a single packet carries the least evidence: the guarded coupling outperforms the neural baseline by a noticeable margin, while cutting the attack false-positive rate by an order of magnitude. The explainability results further reinforce the neural component's dependence on training volume: as training data increases, the model relies less on anchor symbolic rules and shifts toward neural-gated rescue rules because the neural component becomes more reliable.

The advantage is largest in the low-data regime, but it does not vanish once data is plen-

tiful. At $k = 500$ the per-attack quantities do converge: the neural baseline reaches detection and exact-attack rates that match the neuro-symbolic models and in places slightly surpass them, so the symbolic layer no longer recovers attacks the network would otherwise miss. The headline macro-F1, however, stays well apart and the source of that persistent gap is the false-positive axis. A benign prediction still has to survive the absence of attack evidence, so the gated couplings hold a low attack false-positive rate across the whole sweep while the neural baseline only approaches it slowly, and macro-F1 punishes the baseline's residual benign errors. The symbolic layer therefore does more than buy data efficiency: even when detection has converged, the false-positive suppression keeps the macro-F1 ceiling above the purely neural one.

These observations line up with prior work from three directions. The low-data advantage echoes the sample-efficiency result reported for DeepProbLog in non-security domains by Manhaeve et al. [31], now observed in a setting where labelled attack traffic is genuinely scarce. The false-positive control speaks directly to the problem Ahmad et al. [1] identify as one of the main practical barriers to operational NIDS, and the experiments show that the gating policies act on exactly that axis. And the guidance mechanism, in which a rule constrains the hypotheses the network has to consider, is the same effect that the single-axiom precision gain of Grov et al. [18] illustrated in a simpler setting. The one condition all of this assumes is that the attack families are separable in the available features, which the UNSW-NB15 results show is not always true, and which Section 7.3 takes up directly.

7.2 Coupling as an operational lever

Another finding is that the way the symbolic and neural components are wired matters as much as whether the symbolic layer is present at all. The couplings of Section 4.3 are points on a spectrum that runs from strict agreement, where an attack needs both the network and a rule, to full permissiveness, where either alone suffices. On clean traffic that spectrum sorts roughly by precision: the gated couplings, which demand corroboration, post the best clean macro-F1, with the guarded and hard-gate couplings holding the highest scores on the original flow data, while the most permissive safety-net coupling caps lower than the neural baseline because its disjunctive rule fires on enough benign flows to cost precision.

The clean ranking inverts almost exactly when the input is degraded, and this inversion is the central result of the robustness experiments. Under Gaussian noise the hard-gate coupling, which has the highest clean score, collapses fastest, degrading heavily at stronger noise levels and reaching roughly the same floor as the neural baseline. The permissive and standalone-symbolic couplings hold a far higher floor, staying at around two thirds of their clean scores, with the categorical model close behind.

The adversarial evasion experiments tell the same story with sharper edges and from a slightly different angle. The strict-gate couplings are the first to fall as soon as a small perturbation flips the neural prediction, while the neural baseline still resists the lowest perturbation levels. The looser variants and the more complex setups that rely on multiple thresholds per feature, such as the tiered architectures, show resistance across all noise levels. These results indicate that relying solely on the symbolic component can be detri-

mental: the clear early fall-off of the strict couplings, while the baseline resists, highlights that the symbolic component failed to detect the perturbed attacks. In fact, relying on a single threshold can be detrimental when manipulated features cross it and stop triggering the rules completely, while defining a wider band of thresholds or relying more on the neural baseline can increase robustness where the symbolic component is brittle. The trade-off figure of Section 6.2.2 states it plainly: the model with the best clean accuracy is the most fragile, and the ranking by robustness is close to the reverse of the ranking by clean accuracy.

The mechanism is the logical shape of each coupling. A strict gate is a conjunction of the neural prediction and a rule, so the proof needs both leaves to hold, and the moment the neural leaf degrades under noise or the symbolic rule is flipped by an evader the whole attack path dies. The permissive couplings keep an alternative proof alive: an OR-rescue or a standalone categorical anchor can support an attack on its own, so a symbolic path survives even when the neural distribution has collapsed and vice versa. The property that makes the safety-net coupling weak on clean traffic, namely that its rule can fire without the network, is the same property that makes it robust under perturbation. The two behaviours are one mechanism seen at two operating points.

This reframes the couplings as an operational lever the deployer sets, instead of a single architecture to be fixed once and reported. The guards act as a recall lever, suppressing benign predictions so the symbolic layer can rescue attacks the network missed, and the gates act as a precision lever, demanding symbolic corroboration before an attack is accepted, with the anchored coupling sitting between them by letting only the most reliable rules act alone. There is no coupling that wins everywhere, and the experiments show why there cannot be: low false alarms on clean traffic and graceful degradation under attack pull in opposite directions, and a coupling buys one with the other. A site that runs mostly clean traffic and pays dearly for false alarms should choose a gate; a site that expects active evasion or noisy measurement should accept a higher clean false-positive rate for a coupling that keeps detecting when its inputs are corrupted. Expressing that choice as part of the model, and measuring its cost on both axes, is something a design built around a single fixed coupling cannot do.

7.3 The separability boundary

The UNSW-NB15 results mark the limitation of the approach, and the limitation lies in the dataset rather than in the method. On that dataset every coupling, the neural baseline, and the symbolic-only model collapse into the same low band: detection rate stays high while the rate of correct family attribution is capped, so the systems can tell that traffic is an attack but cannot tell which family it is. As Section 6.2.1 shows, the cause is in the data. Several attack families share large regions of the flow feature space, a pattern independently reported in separability studies of UNSW-NB15 [51, 48], so a rule precise for one family also fires for the others, and the per-attack false-positive rate rises the moment a rule is broad enough to cover a class.

Our approach cannot solve this, and the reason is structural. A mined rule reads only the

features it is given, so it cannot manufacture a distinction that the features do not contain. Where the families separate, as they do on CIC-IDS2017, the symbolic layer sharpens the decision and suppresses false alarms; where they overlap, no threshold or category test recovers a boundary that is absent from the measurements, and the symbolic layer inherits the same confusion as the neural classifier it is meant to guide. The standardised NetFlow-v3 version of UNSW reproduces the low band at the packet, flow, and hybrid views for the same overlap reason, collected in Appendix C.7, which confirms that the boundary follows the dataset and not the traffic representation. No coupling among the ones studied moves it: even the most expressive variant, a product-of-experts that reads the co-occurrence pattern of fired rules as a second classifier (Figure C.3, Appendix C.6), only shifts the result within the same low band when the rule bank happens to be rich, and falls back to the neural level once the rule bank thins. It narrows the confusion at the margin but does not erase the boundary, so the ceiling stays fixed by how separable the families are in the features the rules can read.

That ceiling is a property of the dataset. UNSW-NB15 is a known synthetic capture whose families overlap because several features are fixed or absent, an effect documented in prior analyses of the dataset [51, 50, 48], so the separability boundary reflects how the capture was generated rather than how real traffic behaves, where the families are expected to be more distinct. UNSW therefore stands as the honest edge of the contribution: the symbolic layer helps when the attacks carry separable signatures, and on data where they do not, it detects the attacks but cannot reliably name them, a gap the approach cannot close.

7.4 Explainability: faithful versus useful

The clearest contribution of the symbolic layer is the kind of explanation it produces. An explanation is read from the same arithmetic circuit that produces the decision, so it names the predicates and probabilities that actually participated in the class query. This faithfulness is structural, which is the difference between the proof of a decision and a commentary examining the behavior without contributing to the classification. The coverage measurements make the point concrete: every attack alert is attributed to a named clause even under the packet pipeline, whose architecture leaves an open neural path that could in principle decide an alert with no rule, and the deciding clauses are accurate, with the anchors reaching very high test fidelities.

The contrast with post-hoc attribution is where the experiments turn a structural claim into a measured one. The symbolic winning clause agrees consistently across five retrainings, while Integrated Gradients and SHAP rarely select the same top features for the test samples across the five replicates. The sharper observation is qualitative: on several attack samples, the neuro-symbolic model labelled the sample correctly while the neural baseline predicted benign, so its attribution explains a missed detection while presenting itself as an explanation of a catch. The denial-of-service case, where the neural decision is itself well grounded and the two explanations agree, is the exception that locates the pattern, because the divergence appears precisely on the samples where the neural decision is shaky and an analyst most needs a dependable reason. The mapping of each champion clause to a MITRE

ATT&CK technique adds the last piece: the conditions the automated pipeline recovered line up with documented attacker behaviour, so the explanations speak in a vocabulary an operator already uses.

What the results establish is faithfulness and stability, and it is worth being precise that this is not the same as demonstrated usefulness to an analyst. The experiments measure properties of the explanation, namely that it is the true reason for the decision and that it does not move under retraining, but they do not run the analyst study that would show the explanation actually speeds triage or improves trust. One caveat keeps the claim honest. The rescue clauses carry lower fidelity by design, so a rescue-explained alert should be treated as lower-confidence triage than an anchor-explained one. The interpretability contribution is therefore strong on the axes it was tested on, and its operational value remains a reasonable expectation that the present evaluation supports but does not prove.

7.5 The price of the symbolic layer

The symbolic layer is not free, and the complexity ablation shows that its cost is paid on three axes that move independently. Inference cost tracks the number of neural leaves in the compiled circuit, because each leaf is one network evaluation and a clause that adds no leaf is compiled away at no inference cost. Memory tracks the compiled-circuit size, which does not grow monotonically with the rule set and in fact peaks at the benign-negation gate before the richer tiered programs compile smaller. Training time tracks the number of soft-tier predicates. None of the three is a proxy for the others, so a program can be cheap to evaluate and slow to train, or rich in clauses and stable in memory.

