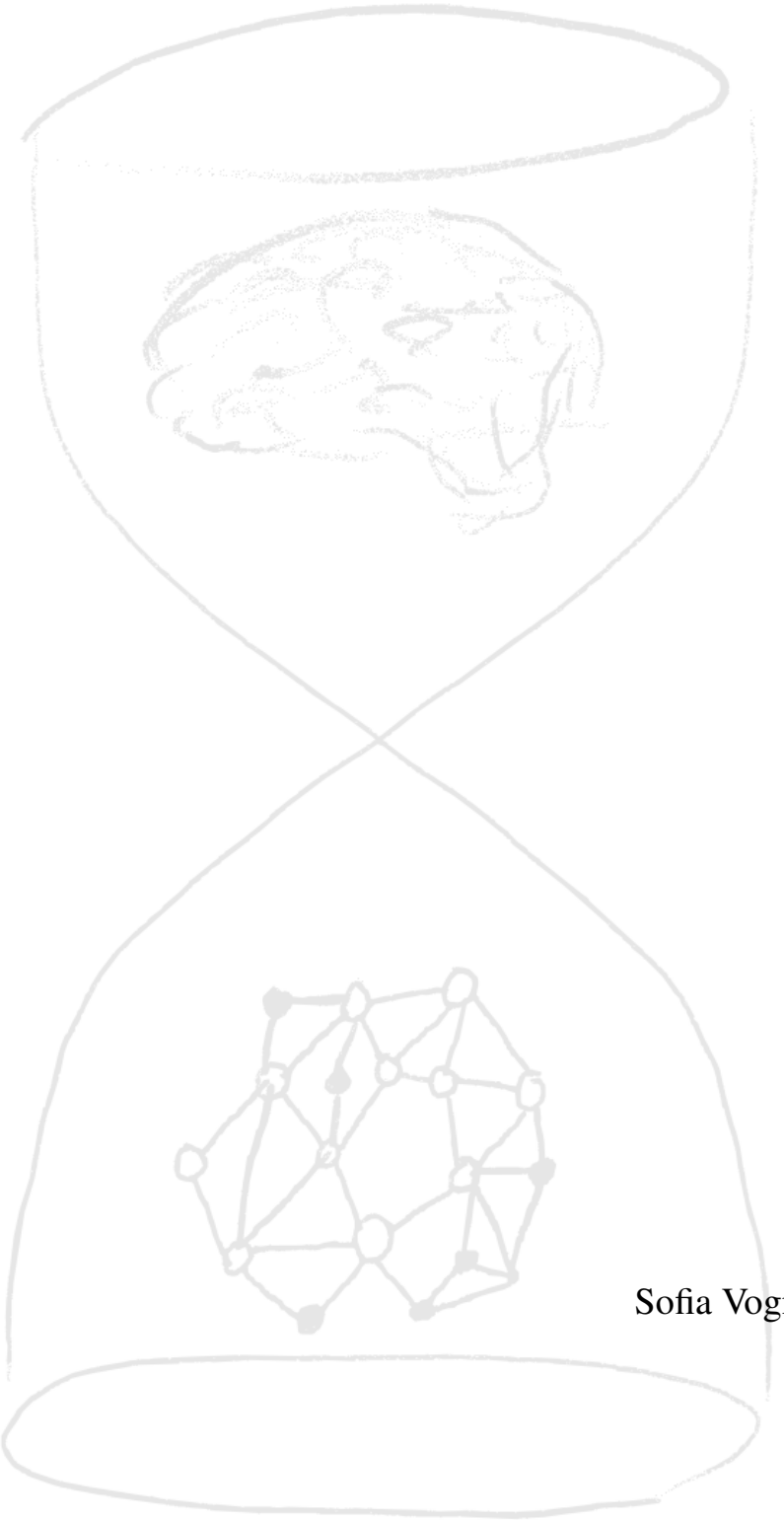
Temporal Neuro-Symbolic Reasoning for Multi-Step Attack Detection

Master's Thesis



Sofia Vognild Burkow

Temporal Neuro-Symbolic Reasoning for Multi-Step Attack Detection

THESIS

submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER SCIENCE

by

Sofia Vognild Burkow
born in Tromsø, Norway



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

©2026 Sofia Vognild Burkow. All rights reserved.

Temporal Neuro-Symbolic Reasoning for Multi-Step Attack Detection

Author: Sofia Vognild Burkow
Student id: 6296416

Abstract

Multi-step attacks pose a significant challenge for intrusion detection systems due to their temporal structure and the dependencies between individual attack steps. Detecting such attacks during their early stages is particularly important, as it enables defenders to intervene before an attack reaches its final objective.

This thesis presents a neuro-symbolic framework for real-time multi-step attack detection based on DeepProbLog. The proposed approach combines neural attack-step classification with symbolic temporal reasoning, integrating knowledge of attack progression directly into the learning and inference process. By enforcing temporal constraints, the system aims to improve both attack-step detection and the consistency of predicted attack progressions.

The proposed approach is evaluated on two multi-step attack datasets: DARPA 2000 and AIT-LDS V2. Experimental results show that incorporating symbolic knowledge consistently improves detection performance compared to neural and symbolic baselines. In particular, the neuro-symbolic models achieve higher macro F1-scores, lower false alarm rates, and more temporally consistent predictions. The results further indicate that symbolic constraints improve data efficiency and generalization while maintaining low inference latency.

Overall, this thesis demonstrates that integrating symbolic knowledge of attack progression into neural intrusion detection models provides an effective framework for multi-step attack detection, supporting accurate and timely identification of attacks across multiple stages.

Thesis Committee:

Thesis advisor:	Prof. Dr. G. Smaragdakis, Faculty EEMCS, TU Delft
Daily supervisor:	Dr. K. Liang, Faculty EEMCS, TU Delft
Daily co-supervisors:	A. Agiollo, Faculty EEMCS, TU Delft
	E. Bardhi, Faculty EEMCS, TU Delft
Committee Member:	Dr. S. Dumančić, Faculty EEMCS, TU Delft

Preface

Before diving into the details of what I have been working on over the past year, I would like to express my sincere thanks to the people who supported me along the way.

First and foremost, I would like to thank my daily co-supervisors Andrea Agiollo and Enkeleda Bardhi. You made the thesis process a genuinely enjoyable experience, and for a long time, a surprisingly stress-free one. Thank you for always helping me whenever I got stuck (and, to be honest, you usually had the answers). Thank you for pushing me to aim for better results, and most importantly, thank you for treating me and the other thesis students as more than just students. I think I can safely say on behalf of all of us that you are excellent supervisors and, just as importantly, amazing people.

I would also like to thank my main thesis advisor, George Smaragdakis. Despite a packed schedule as a professor and an impressive number of thesis students, you still took on the responsibility of supervising my thesis. I truly appreciate that, as well as the valuable feedback you provided throughout the project.

Many thanks as well to Kaitai Liang for continuing to supervise me even after moving to Finland, and for the helpful feedback during our meetings.

I am also grateful to Sebastijan Dumančić for agreeing to be part of my thesis committee, and for creating the neuro-symbolic language DeepProbLog that I base my thesis on. I did my best to make full use of its capabilities, although learning it came with a steep learning curve, and I still have much to explore.