A vital figure for deployment is the gap to a plain network. The MLP baseline serves roughly twenty-seven times the rule-free arithmetic circuit and one hundred and eighty times the full program. That multiplier is the concrete price of evaluating a compiled circuit, and it buys the faithful explanations of Section 7.4 and the robustness of Section 7.2. The heaviest program is still, however, well inside flow-level rates, especially for offline classification, so the trade is affordable for the flow and hybrid views, where a decision waits on flow assembly anyway. It is tighter for the packet view, where the appeal of packet-level detection is line-rate classification, and where keeping the leaf count small is the design constraint that decides whether the symbolic layer can be afforded at all. The ablation also reminds the reader that accuracy does not climb with complexity: it is highest at the simplest rungs and dips at the tiered one, and the deployed couplings make the point sharper still, since they carry an almost identical structural cost while their detection spreads clearly in terms of macro-F1 score. The cheapest circuit among them is the weakest, and the best coupling is undercut on every cost axis by a slightly weaker one, so the cost should be spent on the rules that carry signal and not on rule count for its own sake.

Two operator-facing questions sit under these figures, and the algorithmic shape of the system answers both. The first is whether the number of flows evaluated per second is sufficient, and the rate is a single-thread figure on one commodity core. Inference treats each flow independently over one shared compiled circuit, so the work is embarrassingly parallel: aggregate throughput scales with the core count at no change to the circuit, and it

is the per-flow latency that bounds responsiveness rather than the single-thread rate. A busy enterprise edge opens new flows in the low thousands per second, so a few cores already clear it, and the number that reads as a ceiling on one thread is a floor that lifts with the machine. The second question is whether the cost survives a larger rule set, and here the worst case is real: knowledge compilation can yield a circuit exponential in the treewidth of the rule program's interaction graph, so a dense, heavily entangled rule base could break the memory axis before either other axis moves. The programs mined here stay far from that regime because the rules are few and read mostly local feature conditions, which is why their compiled circuits even shrink as structure is added. The method is thus practical for the compact, benign-calibrated rule layers it is built around, and a deployer who grows the rule base toward a dense expert knowledge base should expect compilation, not inference or training, to be the first axis to break.

7.6 Generalisation and portability

The transfer experiments answer a question that matters for any rule-based component: whether a rule carries detection knowledge to a different network or only encodes an artefact of the one it was mined on. The result is that rules transfer when the attack behaviour they describe is shared across the two networks, and that two construction choices decide whether that transfer is realised. The first is selecting rules that generalise well by clearing precision and recall floors on both datasets at once, which discards conjunctions based on dataset artefacts that are selective on one side and meaningless on the other. The second is referencing each quantile threshold to the benign distribution of the target network, so that a high inbound window means high relative to where the model is deployed; pointing the reference at the source benign tail measures a distance that has no meaning on a different network. With both corrections, the neuro-symbolic transfer exceeds the neural transfer in every cell of the enterprise-to-IoT sweep, and the regularised neural control stays below it, which attributes the gain to the implemented neuro-symbolic integration and not to incidental regularisation.

Another question answered by this experiment is whether local rules defined for a specific network topology can help a ported neural component adapt to the new network. The results show that local rules provide an even bigger benefit than broad generalisable ones that are ported across the topology with the model itself. In scenarios where it is plausible, defining a rule set on the new topology can help correct the incorrect decisions made by a neural classifier due to the training influence of the previous network it was deployed on. Additionally, creating rules and defining a symbolic component that holds the knowledge of the new network is less expensive than retraining the neural component from scratch, which is a significant advantage when the new network is resource-constrained and cannot afford to retrain the neural component, or when training data is scarce.

7.7 Limitations

Several limitations follow from the findings above and from the design of the study. The methodological threats to validity are stated in Section 4.6; this section collects the limitations that the results themselves expose.

Separability Boundary The symbolic layer cannot recover a family distinction that the feature view does not contain, so on datasets like UNSW-NB15, where the families overlap in flow space, the approach detects attacks but cannot reliably name them. This is a property of feature-based symbolic detection and an error that stems from the dataset itself rather than from the particular rules mined or coupling policies employed.

External Validity The system is evaluated on public benchmarks, which are labelled and reproducible but carry known artefacts and are not live networks. Some mined conditions are faithful to these captures while being specific to the environment that produced them, so a detector built from them generalises only as far as the behaviour those conditions encode. The cross-dataset experiments probe this directly but move between two public datasets, not to an operational deployment.

Limited Probabilistic Model DeepProbLog treats probabilistic facts as independent, while flow features such as duration, packet counts, and rates are correlated, so the system is best understood as a structured probabilistic classifier and not a calibrated model of network traffic. The learned rule weights are likewise tied to the dataset, taxonomy, split, and coupling of their experiment and should be re-estimated before reuse.

Bounded Threat Model The feature-ranking and calibration pipeline reduces but does not remove the need to propose candidate predicates and rule bodies, a barrier that Hakim et al. [20] flag as one of the main obstacles to neuro-symbolic adoption in cybersecurity. The robustness experiments also assume a bounded adversary who perturbs feature values within a budget and cannot move the attack-defining columns. An attacker who knows the rule set and crafts traffic to satisfy or avoid specific clauses, or who poisons the training data, is outside the scope tested here.

7.8 Future work

The limitations point to several directions. The most direct follows from the separability boundary. The product-of-experts coupling discussed in Section C.6 recovered part of the family confusion on UNSW by learning from the co-occurrence of rules, and turning that observation into a general method, one that decides when to read the rule-activation pattern as a second classifier and how to keep its rule bank rich enough, would extend the approach to the overlapping families it currently cannot separate.

A second direction is deployment under real conditions. Evaluating the hybrid assembler as a true streaming system, with incremental circuit evaluation and a bounded packet-

path leaf count, would test the real-time claim that the offline experiments can only approximate, and would show whether the inference cost measured in Section 7.5 holds under sustained traffic.

A third direction is to use more of the probabilistic semantics the framework offers. In the present system the symbolic component acts largely as a set of gates and weighted votes, and the full distribution over possible worlds is underused. Learnable probabilistic rules and explicit reasoning over competing attack explanations could let the system represent situations where several attack progressions are simultaneously plausible, which is closer to how a multi-stage campaign actually unfolds.

A fourth direction is richer symbolic context. The rules here read a single instance: a packet, a flow, or a packet-flow pair. Incorporating structural context such as network topology, host roles, and service-vulnerability relationships would let the symbolic layer reason about whether an attack is plausible against a given target, which is the kind of knowledge that correlation-based detectors use and that the present feature-only rules cannot express.

A final direction is to test the system against an adversary that knows it. A rule-aware evasion model, and an evaluation of poisoning resistance, would turn the bounded robustness result of Section 7.2 into a claim about a realistic attacker, and would show whether the stability of the symbolic layer is a genuine defensive property or an artefact of the threat model under which it was measured.

Chapter 8

Conclusion

Network intrusion detection is pulled in several directions at once. A detector has to learn from the little labelled attack traffic that public data offers, keep false alarms low enough that an analyst will trust it, remain interpretable enough that an alert can be inspected, and hold up when its inputs are noisy or adversarially perturbed. A purely neural classifier struggles on most of these axes at the same time, and a signature-based model cannot learn the patterns it was not told about. This thesis asked whether a neuro-symbolic design built on DeepProbLog, in which mined symbolic rules are coupled to a neural classifier inside one differentiable program, can do better on these axes together, and under what conditions it stops doing so.

The proposed system was evaluated in three forms, a packet-level, a flow-level, and a hybrid detector, against neural, symbolic-only, signature baselines, and state-of-the-art methods, across a low-data training sweep and several datasets, and organised around three research questions on detection efficiency, generalisation and robustness, and explanation.

The first research question (RQ1) asked how coupling mined symbolic rules to a neural classifier affects detection performance (RQ1.1) and data efficiency (RQ1.2). The symbolic layer helped most exactly where the neural classifier was weakest. With fifty labelled samples per class on CIC-IDS2017 flow data the best coupling reached a macro-F1 of 0.903 against the neural baseline's 0.672. It also matched the baseline's best score with roughly fifty times less labelled data, passing it already at ten samples per class. The gain was sharpest at the packet level, where evidence per instance is smallest, and there the guarded coupling cut the attack false-positive rate by close to an order of magnitude. As data grew, detection and exact-attack rates converged, but the headline macro-F1 stayed well apart because the gating policies keep suppressing false alarms long after raw detection has caught up. The UNSW-NB15 results marked the boundary of this contribution: where the attack families overlap in the feature space, no rule can recover a distinction the features do not hold, so the system detects attacks but cannot reliably name them.

The second research question (RQ2) asked whether symbolic knowledge improves generalisation (RQ2.1) and robustness (RQ2.2), and how the coupling mediates this. Two findings stand out. Rules transfer to a different network when the attack behaviour they describe is shared and when their thresholds are referenced to the target's own benign traffic, and under those conditions the neuro-symbolic transfer exceeded the neural transfer across the

board, though it stayed short of a model trained directly on the target. The choice of coupling turned out to be an operational lever: the strict gates that score highest on clean traffic collapse fastest under noise and evasion, while the permissive and standalone-symbolic couplings degrade gracefully, so the deployer trades clean precision for robustness by selecting where on that spectrum to sit.

The third research question (RQ3) asked whether the symbolic component yields explanations that are faithful, stable, and interpretable to an analyst (RQ3.1). The symbolic layer produced reasons that are faithful by construction and read from the same arithmetic circuit that makes the decision. Every attack alert was attributed to a named clause, the deciding clauses were accurate on held-out traffic, and the explanations were stable across retraining, agreeing on 0.964 of samples where post-hoc attributions on the neural baseline agreed on fewer than a quarter. Each champion clause mapped onto a documented MITRE ATT&CK technique, even though it came from the automated mining pipeline and not from a human author. This interpretability is bought at a real computational cost, since evaluating a compiled circuit is far slower than a single neural forward pass, but the heaviest program still runs within flow-level rates.

Taken together, the results support a measured claim. A neuro-symbolic NIDS built on DeepProbLog can be more data-efficient, lower in false alarms, more interpretable, and more robust than its neural baseline, provided the attack families are separable in the feature space the rules can read and the inference cost is acceptable for the traffic view. Where those conditions fail, on overlapping families or at strict line rate, the advantages narrow or disappear. The broader contribution is to show that the coupling between the neural and symbolic parts is itself a design surface: precision, recall, robustness, and explanation can be traded against one another by changing how the two components are wired, and making that trade explicit and measurable is what the framework adds over a fixed detector.

AI Usage

During the preparation of this work, generative AI tools were used to assist with grammar and phrasing, to help structure and refactor the writing of several chapters. All experimental design, results, and conclusions are my own. After using these tools I reviewed and edited the content, validated the outcomes, and I take full responsibility for the content of this thesis.

Bibliography

[1] Zeeshan Ahmad, Adnan Shahid Khan, Wai Shiang Cheah, Johari Abdullah, and Farhan Ahmad. Network intrusion detection system: A systematic study of machine learning and deep learning approaches. *Transactions on Emerging Telecommunications Technologies*, 32(1):e4150, 2021. https://doi.org/10.1002/ett.4150. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/ett.4150.

[2] Hashem Alaidaros, Massudi Mahmuddin, and Ali Al Mazari. An overview of flow-based and packet-based intrusion detection performance in high-speed networks. 01 2011.

[3] Afnan Alotaibi and Murad A. Rassam. Adversarial machine learning attacks against intrusion detection systems: A survey on strategies and defense. *Future Internet*, 15 (2):62, 2023. 10.3390/fi15020062.

[4] Giovanni Apruzzese, Pavel Laskov, and Johannes Schneider. Sok: Pragmatic assessment of machine learning for network intrusion detection, 2023. URL https://arxiv.org/abs/2305.00550.

[5] H. Asad, S. Adhikari, and Ilir Gashi. A perspective–retrospective analysis of diversity in signature-based open-source network intrusion detection systems. *Int. J. Inf. Secur.*, 23(2):1331–1346, December 2023. ISSN 1615-5262. 10.1007/s10207-023-00794-9. URL https://doi.org/10.1007/s10207-023-00794-9.

[6] Samy Badreddine, Artur d'Avila Garcez, Luciano Serafini, and Michael Spranger. Logic tensor networks. *Artificial Intelligence*, 303:103649, 2022. ISSN 0004-3702. https://doi.org/10.1016/j.artint.2021.103649. URL https://www.sciencedirect.com/science/article/pii/S0004370221002009.

[7] Bikram Pratim Bhuyan, Amar Ramdane-Cherif, Ravi Tomar, and TP Singh. Neuro-symbolic artificial intelligence: a survey. *Neural Computing and Applications*, 36(21): 12809–12844, 2024. 10.1007/s00521-024-09960-z.

[8] Alice Bizzarri, Brian Jalaian, Fabrizio Riguzzi, and Nathaniel D. Bastian. A neuro-symbolic artificial intelligence network intrusion detection system. In *2024 33rd International Conference on Computer Communications and Networks (ICCCN)*, pages 1–9, 2024. 10.1109/ICCCN61486.2024.10637618.

[9] Yaying Chen, Siamak Layeghy, Liam Daly Manocchio, and Marius Portmann. P4-NIDS: High-performance network monitoring and intrusion detection in P4. In *Intelligent Computing*, Lecture Notes in Networks and Systems, pages 355–373. Springer Nature Switzerland, 2025. 10.1007/978-3-031-92611-2_24. arXiv:2411.17987.

[10] Alessandro Daniele, Tommaso Campari, Sagar Malhotra, and Luciano Serafini. Deep symbolic learning: Discovering symbols and rules from perceptions. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence*, IJCAI-2023, page 3597–3605. International Joint Conferences on Artificial Intelligence Organization, August 2023. 10.24963/ijcai.2023/400. URL http://dx.doi.org/10.24963/ijcai.2023/400.

[11] Adnan Darwiche and Pierre Marquis. A knowledge compilation map. *Journal of Artificial Intelligence Research*, 17:229–264, 2002. 10.1613/jair.989.

[12] Luc De Raedt, Angelika Kimmig, and Hannu Toivonen. Problog: a probabilistic prolog and its application in link discovery. In *Proceedings of the 20th International Joint Conference on Artifical Intelligence*, IJCAI'07, page 2468–2473, San Francisco, CA, USA, 2007. Morgan Kaufmann Publishers Inc.

[13] Mohamed El-Sherif, Ahmed Khattab, and Magdy El-Soudani. Intrusion detection using TCP/IP single packet header binary image for IoT networks. *Cybersecurity*, 8 (1):104, 2025. 10.1186/s42400-025-00441-x.

[14] ENISA. ENISA Threat Landscape 2025. Technical report, European Union Agency for Cybersecurity (ENISA), oct 2025. URL https://www.enisa.europa.eu/publications/enisa-threat-landscape-2025.

[15] Kaniz Farhana, Maqsudur Rahman, and Md. Tofael Ahmed. An intrusion detection system for packet and flow based networks using deep neural network approach. *International Journal of Electrical and Computer Engineering (IJECE)*, 10:5514–5525, 10 2020. 10.11591/ijece.v10i5.pp5514-5525.

[16] Fox-IT. Report on the Investigation into the Cyber Attack on Maastricht University. Technical report, Fox-IT and Maastricht University, feb 2020. URL https://www.maastrichtuniversity.nl/file/foxitrapportreactieuniversiteitmaastrichtnl10-02pdf.

[17] Google Threat Intelligence Group. Hello 0-Days, My Old Friend: A 2024 Zero-Day Exploitation Analysis. Technical report, Google and Mandiant, 2025. URL https://cloud.google.com/blog/topics/threat-intelligence/2024-zero-day-trends.

[18] Gudmund Grov, Jonas Halvorsen, Magnus Wiik Eckhoff, Bjørn Jervell Hansen, Martin Eian, and Vasileios Mavroeidis. On the use of neurosymbolic ai for defending against cyber attacks, 2024. URL https://arxiv.org/abs/2408.04996.

[19] Xin Fan Guo, Xinran Zheng, Albert Merono Penuela, Lorenzo Cavallaro, Sergio Maffeis, and Fabio Pierazzi. KnowML: Improving generalization of ML-NIDS with attack knowledge graphs. In *IEEE European Symposium on Security and Privacy (EuroS&P)*, 2026. URL https://arxiv.org/abs/2506.19802.

[20] Safayat Bin Hakim, Muhammad Adil, Alvaro Velasquez, Shouhuai Xu, and Houbing Herbert Song. Neuro-symbolic ai for cybersecurity: State of the art, challenges, and opportunities, 2025. URL https://arxiv.org/abs/2509.06921.

[21] Swetha Hariharan, R. R. Rejimol Robinson, Rendhir R. Prasad, Ciza Thomas, and N. Balakrishnan. Xai for intrusion detection system: comparing explanations based on global and local scope. *Journal of Computer Virology and Hacking Techniques*, 19, 07 2022. 10.1007/s11416-022-00441-2.

[22] Mohammad Mehedi Hassan, Abdu Gumaei, Ahmed Alsanad, Majed Alrubaian, and Giancarlo Fortino. A hybrid deep learning model for efficient intrusion detection in big data environment. *Information Sciences*, 513:386–396, 2020. 10.1016/j.ins.2019.10.069.

[23] IBM Security. Cost of a Data Breach Report 2025. Technical report, IBM Security, 2025. URL https://www.ibm.com/reports/data-breach.

[24] Chathuranga Sampath Kalutharage, Xiaodong Liu, and Christos Chrysoulas. Neurosymbolic learning and domain knowledge-driven explainable ai for enhanced iot network attack detection and response. *Computers & Security*, 151:104318, 2025. ISSN 0167-4048. https://doi.org/10.1016/j.cose.2025.104318. URL https://www.sciencedirect.com/science/article/pii/S0167404825000070.

[25] Henry Kautz. The third ai summer: Aaai robert s. engelmore memorial lecture. *AI Magazine*, 43(1):105–125, Mar. 2022. 10.1002/aaai.12036. URL https://ojs.aaai.org/aimagazine/index.php/aimagazine/article/view/19122.

[26] Ansam Khraisat, Iqbal Gondal, Peter Vamplew, and Joarder Kamruzzaman. Survey of intrusion detection systems: techniques, datasets and challenges. *Cybersecurity*, 2, 12 2019. 10.1186/s42400-019-0038-7.

[27] Aklil Kiflay, Athanasios Tsokanos, Mahmood Fazlali, and Raimund Kirner. Network intrusion detection leveraging multimodal features. *Array*, 22:100349, 2024. ISSN 2590-0056. https://doi.org/10.1016/j.array.2024.100349. URL https://www.sciencedirect.com/science/article/pii/S2590005624000158.

[28] Ayush Kumar and Vrizlynn L. L. Thing. Evaluating the explainability of state-of-the-art deep learning-based network intrusion detection systems, 2025. URL https://arxiv.org/abs/2408.14040.