Naturally, I would also like to thank my parents and friends for their constant support. You helped me calm down during stressful periods, take breaks and put a smile on my face when I needed it the most, keep track of thesis details I definitely would not have managed on my own, and fill my water bottle when I was too focused on writing to move away from my desk.

And finally, a special thanks to my trusty computer, which ran experiments day and night for months on end, usually accompanied by a YouTube fireplace in the background.

It has been a rewarding but challenging journey. I am proud of the result and deeply grateful to everyone who helped me along the way. I hope this thesis is a pleasant read and perhaps sparks some curiosity along the way.

Sofia Vognild Burkow
Delft, the Netherlands
June 19, 2026

Contents

Preface iii

Contents v

List of Figures vii

1 Introduction 1

2 Background 5
2.1 Network Intrusion Detection Systems . . . 5
2.2 Multi-Step Attacks . . . 6
2.3 Neuro-Symbolic AI . . . 6
2.4 DeepProbLog . . . 6

3 Related Work 9
3.1 Correlation-Based Approaches . . . 9
3.2 Data-Driven Approaches . . . 10

4 Threat Model 13
4.1 Target Environment . . . 13
4.2 Cost of Missed Attacks and False Alarms . . . 13
4.3 Attacker Model . . . 13
4.4 Evasion Assumptions . . . 14

5 Neuro-Symbolic MSA Detection 15
5.1 System Overview . . . 15
5.2 Data Preparation . . . 15
5.3 Neuro-Symbolic Attack-Phase Inference . . . 18
5.4 Decision and Memory . . . 20
5.5 Multi-Step Attack Knowledge . . . 21

6 Implementation 23
6.1 Datasets . . . 23
6.2 Model Design . . . 25
6.3 Data Preparation . . . 27

7 Experimental Evaluation 31
7.1 Evaluation Metrics . . . 31
7.2 Baselines . . . 32

7.3 Experiments 32

8 Discussion **45**
8.1 The Role of Symbolic Knowledge 45
8.2 End-to-End Learning vs. Pretraining 46
8.3 Data Efficiency 46
8.4 Temporal Context 47
8.5 Computational Performance 47
8.6 Limitations 47
8.7 Future Work 49

9 Conclusions **51**

Bibliography **53**

A Glossary **57**

B Extended Experimental Results **59**
B.1 Ablation Study 59

List of Figures

2.1 Classification of network intrusion detection systems based on the detection style. 5

5.1 Overview of the proposed neuro-symbolic multi-step attack detection system. 16

7.1 Confusion matrices for Kitsune on the AIT-LDS V2 and DARPA 2000 datasets. The matrices illustrate the distribution of correct predictions and misclassifications across the evaluated classes, highlighting Kitsune's performance per attack-phase. 36
7.2 Temporal consistency comparison across neural and neuro-symbolic models on AIT-LDS V2, highlighting differences between baseline LSTMs and DPL models. 36
7.3 Temporal consistency comparison across neural and neuro-symbolic models on DARPA 2000, highlighting differences between baseline LSTMs and DPL models. 37
7.4 Effect of training set size on model performance across AIT-LDS V2 (left) and DARPA 2000 (right). Results show how macro F1, FAR, and DR vary as the number of training samples increases for different model configurations. 38
7.5 Confusion matrices illustrating cross-dataset generalization on AIT-LDS V2. Each matrix compares model performance when trained and tested on the same scenario with performance when evaluated on a different dataset. 40
7.6 Confusion matrices illustrating cross-dataset generalization on DARPA 2000. Each matrix compares model performance when trained and tested on the same scenario with performance when evaluated on a different dataset. 41
7.7 Confusion matrices for the end-to-end DPL model on AIT-LDS V2. Subfigure (a) shows the effect of removing attack phases when using the original logic. Subfigure (b) shows performance on the corrupted dataset after introducing the revised logic. The comparison illustrates the robustness of the proposed logic to missing attack phases. 42

Chapter 1

Introduction

The move to a digital world has made us susceptible to new threats in the form of cyberattacks. The goal of cyberattacks is to steal data, or cause harm to a system [29]. At the same time as defenders develop new security mechanisms, attackers continuously evolve their methods, creating increasingly sophisticated techniques. Rather than occurring as isolated events, many attacks unfold over multiple stages, enabling adversaries to gradually gain access to valuable assets while evading detection [25]. Such attacks are commonly referred to as multi-step attacks (MSAs).

In a multi-step attack, individual attack steps follow a temporal progression in which one action prepares for the next [25]. This strategy is frequently employed by Advanced Persistent Threats (APTs), which seek to establish and maintain long-term access to a target environment [34]. The consequences of successful APT campaigns can be severe, ranging from data theft [41] and service disruption [6] to the compromise of critical infrastructure [40].

Detecting an attack during its intermediate stages provides defenders with an opportunity to intervene before the adversary reaches its final objective. The importance of such early detection was illustrated by the cyberattack against Eindhoven University of Technology (TU/e) in January 2025 [9]. Although the attacker had gained broad access to the university network, suspicious activity was detected before the compromise had escalating into a successful attack. This allowed the university to take its network offline, preventing a potentially far more damaging outcome such as a ransomware deployment [9]. Consequently, detecting attacks as early as possible is of particular importance.

Timely detection of multi-step attacks requires systems that can operate in real time, detect individual attack steps, and reason about how those steps fit within a plausible multi-step attack progression. In modern cybersecurity, individual attack steps are typically detected by Network Intrusion Detection Systems (NIDSs). Most deployed NIDSs are rule-based systems, such as Snort [35] and Suricata [39], which identify malicious activity by matching network traffic against signatures of known attacks [38]. These systems operate in real time and are therefore well suited for timely detection. However, they are primarily designed to detect individual attack actions and generally do not consider the temporal relationships between alerts that characterize multi-step attacks.

As a result, a substantial body of research has focused on alert correlation [25, 17, 34]. Alert correlation systems link individual intrusion alerts together to reconstruct attack scenarios and reason about attack progression. By modeling relationships between alerts, these approaches provide analysts with a higher-level view of ongoing attacks and can help distinguish meaningful attack sequences from isolated alerts. While many correlation techniques are applied post hoc for forensic analysis [25], several studies have investigated real-time correlation to support earlier detection and response [4, 1].

Despite these advances, correlation-based approaches remain fundamentally dependent on the alerts generated by the underlying intrusion detection system (IDS). If critical attack steps are not detected, the resulting attack reconstruction becomes incomplete [27, 26]. Furthermore, many intrusion detection signatures are designed to capture activities that occur in the later stages of an attack, while early-stage behaviors such as reconnaissance and scanning may appear benign when viewed in isolation [30, 4]. Consequently, timely detection before the attacker reaches the final objective remains challenging.

An alternative is provided by data-driven intrusion detection systems. Rather than relying on predefined signatures, these systems learn patterns directly from network traffic and can therefore detect previously unseen behavior. Data-driven approaches have demonstrated the ability to identify early-stage attack activities while maintaining real-time performance [22]. In addition, modern learning-based methods can capture temporal dependencies in data [30, 31], allowing them to model aspects of multi-step attack progression implicitly.