[29] Ayush Kumar, Kar Wai Fok, and Vrizlynn L.L. Thing. Expids: A drift-adaptable network intrusion detection system with improved explainability. In *2024 34th International Telecommunication Networks and Applications Conference (ITNAC)*, pages 1–7, 2024. 10.1109/ITNAC62915.2024.10815461.

[30] Jinghong Lan, Xudong Liu, Bo Li, Jie Sun, Beibei Li, and Jun Zhao. Member: A multi-task learning model with hybrid deep features for network intrusion detection. *Computers & Security*, 123:102919, 2022. ISSN 0167-4048. https://doi.org/10.1016/j.cose.2022.102919. URL https://www.sciencedirect.com/science/article/pii/S016740482200311X.

[31] Robin Manhaeve, Sebastijan Dumančić, Angelika Kimmig, Thomas Demeester, and Luc De Raedt. Neural probabilistic logic programming in deepproblog. *Artificial Intelligence*, 298:103504, 2021. ISSN 0004-3702. https://doi.org/10.1016/j.artint.2021.103504. URL https://www.sciencedirect.com/science/article/pii/S0004370221000552.

[32] Ziadoon K. Maseer, Qusay Kanaan Kadhim, Baidaa Al-Bander, Robiah Yusof, and Abdu Saif. Meta-analysis and systematic review for anomaly network intrusion detection systems: Detection methods, dataset, validation methodology, and challenges. *IET Networks*, 13(5-6):339–376, 2024. https://doi.org/10.1049/ntw2.12128. URL https://ietresearch.onlinelibrary.wiley.com/doi/abs/10.1049/ntw2.12128.

[33] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space, 2013. URL https://arxiv.org/abs/1301.3781.

[34] Pasquale Minervini, Sebastian Riedel, Pontus Stenetorp, Edward Grefenstette, and Tim Rocktäschel. Learning reasoning strategies in end-to-end differentiable proving, 2020. URL https://arxiv.org/abs/2007.06477.

[35] Yisroel Mirsky, Tomer Doitshman, Yuval Elovici, and Asaf Shabtai. Kitsune: An ensemble of autoencoders for online network intrusion detection. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2018. 10.14722/ndss.2018.23204.

[36] Jose Carlos Mondragon, Paula Branco, Guy-Vincent Jourdan, Andres Eduardo Gutierrez-Rodriguez, and Rajesh Roshan Biswal. Advanced ids: a comparative study of datasets and machine learning algorithms for network flow-based intrusion detection systems. *Applied Intelligence*, 55(7):608, 2025. 10.1007/s10489-025-06422-4.

[37] 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, 2015. 10.1109/MilCIS.2015.7348942.

[38] Jeffrey Pennington, Richard Socher, and Christopher Manning. GloVe: Global vectors for word representation. In Alessandro Moschitti, Bo Pang, and Walter Daelemans, editors, *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar, October 2014. Association for Computational Linguistics. 10.3115/v1/D14-1162. URL https://aclanthology.org/D14-1162/.

[39] Weicheng Qiu, Yinghua Ma, Xiuzhen Chen, Haiyang Yu, and Lixing Chen. Hybrid intrusion detection system based on dempster-shafer evidence theory. *Computers & Security*, 117:102709, 2022. ISSN 0167-4048. https://doi.org/10.1016/j.cose.2022.102709. URL https://www.sciencedirect.com/science/article/pii/S0167404822001079.

[40] Mohanad Sarhan, Siamak Layeghy, and Marius Portmann. Evaluating standard feature sets towards increased generalisability and explainability of ml-based network intrusion detection. *Big Data Research*, 30:100359, 2022. ISSN 2214-5796. https://doi.org/10.1016/j.bdr.2022.100359. URL https://www.sciencedirect.com/science/article/pii/S2214579622000533.

[41] 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, 2018. 10.5220/0006639801080116.

[42] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016. 10.1038/nature16961. URL http://www.nature.com/nature/journal/v529/n7587/full/nature16961.html.

[43] 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, 2010. 10.1109/SP.2010.25.

[44] Didik Sudyana, Fietyata Yudha, Ying-Dar Lin, Chia-Hung Lai, Po-Ching Lin, and Ren-Hung Hwang. From flow to packet: A unified machine learning approach for advanced intrusion detection. *Security and Communication Networks*, 2025(1):5729035, 2025. https://doi.org/10.1155/sec/5729035. URL https://onlinelibrary.wiley.com/doi/abs/10.1155/sec/5729035.

[45] Shahroz Tariq, Mohan Baruwal Chhetri, Surya Nepal, and Cecile Paris. Alert fatigue in security operations centres: Research challenges and opportunities. *ACM Computing Surveys*, 57(9):1–38, 2025. 10.1145/3723158.

[46] Verizon. 2025 Data Breach Investigations Report. Technical report, Verizon Business, 2025. URL https://www.verizon.com/business/resources/reports/2025-dbir-data-breach-investigations-report.pdf.

[47] Yaxuan Wang. Deep learning-based network intrusion detection systems. *Applied and Computational Engineering*, 109:179–188, 12 2024. 10.54254/2755-2721/2024.18104.

[48] Jiang Xie, Shuhao Li, Yongzheng Zhanga, Peishuai Sun, and Hongbo Xu. Analysis and detection against network attacks in the overlapping phenomenon of behavior attribute, 2023. URL https://arxiv.org/abs/2310.10660.

[49] Chao Zha, Zhiyu Wang, Yifei Fan, Bing Bai, Yinjie Zhang, Sainan Shi, and Ruyun Zhang. A-nids: Adaptive network intrusion detection system based on clustering and stacked ctgan. *IEEE Transactions on Information Forensics and Security*, 20:3204–3219, 2025. 10.1109/TIFS.2025.3551643.

[50] Zeinab Zoghi and Gursel Serpen. Building an intrusion detection system on UNSW-NB15: Reducing the margin of error to deal with data overlap and imbalance. *Concurrency and Computation: Practice and Experience*, 36(19):e8242, 2024. 10.1002/cpe.8242.

[51] Zeinab Zoghi and Gursel Serpen. UNSW-NB15 computer security dataset: Analysis through visualization. *Security and Privacy*, 7(1):e331, 2024. 10.1002/spy2.331. arXiv:2101.05067.

Appendix A

Glossary

This appendix gives an overview of frequently used terms, abbreviations, and dataset names.

NIDS: Network Intrusion Detection System — a system that inspects network traffic and flags malicious activity.

NeSy: Neuro-symbolic — a model that couples a neural classifier with a symbolic (logic-rule) layer.

DeepProbLog (DPL): A neural probabilistic-logic framework in which neural networks supply probabilities to atoms of a probabilistic logic program; inference is performed by weighted model counting over a compiled arithmetic circuit.

MLP: Multilayer perceptron — the feed-forward neural baseline used throughout.

AC: Arithmetic circuit — the sum/product DAG to which a ground DeepProbLog program is compiled.

SDD / d-DNNF: Sentential Decision Diagram / deterministic Decomposable Negation Normal Form — the tractable circuit classes on which weighted model counting is linear in circuit size.

WMC: Weighted Model Counting — the inference operation evaluated on the compiled circuit.

Coupling: The policy that fuses the neural and symbolic decisions in the `classify` head (anchored, same-class boost, guarded, hard-gate, safety-net).

Anchor / rescue: A high-precision rule that may flag an attack on its own (anchor) versus a rule that fires only when the neural component already predicts the attack (rescue).

Macro-F1: Unweighted mean of the per-class F1 scores.

DR: Detection rate — recall on the attack classes.

FPR: Attack false-positive rate — the fraction of benign flows flagged as attacks.

CIC-IDS2017: The Canadian Institute for Cybersecurity Intrusion Detection 2017 flow dataset.

CSE-CIC-IDS2018: The Communications Security Establishment / CIC 2018 intrusion-detection dataset.

NetFlow-v3 (NF-v3): The standardised NetFlow-v3 feature schema (NF-CSE-CIC-IDS2018-v3, NF-ToN-IoT-v3, NF-UNSW-NB15-v3) that re-expresses several datasets on one common feature set.

UNSW-NB15: The University of New South Wales 2015 intrusion-detection dataset.

ToN-IoT: The Telemetry of Network and IoT intrusion-detection dataset used for the portability study.

Kitsune / KitNET: An unsupervised autoencoder-ensemble intrusion detector and its core learner, used as a state-of-the-art baseline.

Treewidth: A structural width measure of the program's interaction graph that governs the worst-case size of the compiled circuit.

Appendix B

Implementation details

B.1 Packet-level feature set

Table B.1 lists the packet-header features extracted from the NF-CSE-CIC-IDS2018-v3 capture and supplied to the packet-level system of Section 6.2.1. Features are read directly from packet headers at parsing time, with no payload inspection, so the detector stays applicable to encrypted traffic. The neural classifier receives the full vector; symbolic predicates read selected interpretable features from the same representation. Flow-identifier and label columns (`flow_id`, `ip.src`, `ip.dst`, `frame.time_epoch`, `Label`, `Attack`, `label_binary`, `pcap_file`) are excluded from the model input and kept only for flow reconstruction, labelling, and leakage control.

Table B.1: Packet-header feature set extracted for the packet-level model on NF-CSE-CIC-IDS2018-v3.

Feature	Description	Type
ip.proto	IP protocol number (TCP / UDP / ICMP / . . .)	categorical
ip.ttl	IP time-to-live	numeric
ip.dsfield	IP differentiated-services field (DSCP / ECN)	categorical
ip.len	IP total length (bytes)	numeric
frame.len	Captured frame length (bytes)	numeric
frame.time_delta	Inter-packet time since the previous frame	numeric
srcport	Transport-layer source port	numeric
dstport	Transport-layer destination port	categorical
tcp.flags	TCP flag bitfield (SYN, ACK, FIN, . . .)	categorical
tcp.window_size_value	TCP advertised window size	numeric
tcp.hdr_len	TCP header length (bytes)	numeric
tcp.seq	TCP sequence number	numeric
tcp.ack	TCP acknowledgement number	numeric
tcp.len	TCP payload length (bytes)	numeric
udp.length	UDP datagram length (bytes)	numeric
icmp.type	ICMP message type	categorical
icmp.code	ICMP message code	categorical
dns.id	DNS transaction identifier	numeric
dns.qry.type	DNS query type	categorical
dns.resp.ttl	DNS response record time-to-live	numeric
ftp.response.code	FTP server response code	categorical
frame.protocols	Protocol-stack string (e.g. eth:ip:tcp:http)	categorical

B.2 Symbolic-rule mining algorithm

Algorithm 1 gives the full offline rule-mining pipeline described in Section 5.4, with the concrete thresholds used in the experiments. The four stages, scoring, calibration, program assembly, and soft rewrite, run entirely on the training pool, which doubles as the scoring and reference pool, and the validation split, so the test set never participates in rule construction.

Algorithm 1 Offline symbolic-rule mining

Require: training/scoring pool S, validation split V, attack classes $\mathcal{A}$, top count $K = 12$, statistic weights w

Ensure: DeepProbLog program Π

$\mathcal{R} \leftarrow \emptyset$
for all attack $a \in \mathcal{A}$ **do**
  for all candidate feature f eligible for the symbolic vocabulary **do**
    compute KS, Cliff's δ, Hellinger, folded AUC (numeric) or Cramer's V, TVD, categorical AUC (categorical) on S for a versus benign
    score$(f, a) \leftarrow$ weighted mean of the per-attack min-max normalised statistics, using w
  end for
  $F_a \leftarrow$ top-K features of a by score$(\cdot, a)$, each carrying its predicate hint (quantile tier or category)
  $C_a \leftarrow$ candidate rule bodies: each top feature alone, plus short conjunctions of their hint predicates
  for all candidate body $c \in C_a$ **do**
    measure precision p, recall r, benign FPR ϕ of c on V; measure $p_{\text{tr}}, \phi_{\text{tr}}$ on S
    stable $\leftarrow (p_{\text{tr}} - p \leq 0.20) \wedge (\phi_{\text{tr}} - \phi \leq 0.05)$ {drop rules selective only on mining data}
    if *stable* $\wedge\, p \geq 0.90 \wedge \phi \leq 0.015$ **then**
      role$(c) \leftarrow$ ANCHOR {may support a on its own}
    else if *stable* $\wedge\, r \geq 0.35 \wedge \phi \leq 0.050$ **then**
      role$(c) \leftarrow$ RESCUE {fires only behind the neural prediction}
    else if *stable* $\wedge\, p \geq 0.90 \wedge \phi \leq 0.020$ on a rival class **then**
      role$(c) \leftarrow$ DISQUALIFIER
    else
      discard c
    end if
    $\mathcal{R} \leftarrow \mathcal{R} \cup \{c\}$
  end for
end for
group each class's anchors and rescues into per-class pockets; wrap pockets in learnable predicates $t(p) :: symbolic_a_anchor(X)$ whose initial p is the calibrated confidence
attach the query head $classify/2$ for the configured coupling; emit hard externals
rewrite each hard external to its soft module (Section 5.5) and split multi-threshold bodies into single-external helpers
return Π

Appendix C

Extended results

This appendix collects the full per-coupling sweeps summarised in Chapter 6. Each multiclass table reports, for every coupling and training size k, the accuracy, attack false-positive rate, exact-attack rate, detection rate, macro-F1, and weighted-F1, averaged over five replicates. The sections follow the order of the results chapter: the binary attack-specific sweep; the flow-level results on the original CIC-IDS2017 and the standardised NetFlow-v3 datasets; the standardised packet-level and hybrid-level results; the UNSW-NB15 results, original and standardised; the hard-versus-soft threshold comparison; and a theoretical complexity analysis.

C.1 Binary attack-specific flow results

Table C.1 gives the full binary attack-versus-benign sweep summarised in Section 6.2.1. For each family and system it reports macro-F1 at every training size, with the detection rate and attack false-positive rate at $k = 50$. Families marked with † (PortScan and WebAttack) received no rule that cleared the validation precision floor, so their anchored and tiered rows reduce to the neural path and differ from the baseline only through training variance.

C.2 CIC-IDS2017 original flow-level results

Table C.2 gives the full multiclass sweep on the original CIC-IDS2017 flow dataset summarised in Section 6.2.1.

C.3 Standardised NetFlow-v3 CSE-CIC-IDS2018 flow-level results

Table C.3 gives the full per-coupling sweep for the flow view of the standardised NetFlow-v3 CSE-CIC-IDS2018 dataset, summarised in Section 6.2.1: the five couplings against the neural and symbolic baselines, with accuracy, attack false-positive rate, exact-attack rate, detection rate, macro-F1 and weighted-F1 averaged over five replicates.

Family	System	F1@5	F1@10	F1@25	F1@50	DR@50	FPR@50
CIC-IDS2017							
bot	MLP	0.598	0.576	0.588	0.609	0.96	0.045
	anchored	0.873	0.873	0.873	0.873	0.65	0.001
	tiered	0.873	0.873	0.873	0.873	0.65	0.001
bruteforce	MLP	0.594	0.634	0.694	0.630	0.99	0.069
	anchored	0.762	0.801	0.804	0.795	0.52	0.003
	tiered	0.766	0.799	0.800	0.808	0.52	0.002
dos	MLP	0.834	0.884	0.937	0.933	0.93	0.030
	anchored	0.861	0.861	0.864	0.864	0.63	0.003
	tiered	0.746	0.746	0.748	0.752	0.65	0.099
portscan†	MLP	0.908	0.898	0.915	0.903	0.99	0.032
	anchored	0.901	0.917	0.944	0.930	0.98	0.021
	tiered	0.897	0.898	0.926	0.926	0.98	0.022
webattack†	MLP	0.691	0.692	0.696	0.706	0.99	0.020
	anchored	0.742	0.734	0.788	0.709	0.98	0.019
	tiered	0.683	0.733	0.769	0.758	0.96	0.013
UNSW-NB15							
analysis	MLP	0.742	0.809	0.787	0.814	0.99	0.013
	anchored	0.883	0.915	0.906	0.931	0.78	0.000
	tiered	0.835	0.877	0.895	0.903	0.81	0.002
backdoor	MLP	0.758	0.784	0.784	0.779	1.00	0.018
	anchored	0.917	0.890	0.938	0.903	0.86	0.004
	tiered	0.846	0.884	0.880	0.877	0.90	0.006
dos	MLP	0.644	0.776	0.787	0.767	0.95	0.021
	anchored	0.604	0.604	0.604	0.604	0.12	0.000
	tiered	0.604	0.604	0.604	0.604	0.12	0.000
exploits	MLP	0.780	0.833	0.860	0.867	0.96	0.016
	anchored	0.822	0.844	0.874	0.884	0.90	0.010
	tiered	0.788	0.833	0.873	0.871	0.88	0.012
fuzzers	MLP	0.800	0.830	0.827	0.830	0.97	0.020
	anchored	0.829	0.837	0.827	0.847	0.94	0.016
	tiered	0.814	0.828	0.833	0.822	0.95	0.021
generic	MLP	0.816	0.810	0.846	0.851	0.98	0.019
	anchored	0.884	0.876	0.888	0.882	0.92	0.011
	tiered	0.828	0.849	0.873	0.885	0.94	0.012
reconnaissance	MLP	0.807	0.817	0.842	0.853	0.98	0.013
	anchored	0.761	0.762	0.779	0.788	0.53	0.005
	tiered	0.826	0.832	0.857	0.853	0.98	0.014

Table C.1: Binary attack-versus-benign flow sweep on the original CIC-IDS2017 and UNSW-NB15 datasets. Macro-F1 at $k \in \{5, 10, 25, 50\}$, with detection rate and attack false-positive rate at $k = 50$, averaged over three replicates. Systems are the binary MLP baseline, the anchored neuro-symbolic detector (crisp quantile evidence), and the tiered neuro-symbolic detector (learnable four-band ladder).

C.4 Standardised NetFlow-v3 CSE-CIC-IDS2018 packet-level results

Table C.4 gives the full per-coupling sweep for the packet view, the five couplings against the neural and symbolic baselines on the bot roster (benign, bot, brute force, DDoS, DoS), with the same six metrics averaged over five replicates.