However, purely data-driven approaches also face limitations. Many methods focus on determining whether a sequence of events constitutes a multi-step attack rather than identifying specific attack stages [31], which limits their usefulness for early intervention. In addition, these models are often described as black-box systems with limited explainability [21]. They also typically require large amounts of training data to learn reliable attack patterns [38], while publicly available datasets with realistic multi-step attack scenarios remain scarce [3].

Neuro-symbolic artificial intelligence (NeSy AI) offers a potential solution. By combining neural learning with symbolic reasoning, neuro-symbolic methods enable models to learn from data while also leveraging structured domain knowledge. Multi-step attack detection fits this setting well, as it requires low-level perception to identify malicious patterns in network traffic while simultaneously reasoning about their temporal progression. Such approaches have demonstrated benefits across a range of structured learning tasks, addressing several limitations of purely data-driven models, including improved robustness [2, 21, 12], better interpretability [3, 43], and reduced dependence on large labeled datasets [21, 12].

In this thesis, we investigate the effectiveness and feasibility of neuro-symbolic artificial intelligence for multi-step attack detection. Specifically, we explore how the neural probabilistic logic programming framework DeepProbLog (DPL) [21] can be used to incorporate explicit knowledge of attack progression into the learning and inference process of a neural intrusion detection model.

The proposed system combines neural flow classification with symbolic temporal reasoning to enforce constraints on valid attack progressions. In particular, previous observations are used in real time to guide future predictions, ensuring that detected attack steps are temporally consistent. This enables the system to operate on incoming network flow data in real time, supporting timely detection of individual attack steps throughout the attack lifecycle and allowing for early intervention before an attacker reaches their final objective.

To the best of our knowledge, this thesis represents the first investigation of neuro-symbolic artificial intelligence for multi-step attack detection. Rather than treating attack progression as something to be inferred post hoc or learned implicitly from data, the proposed approach integrates this knowledge directly into the detection process, allowing it to actively constrain and guide predictions.

The goal is not to replace rule-based systems such as Snort [35] and Suricata [39], but to complement them with a system capable of capturing a broader range of attack behaviors and enabling real-time multi-step attack detection.

The proposed approach is evaluated on two public multi-step attack datasets: DARPA 2000 [18] and AIT-LDS V2 [14, 15]. Through a series of experiments, we investigate the ability of the system to (i) detect individual attack steps, (ii) enforce temporally valid attack progressions, (iii) learn effectively from limited training data, (iv) generalize to previously unseen attack scenarios, (v) remain robust in the presence of missing attack phases, and (vi) achieve computational performance suitable for real-time deployment. We compare our approach against neural, logical, and state-of-the-art anomaly-based baselines [22].

To guide this investigation, we formulate the following research question and sub-questions:

RQ1. How does a neuro-symbolic approach compare to purely neural methods for multi-step attack detection?

RQ1.1. How does the neuro-symbolic approach compare to purely neural methods in terms of F1-score, false alarm rate (FAR), and detection rate (DR)?

RQ1.2. To what extent does the neuro-symbolic approach enforce the temporal constraints inherent in multi-step attacks?

RQ1.3. To what degree can neuro-symbolic integration achieve competitive detection performance while requiring fewer training samples than purely data-driven models?

RQ1.4. How does symbolic knowledge affect generalization to unseen attack scenarios and environments?

RQ1.5. To what extent is DeepProbLog suitable for real-time multi-step attack detection in terms of latency and computational efficiency?

The remainder of this thesis is organized as follows. Chapter 2 provides relecant background on network intrusion detection systems, multi-step attacks, neuro-symbolic AI, and DeepProbLog. Chapter 3 reviews related work in multi-step attack detection. Chapter 4 introduces the threat model considered in this work. Chapter 5 presents the proposed neuro-symbolic multi-step attack detection system, while Chapter 6 describes its implementation. Chapter 7 presents the experimental setup and results, which are discussed in Chapter 8. Finally, Chapter 9 concludes the thesis.

Chapter 2

Background

This chapter presents the relevant background and foundational concepts required to understand the remainder of this thesis. Specifically, we provide relevant background knowledge on network intrusion detection systems, multi-step attacks, neuro-symbolic AI and the neuro-symbolic framework DeepProbLog [20, 21].

2.1 Network Intrusion Detection Systems

Network intrusion detection systems (NIDSs) play a critical role in modern cybersecurity by monitoring network traffic and identifying malicious activities. Broadly, NIDSs can be categorized into signature-based and anomaly-based systems according to their detection strategy [38] (see Figure 2.1). Signature-based systems detect attacks by matching traffic against known patterns or predefined rules, whereas anomaly-based systems rely on data-driven models to identify deviations from learned normal behavior.

Despite their widespread deployment, both paradigms suffer from fundamental limitations. Signature-based approaches are effective at detecting known threats but struggle to generalize to previously unseen attacks. In contrast, anomaly-based systems offer the potential to detect novel threats but depend heavily on large amounts of labeled training data and often lack interpretability. The complex and highly variable nature of network traffic further adds to these challenges, resulting in high false positive rates, which in practice lead to alert fatigue and reduced operational usability.

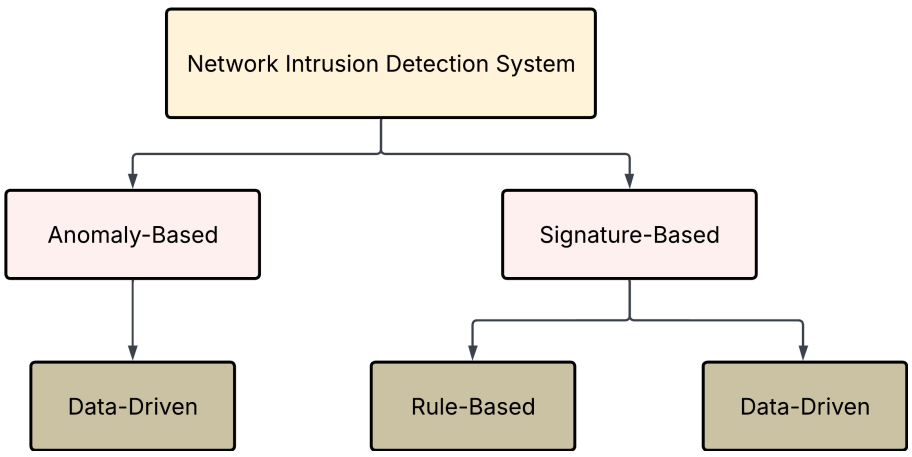


Figure 2.1: Classification of network intrusion detection systems based on the detection style.

2.2 Multi-Step Attacks

Traditional intrusion detection systems typically focus on identifying individual malicious events. However, many real-world intrusions consist of multiple coordinated actions rather than isolated events. These are commonly referred to as multi-step attacks [34].

Several frameworks, such as the Cyber Kill Chain [19] and MITRE ATT&CK [23], describe how multi-step attacks evolve over time and which tactics are associated with each stage. The Cyber Kill Chain consists of seven phases: reconnaissance, weaponization, delivery, exploitation, installation, command & control, and actions on objectives. An attack may begin with reconnaissance to gather information about the target environment, followed by exploitation of a vulnerable service, privilege escalation, lateral movement, and ultimately the achievement of a final objective.

Detecting multi-step attacks is challenging for several reasons [25]. First, attack steps may be separated by long time intervals, making it difficult to correlate related events. Second, individual steps may resemble legitimate user behavior, leading to ambiguity when considered in isolation . Finally, attackers may vary the exact implementation of an attack while preserving the same overall objective, resulting in substantial variability between attack instances.

2.3 Neuro-Symbolic AI

Neuro-symbolic artificial intelligence combines neural learning with symbolic reasoning. Neural networks excel at learning patterns from data but often lack interpretability and struggle to incorporate prior knowledge. Symbolic systems, in contrast, provide interpretable reasoning and allow expert knowledge to be encoded explicitly.

By combining these paradigms, neuro-symbolic systems aim to leverage the strengths of both approaches. Neural components are responsible for perception and representation learning, while symbolic components provide structured reasoning and the ability to incorporate domain knowledge.

2.4 DeepProbLog

DeepProbLog [20, 21] is a neuro-symbolic probabilistic logic programming framework that extends ProbLog [7] with neural predicates. Rather than integrating symbolic reasoning into a neural architecture, DeepProbLog embeds neural networks into a probabilistic logic program while retaining the full expressiveness of probabilistic logical reasoning. This allows neural predictions to be used as probabilistic facts during logical inference, enabling perception and reasoning to be combined within a single framework.

2.4.1 Program Structure

A DeepProbLog program consists of three main components: logical rules, probabilistic facts, and neural predicates [20]. Logical rules define how conclusions can be derived from known facts, while probabilistic facts associate uncertainty with logical statements, enabling probabilistic reasoning. Neural predicates connect perception and reasoning by replacing manually specified probabilities with values predicted by neural networks.

The interaction between these components can be illustrated through a simple multi-step attack detection example. A rule may specify that a host is compromised if a successful exploitation event is followed by privilege escalation. In this setting, the probability of observing an exploitation event can be provided by a neural network, while the logical rule captures the dependency between attack stages.

2.4.2 Neural Predicates

Neural predicates provide the interface between perception and reasoning in DeepProbLog. Instead of assigning fixed probabilities to facts, a neural predicate derives its probability from the output of a neural

network. For example, a network may take network traffic features as input and estimate the probability that an observation corresponds to a specific attack stage. These probabilities can then be used directly within logical rules, in the same way as conventional probabilistic facts.

2.4.3 Inference

Inference in DeepProbLog consists of computing the probability that a query is true given a set of logical rules, probabilistic facts, and neural predictions. To answer a query, the relevant portion of the logic program is grounded, transformed into a probabilistic graphical representation, and evaluated using probabilistic inference techniques.

2.4.4 End-to-End Learning

A distinguishing feature of DeepProbLog is that learning is guided by the final logical objective rather than solely by local classification errors. During training, gradients are propagated through the probabilistic inference process and into the neural components. Consequently, symbolic knowledge can influence the representations learned by the neural network. Hence, reasoning and perception are optimized jointly.

Chapter 3

Related Work

Since the early 2000s, significant research has focused on the detection of multi-step attacks, also referred to as attack scenarios [25]. Existing approaches have largely concentrated either on correlating alerts raised by intrusion detection systems (IDSs), or detecting multi-step attacks directly from data.

3.1 Correlation-Based Approaches

Unlike traditional intrusion detection systems, which analyze events independently, correlation-based approaches aim to reconstruct attack progression by modeling relationships between individual attack steps [25, 4].

3.1.1 Linking Strategies

Existing correlation methods are categorized according to their linking approach: similarity-based, causal-based, case-based, and structural-based approaches [25, 17].

Similarity-based approaches correlate alerts by identifying shared characteristics between events, such as identical IP addresses, ports, or timestamps [25, 17]. For example, Chen et al. link events when a suspicious "network subject" shares the same IP address or port across consecutive sessions [4]. While these methods can identify previously unseen attack variants, they often suffer from high false positive rates when similarity criteria are too permissive [25, 17].

Causal-based approaches use a "storytelling" logic in which alerts are linked if the consequences of one step satisfy the preconditions of a subsequent step [27, 26]. These approaches are highly interpretable and typically produce fewer false positives than similarity-based methods. However, they rely heavily on manually defined attack knowledge and struggle to generalize to previously unseen attack strategies [25].

Case-based approaches rely on a library of predefined attack scenarios, where each case describes an ordered sequence of attack steps. Observed alert sequences are matched against these stored cases to identify ongoing attacks [25]. Because detection requires a close match to a known scenario, this approach typically achieves low false positive rates. However, its main limitation is its dependence on predefined attack templates, which makes it difficult to detect novel or significantly modified attack strategies.

Structural approaches incorporate additional contextual information, such as network topology and vulnerability data, to filter irrelevant or implausible alerts [25]. For instance, NOC-IDS combines ontologies and non-monotonic reasoning to determine whether a target service is actually vulnerable to a detected exploit, thereby reducing the number of irrelevant alerts presented to security operators [5].

While early correlation systems were primarily designed for post-hoc forensic analysis, more recent research has explored real-time correlation. Chen et al. introduced Active Event Correlation (AEC), which quarantines suspicious subjects once a security policy violation is observed [4]. Similarly, An-

gelini et al. use attack graphs to estimate whether an ongoing sequence of alerts is progressing toward a known attack path, allowing intervention before the final objective is reached [1].

3.1.2 Limitations

Despite significant advances in alert correlation, these approaches remain fundamentally constrained by the granularity of the underlying intrusion detection system (IDS) alerts.

Because correlation systems operate on alerts generated by an IDS, they can only reason over attack steps that are actually detected. As a result, early-stage activities such as reconnaissance are often missed, since they may appear benign in isolation and are therefore difficult to capture using signature-based rules alone [30, 25]. This can limit the completeness of the resulting attack reconstruction and, in some cases, prevent meaningful correlation altogether.

Furthermore, if critical attack steps are not detected, the resulting attack graph becomes incomplete or fragmented, which hinders accurate reconstruction of the full attack scenario [27, 26]. To address this issue, Ning and Xu propose methods for hypothesizing missing attack steps and validating them using additional audit data [26].

In addition, many correlation systems rely on deep packet inspection, which makes them increasingly difficult to deploy in environments dominated by encrypted traffic [31].

3.2 Data-Driven Approaches

Data-driven intrusion detection systems operate directly on network traffic or network flows rather than on pre-generated IDS alerts. This allows them to learn complex behavioral patterns from data and potentially detect attack variants that are difficult to capture through manually defined signatures [11, 30].

3.2.1 Intrusion Detection

One of the most influential examples is Kitsune [22], which uses an ensemble of autoencoders to perform online anomaly detection. Kitsune is lightweight and capable of operating in real time, but it focuses on identifying anomalous traffic patterns rather than modeling attack progression. As a result, its performance depends heavily on the chosen feature representation and it may fail to detect attacks that do not manifest themselves through the monitored features [11].