Architecture	k	Accuracy	FPR	Exact Attack	Detection Rate	Macro-F1	Weighted-F1
NN baseline	5	0.799	0.180	0.719	0.746	0.530	0.832
NN baseline	10	0.821	0.174	0.801	0.810	0.574	0.850
NN baseline	25	0.886	0.115	0.888	0.895	0.651	0.904
NN baseline	50	0.899	0.116	0.955	0.959	0.672	0.917
NN baseline	500	0.960	0.046	0.982	0.986	0.817	0.964
Anchored	5	0.912	0.037	0.728	0.733	0.773	0.909
Anchored	10	0.947	0.042	0.905	0.907	**0.840**	**0.949**
Anchored	25	0.958	0.035	0.931	0.932	**0.883**	**0.959**
Anchored	50	0.965	0.030	0.949	0.950	0.899	**0.966**
Anchored	500	0.982	0.014	0.967	0.968	0.937	**0.982**
Same-class boost	5	0.839	0.140	0.764	0.772	0.605	0.855
Same-class boost	10	0.873	0.111	0.815	0.822	0.633	0.887
Same-class boost	25	0.909	0.080	0.870	0.876	0.704	0.917
Same-class boost	50	0.933	0.072	0.952	0.955	0.765	0.939
Same-class boost	500	0.948	0.047	0.928	0.930	0.820	0.951
Guarded	5	0.918	0.033	0.743	0.747	**0.781**	0.918
Guarded	10	0.946	0.027	0.848	0.850	0.824	0.947
Guarded	25	0.954	0.034	0.912	0.914	0.876	0.955
Guarded	50	0.965	0.032	0.953	0.953	**0.903**	0.966
Guarded	500	0.979	0.019	0.975	0.976	**0.942**	0.980
Hard gate	5	0.890	0.029	0.596	0.599	0.703	0.882
Hard gate	10	0.919	0.032	0.743	0.744	0.799	0.918
Hard gate	25	0.944	0.034	0.861	0.862	0.865	0.944
Hard gate	50	0.963	0.033	0.946	0.946	0.902	0.963
Hard gate	500	0.973	0.026	0.969	0.969	0.929	0.974
Safety net	5	0.935	0.011	0.740	0.740	0.690	**0.926**
Safety net	10	0.931	0.021	0.755	0.757	0.733	0.925
Safety net	25	0.933	0.030	0.796	0.804	0.726	0.930
Safety net	50	0.938	0.049	0.891	0.894	0.794	0.942
Safety net	500	0.927	0.078	0.946	0.950	0.793	0.934
Symbolic baseline	–	0.914	0.058	0.813	0.814	0.703	0.912
Suricata baseline	–	0.885	0.029	0.570	0.570	0.311	0.876

Table C.2: CICIDS flow-level multiclass results. Bold indicates the highest Macro-F1 and Weighted-F1 for each training size k.

C.5 Standardised NetFlow-v3 CSE-CIC-IDS2018 hybrid-level results

Table C.5 gives the hybrid view, a fusion comparison between a neural late-fusion baseline, a symbolic-only model, and the neuro-symbolic hybrid, since the coupling axis does not transfer to two views.

Architecture	k	Accuracy	FPR	Exact Attack	Detection Rate	Macro-F1	Weighted-F1
NN baseline	5	0.904	0.084	0.822	0.908	0.697	0.918
NN baseline	10	0.952	0.039	0.886	0.939	0.794	0.955
NN baseline	25	0.972	0.019	0.912	0.985	0.857	0.973
NN baseline	50	0.980	0.014	0.938	0.989	0.881	0.980
NN baseline	500	0.990	0.003	0.944	0.992	0.922	0.990
Anchored	5	0.951	0.032	0.835	0.927	0.779	0.955
Anchored	10	0.954	0.038	0.898	0.949	0.826	0.958
Anchored	25	0.976	0.017	0.931	0.980	0.876	0.977
Anchored	50	0.982	0.009	0.919	0.989	0.887	0.982
Anchored	500	0.991	0.002	0.946	0.994	0.926	0.991
Same-class boost	5	0.921	0.062	0.800	0.905	0.701	0.928
Same-class boost	10	0.938	0.053	0.872	0.938	0.766	0.944
Same-class boost	25	0.966	0.026	0.914	0.988	0.842	0.968
Same-class boost	50	0.982	0.012	0.940	0.990	0.883	0.982
Same-class boost	500	0.991	0.002	0.947	0.997	0.928	0.991
Guarded	5	0.971	0.012	0.854	0.908	0.813	0.968
Guarded	10	0.980	0.004	0.873	0.925	0.866	0.978
Guarded	25	0.985	0.004	0.907	0.959	0.886	0.983
Guarded	50	0.987	0.003	0.916	0.964	**0.894**	0.986
Guarded	500	0.991	0.002	0.941	0.989	0.925	0.991
Hard gate	5	0.975	0.006	0.844	0.896	0.820	0.971
Hard gate	10	0.980	0.005	0.878	0.927	0.869	0.978
Hard gate	25	0.985	0.005	0.910	0.962	0.887	0.983
Hard gate	50	0.987	0.003	0.918	0.968	**0.895**	0.985
Hard gate	500	0.992	0.000	0.938	0.986	0.927	0.991
Safety net	5	0.922	0.020	0.522	0.629	0.683	0.918
Safety net	10	0.922	0.023	0.543	0.683	0.700	0.921
Safety net	25	0.932	0.017	0.576	0.789	0.693	0.931
Safety net	50	0.947	0.006	0.619	0.784	0.744	0.941
Safety net	500	0.956	0.003	0.675	0.840	0.763	0.951
Symbolic baseline	5	0.968	0.026	0.922	0.981	0.858	0.968
Symbolic baseline	10	0.948	0.014	0.682	0.734	0.794	0.939
Symbolic baseline	25	0.961	0.022	0.847	0.904	0.840	0.959
Symbolic baseline	50	0.954	0.019	0.772	0.827	0.821	0.950
Symbolic baseline	500	0.950	0.014	0.697	0.749	0.806	0.942

Table C.3: Standardised NetFlow-v3 CSE-CIC-IDS2018 flow-level multiclass results across training size. Bold marks the highest macro-F1 per k.

C.6 UNSW-NB15 original flow-level results

Table C.6 gives the full per-coupling sweep on the original UNSW-NB15 flow dataset summarised in Section 6.2.1. All systems share the same low macro-F1 band because the attack families overlap in the flow feature space; the detection rate is high while the exact-attack rate stays low, which shows the systems detect attacks but cannot separate the families.

Figure C.1 shows the macro-F1 and attack false-positive heatmap and Figure C.2 the detection-rate and exact-attack bubble grids for the same systems; the high detection rate beside the low exact-attack rate is the signature of the family overlap discussed in Section 6.2.1.

Product-of-experts on the overlapping families. The structural couplings of Section 4.3 all track the neural baseline on UNSW-NB15 because a single gated rule expresses one condition per class, which the overlap defeats. The product-of-experts coupling instead

Architecture	k	Accuracy	FPR	Exact Attack	Detection Rate	Macro-F1	Weighted-F1
NN baseline	5	0.854	0.145	0.839	0.898	0.518	0.889
NN baseline	10	0.895	0.101	0.864	0.930	0.579	0.922
NN baseline	25	0.907	0.091	0.897	0.955	0.615	0.933
NN baseline	50	0.922	0.075	0.899	0.948	0.632	0.943
NN baseline	500	0.967	0.031	0.951	0.992	0.776	0.974
Anchored	5	0.967	0.027	0.922	0.931	0.860	0.974
Anchored	10	0.969	0.026	0.931	0.946	0.847	0.975
Anchored	25	0.963	0.036	0.961	0.973	0.868	0.973
Anchored	50	0.963	0.037	0.959	0.973	0.864	0.972
Anchored	500	0.984	0.012	0.950	0.992	0.883	0.986
Same-class boost	5	0.883	0.113	0.849	0.914	0.529	0.906
Same-class boost	10	0.879	0.121	0.876	0.946	0.570	0.911
Same-class boost	25	0.893	0.106	0.891	0.962	0.594	0.924
Same-class boost	50	0.924	0.071	0.891	0.943	0.644	0.944
Same-class boost	500	0.963	0.035	0.940	0.981	0.758	0.971
Guarded	5	0.986	0.002	0.894	0.894	**0.878**	0.985
Guarded	10	0.986	0.005	0.915	0.915	**0.897**	0.986
Guarded	25	0.988	0.005	0.932	0.933	**0.925**	0.988
Guarded	50	0.987	0.008	0.943	0.947	**0.924**	0.988
Guarded	500	0.992	0.002	0.941	0.951	**0.940**	0.992
Hard gate	5	0.976	0.005	0.821	0.837	0.712	0.973
Hard gate	10	0.978	0.007	0.857	0.878	0.769	0.977
Hard gate	25	0.980	0.009	0.890	0.909	0.817	0.980
Hard gate	50	0.983	0.005	0.889	0.913	0.823	0.983
Hard gate	500	0.991	0.001	0.922	0.936	0.914	0.990
Safety net	5	0.981	0.006	0.882	0.886	0.842	0.980
Safety net	10	0.969	0.023	0.908	0.918	0.840	0.974
Safety net	25	0.969	0.028	0.951	0.964	0.882	0.977
Safety net	50	0.969	0.030	0.959	0.973	0.876	0.976
Safety net	500	0.988	0.008	0.953	0.993	0.894	0.990
Symbolic baseline	5	0.945	0.047	0.880	0.919	0.708	0.956
Symbolic baseline	10	0.950	0.040	0.871	0.908	0.715	0.959
Symbolic baseline	25	0.946	0.047	0.889	0.926	0.726	0.957
Symbolic baseline	50	0.946	0.046	0.887	0.923	0.725	0.957
Symbolic baseline	500	0.964	0.026	0.884	0.893	0.812	0.970

Table C.4: Standardised NetFlow-v3 CSE-CIC-IDS2018 packet-level multiclass results across training size. Bold marks the highest macro-F1 per k.

treats the symbolic side as a second learnable classifier over the rule-activation pattern, the set of rules that fired for a sample, and multiplies its class distribution with the neural one, so the co-occurrence of several rules can carry information about which overlapping family is present. On the original UNSW-NB15 flow data at $k = 50$ this is the one variant whose improvement over the neural baseline survives a significance test, reaching a macro-F1 of 0.600 and an exact-attack rate of 0.610 against the baseline's 0.546 and 0.570, and it also exceeds a gradient-boosting classifier trained on the full labelled set. Figure C.3 shows the comparison. The recovery is real but bounded: on the standardised NetFlow-v3 version the pipeline mines only about ten numeric rules, and on that thinner activation pattern the product-of-experts falls back to the neural level, so the ceiling remains set by how separable the families are in the features the rules can read.

System	k	Accuracy	FPR	Exact Attack	Detection Rate	Macro-F1	Weighted-F1
Neural late-fusion	5	0.798	0.127	0.183	0.246	0.355	0.802
Neural late-fusion	10	0.822	0.105	0.229	0.316	0.390	0.820
Neural late-fusion	25	0.875	0.074	0.465	0.493	0.549	0.877
Neural late-fusion	50	0.895	0.068	0.595	0.707	0.620	0.904
Neural late-fusion	500	0.993	0.005	0.977	0.988	0.941	0.993
NeSy hybrid	5	0.946	0.049	0.904	0.944	0.734	0.959
NeSy hybrid	10	0.958	0.038	0.928	0.957	0.755	0.966
NeSy hybrid	25	0.966	0.031	0.943	0.968	0.804	0.972
NeSy hybrid	50	0.978	0.019	0.954	0.977	**0.869**	0.982
NeSy hybrid	500	0.995	0.001	0.967	0.980	**0.971**	0.995
Symbolic-only	5	0.977	0.000	0.792	0.792	0.696	0.970
Symbolic-only	10	0.977	0.000	0.792	0.792	0.696	0.970
Symbolic-only	25	0.977	0.000	0.792	0.792	0.696	0.970
Symbolic-only	50	0.977	0.000	0.792	0.792	0.696	0.970
Symbolic-only	500	0.945	0.040	0.823	0.832	0.712	0.955

Table C.5: Standardised NetFlow-v3 CSE-CIC-IDS2018 hybrid-level results across training size, the fusion comparison between the neural late-fusion baseline, the symbolic-only model, and the neuro-symbolic hybrid. Bold marks the highest macro-F1 per k.

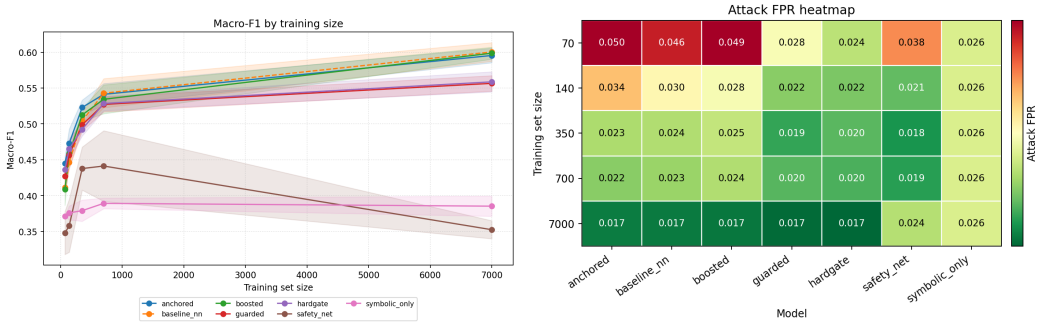


Figure C.1: UNSW-NB15 flow multiclass results for the five couplings and the neural baseline, which share the same low macro-F1 band because the attack families overlap in flow feature space.

C.7 Standardised UNSW-NB15 NetFlow-v3 results

The candidate rule generation pipeline used for CIC-IDS2017 was applied to the standardised NetFlow-v3 version of UNSW-NB15 at all three traffic views. Rules were scored on a training pool and accepted on a separate validation split, with the test set held out. The results reproduce the limitation of Section 6.2.1, and they show where it comes from at each view.

At the flow level the pipeline produced no usable rule. Once the capture timestamp features were excluded, which only separate attacks because the attack traffic was recorded in distinct time windows, no candidate reached the precision needed to be kept for any of the ten families. The best non-temporal rule per family ranged from 0.01 to 0.48 precision, all below the acceptance floor. With no rule to add, the neuro-symbolic flow model reduces to the neural baseline, and its macro-F1 stays in the same low band, near 0.38 at $k = 50$ and

Architecture	k	Accuracy	FPR	Exact Attack	Detection Rate	Macro-F1	Weighted-F1
NN baseline	5	0.894	0.046	0.413	0.934	0.412	0.904
NN baseline	10	0.912	0.030	0.445	0.956	0.446	0.917
NN baseline	25	0.925	0.024	0.515	0.972	0.504	0.929
NN baseline	50	0.931	0.023	0.561	0.986	0.543	0.936
NN baseline	500	0.944	0.017	0.636	0.998	**0.600**	0.947
Anchored	5	0.894	0.050	0.445	0.931	0.445	0.906
Anchored	10	0.913	0.034	0.489	0.949	0.473	0.919
Anchored	25	0.927	0.023	0.534	0.977	0.523	0.932
Anchored	50	0.934	0.022	0.578	0.987	0.541	0.937
Anchored	500	0.945	0.017	0.643	0.998	0.595	0.947
Same-class boost	5	0.889	0.049	0.398	0.916	0.409	0.901
Same-class boost	10	0.914	0.028	0.451	0.943	0.458	0.919
Same-class boost	25	0.926	0.025	0.528	0.980	0.513	0.931
Same-class boost	50	0.930	0.024	0.561	0.991	0.534	0.935
Same-class boost	500	0.945	0.017	0.639	0.998	0.599	0.947
Guarded	5	0.912	0.028	0.435	0.905	0.427	0.914
Guarded	10	0.921	0.022	0.467	0.933	0.457	0.922
Guarded	25	0.931	0.019	0.531	0.961	0.499	0.931
Guarded	50	0.935	0.020	0.580	0.984	0.527	0.936
Guarded	500	0.945	0.017	0.637	0.993	0.557	0.943
Hard gate	5	0.916	0.024	0.438	0.885	0.436	0.916
Hard gate	10	0.923	0.022	0.480	0.935	0.465	0.924
Hard gate	25	0.929	0.020	0.528	0.969	0.492	0.930
Hard gate	50	0.935	0.020	0.579	0.980	0.528	0.936
Hard gate	500	0.944	0.017	0.633	0.992	0.559	0.943
Safety net	5	0.888	0.038	0.293	0.820	0.348	0.892
Safety net	10	0.904	0.021	0.303	0.825	0.358	0.901
Safety net	25	0.915	0.018	0.381	0.876	0.438	0.915
Safety net	50	0.916	0.019	0.394	0.905	0.441	0.917
Safety net	500	0.909	0.024	0.374	0.956	0.353	0.908

Table C.6: Original UNSW-NB15 flow-level multiclass results across training size, for the five couplings and the neural baseline. Bold marks the highest macro-F1 reached on this dataset. Every system stays in the same low band because the attack families overlap in the flow feature space.

0.46 at the largest budget.

At the packet level the pipeline did find rules, for the denial-of-service, exploits, fuzzers, and generic families, and they help a little. Figure C.4 reports the sweep: the neuro-symbolic model edges the neural baseline at the smaller budgets, 0.262 against 0.253 at $k = 50$ and 0.291 against 0.257 at $k = 100$, and the baseline closes the gap at the largest budget. The rules detect attacks reliably, with a detection rate near 1.0 and a false-positive rate below 0.01, but they cannot tell the families apart, so the macro-F1 stays low. This is the overlap reported on the original dataset, now visible at the rule level: a packet-header rule can say that a flow is an attack, but not which attack it is.

At the hybrid level the symbolic layer reuses the flow rules, which are empty, so the hybrid carries no symbolic evidence and reduces to a neural fusion of the packet and flow classifiers. It sits lowest of the three views, near 0.20 at $k = 50$ and 0.31 at the largest budget, because combining two weak views dilutes the signal rather than adding to it. Table C.7 collects the three views.

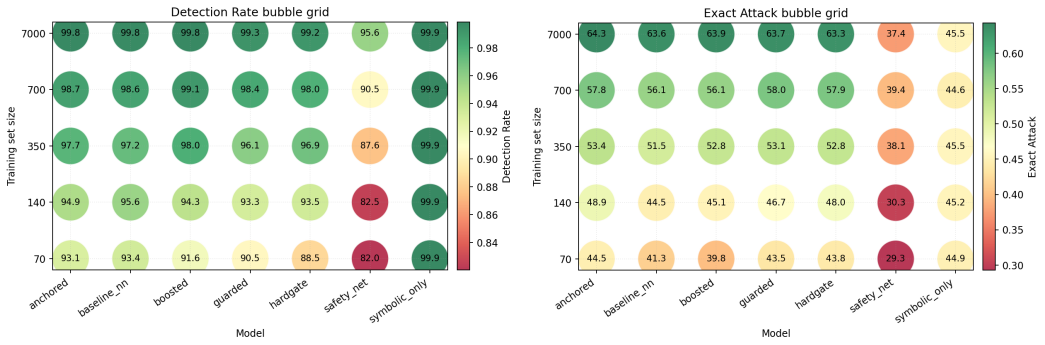


Figure C.2: UNSW-NB15 flow: detection-rate (left) and exact-attack (right) bubble grids over the five couplings and the baselines.