To improve generalization, Guo et al. proposed KNOWML [11], which incorporates symbolic attack knowledge into machine learning models through knowledge graphs automatically generated from attack scripts. While this approach improves robustness against attack variations, symbolic knowledge is primarily used to improve feature representations rather than to reason about attack progression.

3.2.2 Multi-Step Attack Detection

Recent research has increasingly incorporated temporal modeling into learning-based systems in order to capture attack progression across the kill chain.

Ghafir et al. proposed MLAPT, which combines multiple specialized detection modules with a machine learning predictor to estimate whether observed events are likely to evolve into a complete Advanced Persistent Threat (APT) campaign [10]. However, the underlying detection modules remain heavily dependent on manually maintained blacklists and signatures.

More recent approaches employ deep learning to model temporal dependencies directly. StageFinder [30] combines Graph Neural Networks (GNNs) and Long Short-Term Memory (LSTM) networks to estimate an attacker's position within the MITRE ATT&CK kill chain. Similarly, Tracegram [31] models attack traces as collections of flows and uses temporal aggregation mechanisms to identify attack-relevant traffic patterns.

These approaches demonstrate the potential of learning-based systems to capture attack progression. However, they often rely on assumptions regarding trace construction and temporal segmentation. Furthermore, while they learn temporal relationships from data, they generally do not incorporate explicit expert knowledge about valid attack progressions.

In summary, correlation-based approaches provide explicit reasoning about attack progression but depend on alerts generated by an underlying IDS. Learning-based approaches operate directly on network traffic and support real-time detection, but generally model attack progression implicitly. These observations motivate the investigation of approaches that combine attack detection and attack progression reasoning within a single framework.

Chapter 4

Threat Model

In this chapter we specify the threat model under which our system operates. We aim to clarify the assumptions, limitations, and intended deployment setting of the system. Hopefully aiding security professionals in understanding how the system should be used in practice and how its outputs should be interpreted.

4.1 Target Environment

The neuro-symbolic multi-step attack detection system is designed for real-time multi-step attack detection in medium to large-scale network environments. This imposes several practical requirements, including low-latency processing, scalability to high traffic volumes, and robustness under continuous operation. In such environments, a high false alarm rate is particularly problematic, as it can overwhelm security operators and reduce trust in the system.

The system assumes access to network traffic captures, which are processed by network monitoring tools capable of flow-based feature extraction. These tools transform raw traffic into structured flow-level features suitable for downstream analysis. Due to the widespread adoption of encrypted communication protocols (e.g., HTTPS), the system relies exclusively on features derived from packet headers. Payload-based inspection is therefore considered out of scope.

4.2 Cost of Missed Attacks and False Alarms

The primary objective of the system is to detect multi-step attacks as early as possible, ideally before the final attack objective is reached. This introduces different requirements across attack phases.

Early-stage activities, such as reconnaissance, are often weak indicators of malicious intent when considered in isolation. Missing such events is therefore less critical. In contrast, later stages such as exploitation represent strong indicators of compromise and must be detected with high reliability.

Therefore, recall should be prioritized over precision for early attack phases. This is motivated by the sequential nature of multi-step attacks: detection of later phases depends on the successful identification of earlier ones. Missed attacks in early stages can therefore prevent the system from recognizing the full attack sequence.

At the same time, false alarms remain a major concern, particularly for machine learning-based approaches. The system addresses this by only raising an alarm once a sufficient number of attack steps have been observed. This allows the system to filter out false predictions.

4.3 Attacker Model

We consider adversaries with varying levels of sophistication, ranging from low-skilled attackers using automated tools to skilled attackers adopting sophisticated techniques.

We assume that attackers follow multi-step attack patterns consistent with established frameworks such as the Cyber Kill Chain [19]. While the exact implementation of individual attacks may vary, the overall staged structure of reconnaissance, exploitation, and post-exploitation activities is assumed to remain largely consistent.

In addition, we assume that relevant attack steps are observable in network traffic at the flow level, allowing them to be captured by the proposed detection system.

Finally, we assume that multi-step attacks may unfold over extended periods (ranging from hours to days), requiring the system to maintain contextual information over longer time windows in order to correctly associate related events.

4.4 Evasion Assumptions

The system does not explicitly model highly adaptive adversaries attempting to evade detection. In particular, we assume that attackers do not deliberately manipulate their behavior to bypass detection mechanisms, for example by introducing large temporal gaps between attack steps or frequently changing identifiers to break correlations.

While such evasion strategies are possible, prior work [38] suggests that many real-world attacks are opportunistic and do not involve sophisticated evasion techniques. The focus of this work is therefore on detecting realistic multi-step attack scenarios rather than adversarially optimized ones.

Chapter 5

Neuro-Symbolic MSA Detection

Detecting multi-step attacks requires not only identifying individual attack steps, but also reasoning about how they evolve over time. This work introduces a neuro-symbolic MSA detection system built using DeepProbLog [20, 21], which combines neural classification with probabilistic symbolic reasoning to detect multi-step attacks. Unlike traditional multi-step attack detection systems, our approach integrates prior knowledge of attack progression directly into the inference process, allowing previously observed phases to guide subsequent predictions.

In this chapter, we first provide a brief overview of the proposed system. Next, we describe the three main components of the system: (i) data preparation, (ii) neuro-symbolic attack-phase inference, and (iii) memory and multi-step attack decision. Finally, we discuss how to derive the required knowledge for MSA detection.

5.1 System Overview

Figure 5.1 provides an overview of the proposed multi-step attack detection system. During data preparation, incoming network traffic captures are transformed into network flows, statistical features are computed, and DeepProbLog queries are constructed for neuro-symbolic attack-phase inference. During inference, both the neural and symbolic components reason about the most likely attack phase, after which a joint decision is made. If an attack phase is detected, it is stored in a lightweight database that maintains the attack history. This attack history is then used to provide contextual information for subsequent predictions. Finally, a multi-step attack alarm is raised if the accumulated attack history satisfies predefined MSA criteria.

5.2 Data Preparation

The data preparation component is responsible for transforming incoming network traffic captures into a format that can be used for neuro-symbolic inference.

5.2.1 Network Flows

The first stage of the pipeline transforms raw network traffic captures into network flows. A network flow is defined as a grouping of packets sharing the same 5-tuple (source IP, destination IP, source port, destination port, and protocol) within a given time interval [13].

This abstraction provides a condensed view of network activity, capturing communication patterns, volume statistics, and relationships between endpoints. In addition, flows are inherently sequential, allowing observations to be analyzed in the context of preceding and subsequent activity. This is particularly useful for detecting attack behaviors that only become apparent when considered across multiple events, making flow-based representations well suited for machine learning approaches that rely on temporal or relational structure.