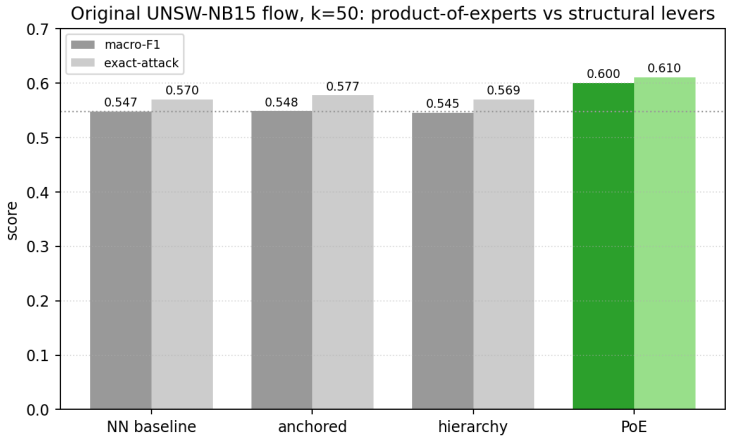


Figure C.3: Product-of-experts coupling against the neural baseline and the structural couplings on the original UNSW-NB15 flow data. The product-of-experts is the only variant that lifts macro-F1 above the neural baseline, by learning from the co-occurrence pattern of fired rules rather than gating on a single rule per class.

C.8 Hard versus soft thresholds

Every rung of the complexity ladder of Section 6.3.2 uses soft thresholds, and this section isolates what that representation costs against the crisp alternative. The two are compared on a single fixed program, the L3 rung of categorical anchors plus two numeric thresholds, changing only whether each feature threshold is a crisp deterministic atom or a learned soft tier. Table C.8 reports the result, and it runs against the naive expectation. The crisp program compiles smaller, one neural leaf against ten and 229 SDD edges against 484, because a deterministic threshold adds no trainable sub-network. Operationally, however, the soft program is far cheaper. A crisp threshold is a deterministic external whose truth depends on the raw feature value, so the grounded program differs from one flow to the next and

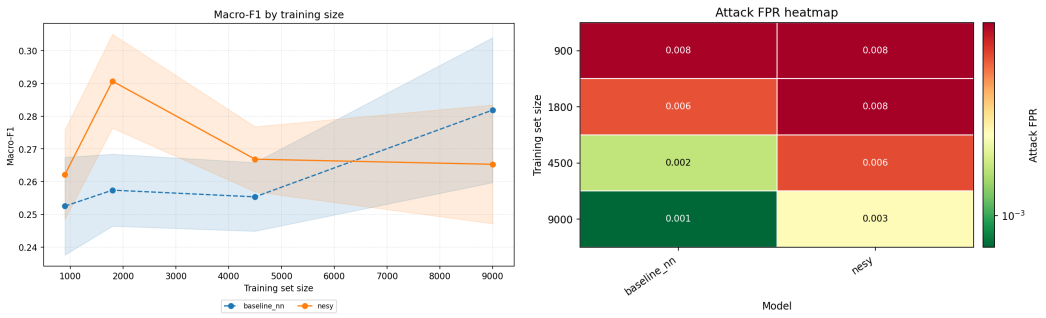


Figure C.4: Standardised UNSW-NB15 NetFlow-v3 packet view: macro-F1 by training set size (left) and attack false-positive heatmap (right), neural baseline against the pipeline neuro-symbolic model. The rules edge the baseline at the smaller budgets but the macro-F1 stays low, because the rules separate attack from benign without separating the families.

Table C.7: Standardised UNSW-NB15 NetFlow-v3 macro-F1 across the three traffic views, neural baseline against the pipeline neuro-symbolic model, at $k = 50$ and at the largest budget. Every view stays in the low band of the original UNSW-NB15 result.

view	system	macro-F1 ($k = 50$)	macro-F1 (largest)
flow	neural baseline	0.376	0.458
flow	NeSy (no rule survived)	0.376	0.458
packet	neural baseline	0.253	0.282
packet	NeSy	0.262	0.265
hybrid	neural fusion	0.201	0.309

no single arithmetic circuit can be reused: each prediction recompiles and training cannot cache, which yields 108 seconds per epoch and only 13 predictions per second. A soft threshold is instead a neural leaf over a fixed circuit, so the same compiled circuit is reused for every flow with the leaf values supplied by the tier network; it is compiled once, cached across training, and evaluated as a batched tensor at inference. The soft program therefore trains an order of magnitude faster and serves roughly three hundred times more flows per second, despite carrying ten neural leaves to the crisp program's one. It also trains stably to 0.701 macro-F1, whereas the crisp variant, whose anchors fire on raw thresholds that the benign-negation gate then over-suppresses, does not train stably and stays near 0.2. Put as a cost, the difference is where compilation is paid. A crisp threshold makes the grounded program depend on the raw feature value, so a separate circuit is compiled for every flow and the compilation is paid $\Theta(N)$ times across N predictions; a soft threshold compiles a single circuit whose leaves the tier network supplies per flow, so the compilation is paid once and amortised over the whole stream, leaving only a batched tensor evaluation at query time. The direction of the table, ten leaves and twice the edges yet three hundred times the throughput, is that amortisation: the soft program carries the larger circuit but builds it once, while the smaller crisp circuit is rebuilt on every prediction. The soft thresholds are thus not only the robustness lever of Section 6.2.2; they are what makes the symbolic layer cacheable

and vectorisable, and hence practical to train and serve.

Table C.8: Hard versus soft thresholds on one fixed program (the L3 rung: categorical anchors plus two numeric thresholds) on CIC-IDS2017 flow at $k = 500$. The only change is whether each feature threshold is a crisp deterministic atom or a learned soft tier. Leaves is the compiled circuit's neural-leaf count; throughput is steady-state predictions per second on the machine of Section 6.1.

	Hard (crisp)	Soft (tiered)
Neural leaves	1	10
SDD edges	229	484
Training (s/epoch)	107.9	9.8
Inference (pred/s)	12.9	4,808
Macro-F1	$\sim$0.2 (unstable)	0.701

C.9 Theoretical complexity analysis

The empirical fits of Section 6.3.2 measure the machine-specific constants of three asymptotic laws that can be derived directly from how DeepProbLog evaluates a program. A program grounds and compiles to an arithmetic circuit C in the d-DNNF/SDD class (decomposable and deterministic), on which weighted model counting (WMC) is tractable. Write E for the number of edges of C (its size), L for the number of neural leaves (atoms whose weight is a network output), P for the number of soft-tier predicates (each backed by a small trainable sub-network), N for the number of training samples, and tw for the treewidth of the program's primal graph. Let c_{nn} be the cost of one shared-network forward pass and c_{tier} the cost of one tier sub-network forward-and-backward.

Proposition 1 (Inference). Per-sample inference costs $\Theta(E + L\,c_{\text{nn}})$, dominated by $L\,c_{\text{nn}}$.

Proof. Inference is the WMC of the query node. First, the L neural leaves each require one network forward pass, contributing $L\,c_{\text{nn}}$; the remaining leaves are $O(1)$ weight lookups. Second, C is a DAG of sum and product nodes evaluated once in reverse topological order, each node computed from its in-neighbours in time proportional to its in-degree; summing over nodes gives $\sum_v \text{indeg}(v) = |E|$, so circuit evaluation is $\Theta(E)$ — this linearity is the defining property of d-DNNF/SDD that makes WMC a single bottom-up pass. The total is $\Theta(E + L\,c_{\text{nn}})$. A network pass dwarfs the $O(1)$ per-edge arithmetic, so the $L\,c_{\text{nn}}$ term dominates and latency follows $a + c_{\text{nn}}\,L$. □

Proposition 2 (Training). One epoch costs $\Theta\big(N\,(E + L\,c_{\text{nn}} + P\,c_{\text{tier}})\big)$, with growth governed by P.

Proof. An epoch is a forward and a backward pass over N samples. The forward pass is inference, $\Theta(E + L\,c_{\text{nn}})$ by Proposition 1. The backward pass computes all parameter gradients by reverse-mode automatic differentiation on the same circuit and networks; by the cheap-gradient theorem (Baur–Strassen), reverse-mode differentiation costs $\Theta(\text{forward})$, so

the backward pass is also $\Theta(E + L c_{\text{nn}})$. Each of the P soft-tier predicates carries an independent trainable sub-network whose forward and backward cost c_{tier}, adding $\Theta(P c_{\text{tier}})$. Per sample this is $\Theta(E + L c_{\text{nn}} + P c_{\text{tier}})$; multiplied by N over the epoch. As a program grows, E and L stay small and co-vary while P grows, so the dominant term is $N P c_{\text{tier}}$ and the epoch time follows $a + (N c_{\text{tier}}) P$. □

Proposition 3 (Compilation). The compiled circuit has size $|C| = O(2^{\text{tw}} n)$, which is tight: some programs force $|C| = 2^{\Omega(\text{tw})}$.

Proof (upper bound). Take a tree decomposition of the primal graph of width tw with n bags. Compile bottom-up over the tree: each bag has at most $\text{tw} + 1$ variables, hence at most $2^{\text{tw}+1}$ consistent partial assignments, and joining a bag with its child marginalises the shared separator variables. Each bag emits $O(2^{\text{tw}+1})$ circuit nodes and each join is $O(2^{\text{tw}+1})$ work, over n bags, giving $|C| = O(n 2^{\text{tw}+1}) = O(2^{\text{tw}} n)$; bounded treewidth therefore yields a polynomial circuit. The matching lower bound — that there exist function families whose every d-DNNF/SDD has size $2^{\Omega(\text{tw})}$ — follows from known knowledge-compilation lower bounds and is cited rather than reproduced here [11]. □

Propositions 1 and 2 explain the empirical readings of Section 6.3.2: inference scales with the neural-leaf count L (fitted slope $c_{\text{nn}} \approx 0.0094$ ms) and an epoch scales with the soft-tier-predicate count P (fitted slope $N c_{\text{tier}} \approx 0.59$ s), while compilation is the one axis with no benign law, its non-monotone memory peak being the treewidth sensitivity of Proposition 3 surfacing on a small program. Because three different structural quantities set the three costs, a coupling can be cheap to serve and slow to train, or heavy in clauses and light in memory, instead of paying on every axis at once.