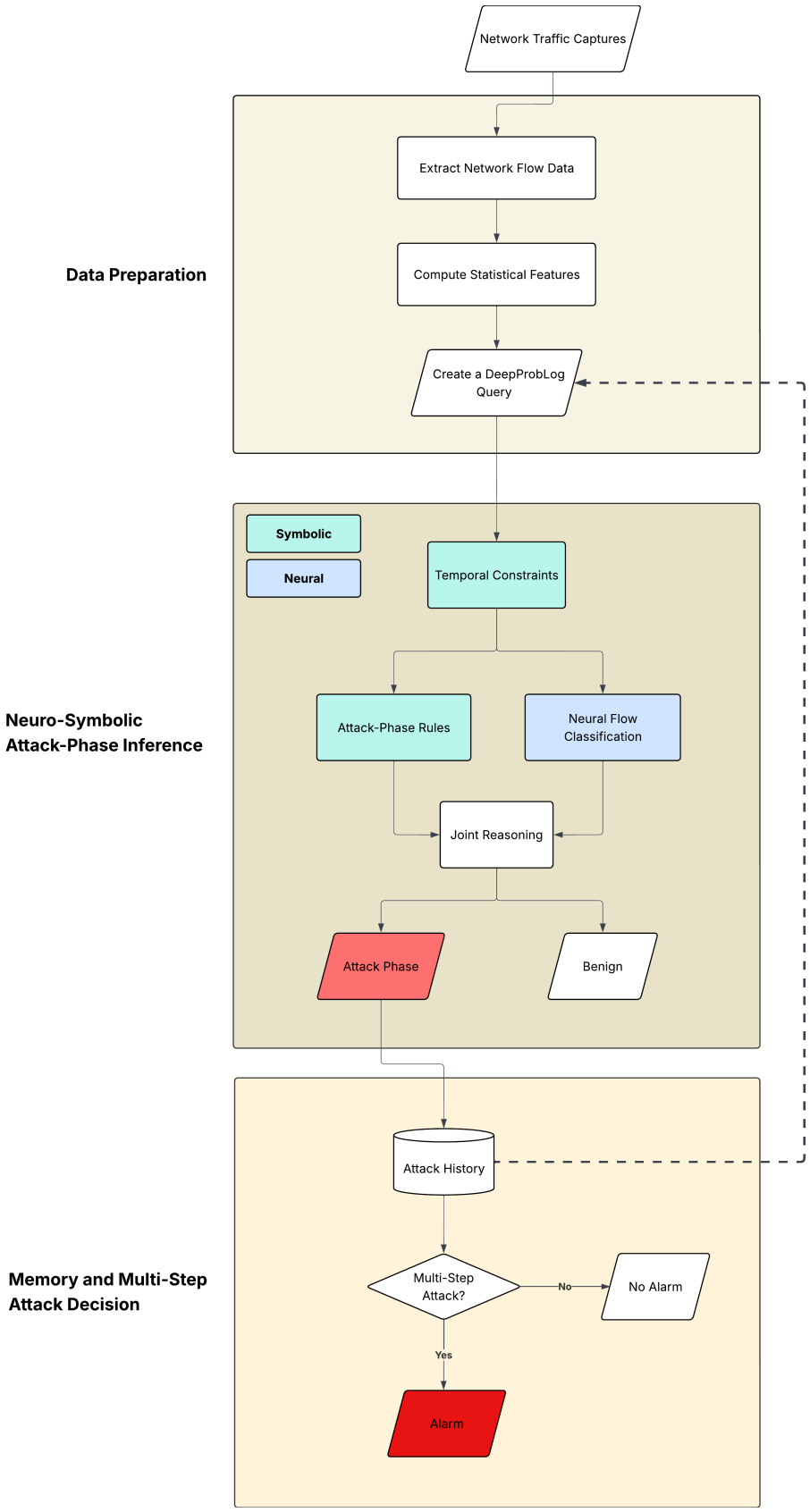


Figure 5.1: Overview of the proposed neuro-symbolic multi-step attack detection system.

We assume encrypted traffic throughout and therefore exclusively extract flow features derived from packet header fields.

5.2.2 Statistical Features

Certain attack behaviors cannot be fully captured using individual network flows, as they emerge from patterns across multiple flows and over time. In particular, early-stage activities such as reconnaissance are often characterized by anomalous interaction patterns rather than properties of a single connection. For example, a single source IP probing multiple ports or multiple destination hosts may indicate scanning behavior [24].

These patterns are inherently distributed and require a broader traffic context than that provided by a single flow, which is defined by a fixed 5-tuple. As a result, flow-level representations alone are insufficient to capture such global interaction patterns.

To address this limitation, and consistent with detection strategies in MITRE ATT&CK [23], we compute both source-centric and destination-centric statistical features over sliding time windows. These features capture aggregated behavior across multiple flows and complement the flow-based representation. Importantly, they can still be derived from encrypted traffic, as they rely solely on packet header information such as timestamps, IP addresses, ports, and connection metadata.

5.2.3 DeepProbLog Query

The DeepProbLog query is constructed from flow features (including the statistical features) and the attack history. It is passed to the neuro-symbolic inference component, which computes the probability p over possible outcomes $\{benign, phase_1, \ldots, phase_n\}$.

For example, the probability of the current flow being classified as $phase_1$ is queried as:

$$phase(C_{attack}, C_{flow}, Flow, phase_1)$$

Attack Context The attack context C_{attack} encodes which attack phases have been observed in the past. These are represented as binary attack-phase flags:

$$C_{attack} = \{F_1, \ldots, F_n\}$$

where $F_i = 1$ indicates that attack phase $phase_i$ has been observed, and $F_i = 0$ otherwise.

Flow Context The flow context C_{flow} contains the symbolic features used to evaluate whether a flow is consistent with known attack characteristics. These include protocol information, communication direction, and service-related attributes, which are commonly used in rule-based intrusion detection systems. Formally, a flow context is defined as:

$$C_{flow} = (Protocol, Origin_{src}, Origin_{dst}, Port_{src}, Port_{dst}).$$

Neural Input The remaining component, $Flow$, represents the input to the neural network. In this work, we exclude timestamps, IP addresses, and port numbers from the neural input. This design choice is motivated by the fact that such features are highly environment-specific and can lead to poor generalization [33]. In particular, attackers can easily change infrastructure, including IP addresses and port assignments, which may cause the neural model to learn patterns tied to a specific campaign rather than generalizable behavioral patterns.

Instead, these attributes are delegated to the symbolic component, which can explicitly encode and reason over them. This separation allows the neural network to focus on learning behavioral patterns that are more robust across different attack scenarios, while structural knowledge is handled through symbolic rules.

5.3 Neuro-Symbolic Attack-Phase Inference

At a high level, an attack phase $phase_i$ is inferred when its temporal prerequisites are satisfied, the observed event is consistent with predefined attack knowledge, and the neural component indicates that the current flow is malicious. This combines temporal constraints, symbolic reasoning, and neural predictions into a unified inference process.

This can be expressed in DeepProbLog as follows:

$$\begin{aligned} phase(C_{attack}, C_{flow}, Flow, phase_i) \ :- \ & plausible_attack_phase(C_{attack}, phase_i), \\ & attack_phase_rule(C_{flow}, phase_i), \\ & neural_evidence(Flow, phase_i). \end{aligned}$$

A flow is considered benign if it does not match any of the defined attack phases. In other words, benign behavior is defined as the absence of evidence for all known attack phases.

The corresponding DeepProbLog rule is:

$$\begin{aligned} phase(C_{attack}, C_{flow}, Flow, benign) \ :- \ & \neg\big(attack_phase(phase_i), \\ & phase(C_{attack}, C_{flow}, Flow, phase_i)\big). \end{aligned}$$

where $phase_i$ refers to any of the attack phases.

In DeepProbLog, negation $\neg$ follows the semantics of negation-as-failure [20], where the probability of a negated goal is given by $1-p$. Hence, the probability of a benign flow can be expressed as:

$$P(benign) = 1 - P\left(\bigvee_{i \in \{1,\ldots,N\}} phase_i\right)$$

In other words, a flow is considered benign with high probability when the probability of all attack phases is low.

5.3.1 Temporal Constraints

Due to the step-by-step nature of multi-step attacks, an attack phase $phase_i$ can only occur if the preceding phase $phase_{i-1}$ has been observed. This temporal structure can be encoded in the symbolic component to restrict predictions to temporally plausible attack steps.

Consider a simplified setting with three ordered phases, where $phase_1$ initiates the attack, $phase_2$ depends on $phase_1$, and $phase_3$ depends on both preceding phases. $phase_1$ is therefore always considered valid, while $phase_2$ and $phase_3$ are conditioned on the presence of earlier phases.

This structure can be encoded in DeepProbLog using the following rules:

$$plausible_attack_phase(F_1, F_2, F_3, phase_1).$$

$$plausible_attack_phase(F_1, F_2, F_3, phase_2) : -F_1 = 1.$$

$$plausible_attack_phase(F_1, F_2, F_3, phase_3) : -F_1 = 1, F_2 = 1.$$

During grounding, only attack phases whose temporal constraints are satisfied will be retained. For example, given the flag configuration $(1,0,0)$, only $phase_1$ and $phase_2$ remain valid, while $phase_3$ is excluded due to the unmet prerequisite $F_2 = 1$.

5.3.2 Attack Phase Rules

In addition to the temporal structure of multi-step attacks, each attack step exhibits characteristic patterns in terms of network behavior and protocol usage. These characteristics can be encoded as attack-phase rules in the symbolic component of the model.

The use of attack-phase rules provides several benefits. First, it provides the "structure" of an attack step for free, allowing the neural component to focus on learning discriminative patterns that are not explicitly modeled in the symbolic rules. Second, the symbolic component enables reasoning over features such as IP addresses and ports which are not passed to the neural component. Finally, the rule-based structure improves interpretability, as predictions can be traced back to explicit symbolic conditions.

To illustrate, consider an attack phase $phase_i$ where an external attacker probes a victim for a vulnerable service running on port 123 using the TCP protocol. This can be expressed as the following rule:

$$\begin{aligned} attack_phase_rule(Protocol, Origin_{src}, Origin_{dst}, Port_{dst}, phase_i) \ :- \ & Protocol = tcp, \\ & O_{src} = external, \\ & O_{dst} = internal, \\ & Port_{dst} = 123. \end{aligned}$$

5.3.3 Neural Flow Classification

The neural component is responsible for detecting malicious patterns in network flows. Its role is to learn low-level temporal and statistical representations that are difficult to capture using symbolic rules alone. The DeepProbLog framework is agnostic to the specific neural network architecture. As a result, the choice of architecture is a design decision that can be adapted to the characteristics of the detection task.

A multi-class attack-phase network can be defined as:

$$nn(net_{multiclass}, [Flow], Event, [benign, phase_1, \ldots, phase_N]) :: neural_evidence(Flow, Event).$$

Here, the argument $\texttt{net}_{\text{multiclass}}$ identifies the neural network model. `[Flow]` represents the network flow given as input to the model, while the output variable $Event \in \{benign, phase_1, \ldots, phase_N\}$ corresponds to the predicted class label. The predicate `neural_evidence` exposes the neural predictions to the symbolic reasoning component.

A multi-class formulation directly predicts the attack phase from the neural component. However, a binary formulation can also be adopted, where the network only distinguishes between *benign* and *malicious* traffic:

$$nn(net_{binary}, [Flow], Event, [benign, malicious]) :: neural_evidence(Flow, Event).$$

Another possible approach is to train a separate binary classifier for each attack phase:

$$\begin{aligned} nn(net_1, [Flow], Event, [benign, phase_1]) :: neural_evidence((Flow, Event). \\ \vdots \\ nn(net_n, [Flow], Event, [benign, phase_n]) :: neural_evidence((Flow, Event). \end{aligned}$$

The choice of neural architecture and classification strategy ultimately depends on the desired balance between model complexity, interpretability, and the amount of available training data.

5.4 Decision and Memory

The decision and memory components decouple higher-level operational logic from the neuro-symbolic inference process. These decisions, such as when to trigger an MSA alert or how long to retain predictions in memory, do not require neuro-symbolic reasoning. On the contrary, both components are designed to be configurable by security operators to support different deployment environments.

5.4.1 Attack Memory

As the neuro-symbolic inference component itself is stateless, the system maintains a lightweight attack memory to preserve contextual information across flows. The memory stores previously observed attack phases for each tracked attack context c.

An attack context c is defined by the IP addresses involved in the attack. This ensures that retrieved attack-phase flags remain relevant to the specific network flow being analyzed. For example, assume an attacker IP_{A1} first performs a scan and later exploits a vulnerability on the target IP_{T1}. In this case, several attack-phase flags will be set for the context associated with (IP_{A1}, IP_{T1}). However, these flags should not influence inference for a different attacker IP_{A2} interacting with the network. By maintaining separate contexts, the system can simultaneously track multiple attack campaigns while preventing unrelated attack histories from affecting predictions.

When an attack phase is detected, a new memory entry is created:

$$(c : (time_1, attack_phase_1, probability_1))$$

As additional attack phases are observed, the memory associated with the context is updated accordingly:

$$(c : (time_1, attack_phase_1, probability_1), \ldots, (time_i, attack_phase_i, probability_i))$$

Here, $time_i$ denotes the start time of the observed attack phase, $attack_phase_i$ represents the detected phase, and $probability_i$ corresponds to the confidence assigned by the neuro-symbolic model.

When processing a new flow, the system queries the memory using the IPs of the current flow. If a matching context exists, the corresponding attack history is retrieved and passed to the neuro-symbolic component to guide inference. Contexts are maintained for a configurable retention period and are removed once they exceed the retention window without further activity.

This memory component enables stateful detection of stealthy multi-step attacks that unfold gradually over extended periods of time. Additionally, it provides information about attack progression, which is useful for security analysts when deciding whether to initiate further investigation.

5.4.2 Alarm Decision

The final alarm decision is based on the accumulation of inferred attack phases within a given context. Rather than triggering alarms based on isolated malicious predictions, the system considers the temporal progression of phases as evidence of an ongoing multi-step attack.

This enables the incorporation of early attack phases such as reconnaissance, which are often noisy but important for early-stage detection. Individually, such phases may produce a large number of false positives. However, by requiring multiple related attack phases to occur within the same context, many of these isolated false positives can be filtered out before an alarm is raised.

An alarm is triggered when a context c satisfies a configurable detection condition based on the set of previously observed attack phases and their associated probabilities. By default, the condition requires the observation of at least two distinct attack phases within the same context. This reduces the likelihood that noisy or isolated predictions result in unnecessary alerts.

The alerting mechanism can be adapted to different operational environments by adjusting detection thresholds and phase requirements without modifying the underlying neuro-symbolic inference model.

5.5 Multi-Step Attack Knowledge

The proposed neuro-symbolic multi-step attack detection system relies on prior knowledge of multi-step attack behavior. This knowledge is essential for defining both the temporal rules and the attack-phase rules used in the symbolic component. A natural question is therefore how such knowledge is obtained.

Unlike single-step attacks, there is no standardized database that explicitly describes complete multi-step attack progressions. Instead, attack knowledge must be constructed from multiple sources that describe either individual attack characteristics or higher-level attacker behavior. In some cases, it can also be derived directly from analysis of multi-step attack scenarios.

However, due to the joint reasoning between the symbolic and neural components, there is no need to design overly specialized or highly tailored rules, as is typically required in purely signature-based systems. Instead, the system relies on agreement between neural evidence and symbolic constraints to infer attack phases.

5.5.1 Attack-Phase Rules

Knowledge of individual attack steps can be derived from publicly available intrusion detection rule sets such as Snort [35], Suricata [39], and Emerging Threats [8]. These rule sets describe observable network characteristics associated with known malicious activity, typically corresponding to exploitation or misuse of system vulnerabilities.

In addition to IDS rule sets, the MITRE ATT&CK framework [23] provides descriptions of common attacker tactics and techniques. These behavioral models are particularly useful for identifying early-stage attack phases such as reconnaissance [24].

5.5.2 Attack Progression

To map individual attack rules to a full multi-step attack, we rely on higher-level attack progression models. Frameworks such as the Cyber Kill Chain [19] and MITRE ATT&CK [23] describe how attacks typically evolve over time and which tactics are associated with each stage.

These frameworks encode common dependencies between phases. For example, command-and-control communication is unlikely to occur before successful exploitation or installation. Similarly, data exfiltration typically follows compromise and lateral movement.

5.5.3 Multi-Step Attack Sample Analysis

In addition to generic frameworks, multi-step attack knowledge can be derived from direct analysis of specific attack scenarios. Such analysis provides concrete information about the ordering of phases as well as the characteristics of each individual step within a given campaign.

Chapter 6

Implementation

In this chapter, we describe the implementation of the proposed neuro-symbolic multi-step attack detection system. Our primary focus is on the neuro-symbolic attack phase inference component, which constitute the core research contribution. As a result, we do not implement a fully end-to-end multi-step attack detection system as illustrated in Figure 5.1, but instead focus on the components necessary to evaluate this inference module in isolation.

We begin by introducing the two datasets used for training and evaluation. Next, we describe the implementation details of the DeepProbLog models for each dataset. Finally, we present the data preparation pipeline used to transform the raw data into a format suitable for model training and evaluation.

6.1 Datasets

This work evaluates the proposed approach using two publicly available multi-step attack (MSA) datasets: DARPA 2000 [18] and AIT Log Data Set V2 (AIT-LDS V2) [14, 15]. Both datasets contain network traffic packet captures (PCAPs) and labeled attack phases. The two datasets are chosen to provide complementary perspectives, combining a widely used benchmark with a more recent and realistic traffic environment.

6.1.1 DARPA 2000 Intrusion Detection Datasets

DARPA 2000 was produced by the Lincoln Laboratory and contains network activity from a small simulated Air Force network. The dataset include PCAPs recorded from both the inside and outside sensors (demilitarized zone, DMZ).

It is composed of two attack scenario datasets, each containing instances of a slightly different distributed denial-of-service (DDoS) attack. The first scenario, LLDOS 1.0, includes an attack run by a novice attacker, while the second scenario, LLDOS 2.0.2, features an attack conducted by a more stealthy attacker. The second scenario therefore contains fewer attack samples than the first. As neural models rely on sufficient data to learn patterns, we proceed with training and evaluating our model on traffic from the first scenario (both inside and DMZ).

The multi-step attack included in the first scenario comprise the following five attack phases [18]:

1. **Scanning**: the attacker sends ICMP Echo Requests to identify active hosts
2. **Probing:** the attacker sends sadmind ”ping” requests to detect the presence of the sadmind daemon running on Solaris hosts
3. **Exploitation:** the attacker exploits the sadmind vulnerability (CVE-1999-0977 [28]) to break into vulnerable hosts
4. **Installation:** installation of the mstream DDoS trojan software on the compromised hosts
5. **Launching DDoS:** execution of a distributed denial-of-service attack against an off-site server

6.1.2 AIT Log Dataset V2

The AIT Log Dataset V2 consists of eight synthetic datasets collected from testbeds built at the Austrian Institute of Technology (AIT) in 2022 [14, 15]. Each dataset represents a small enterprise network with a complex environment, diverse user behavior, and logs collected from all hosts in the network. The datasets contain 4-6 days of benign activity, during which a multi-step attack is launched at one of the final days.

The multi-step attack comprises the following six phases [14, 15]:

1. **Scanning:** the attacker performs several scans (host discovery, dns brute-force, service scan, dirb scan, wpscan) of the network using tools such as nmap, WPScan, and dirb
2. **Exploitation:** the attacker exploits a vulnerable plugin (CVE-2020-24186), allowing the upload of a PHP webshell as a backdoor. The attacker then executes several commands to gather information about the host
3. **Password Cracking:** the attacker downloads the password cracking tool John the Ripper and attempts to crack hashed passwords using a list of common passwords
4. **Privilege Escalation:** after obtaining the password, the attacker uploads a fully interactive reverse shell and misuses the compromised account to escalate their privileges to root level
5. **Remote Command Execution:** the attacker executes several commands, some of which require root privileges
6. **Data Exfiltration:** the attacker runs the DNSteal tool to exfiltrate sensitive data from a file share located in the intranet zone

Although all eight datasets include the same multi-step attack, they differ in characteristics such as scan order, scan duration, number of labeled phases, and event frequency. We choose to only proceed with two datasets for now, specifically *fox* and *santos*. Fox exhibits a high scan volume, while santos has a comparatively low scan volume, allowing us to evaluate model generalization across different operational conditions. The relevant dataset properties [14] are summarized in Table 6.1.

Table 6.1: Overview of the AIT Log Datasets V2. Adapted from [16]

Scenario	Simulation Time	Attack Time	Scan Volume	Size
fox	2022-01-15 to 2022-01-20	2022-01-18 11:59 to 13:15	High	26 GB
santos	2022-01-14 to 2022-01-18	2022-01-17 11:15 to 11:59	Low	17 GB

To the best of our knowledge, this is the first work to evaluate multi-step attack detection methods on AIT-LDS V2.

Modelled MSA

In order to properly evaluate a model's ability to detect multi-step attacks, we need to ensure that all attack phases in the dataset occur in a temporal order and are visible in the network traffic.

Since the dataset primarily was intended to be used as a log dataset, some attack phases rely on log information in order to be detected. Specifically, this applies to the fourth and fifth phases: privilege escalation and remote command execution. As we solely rely on network traffic, we exclude these phases from our envisioned multi-step attack.

Additionally, the authors chose to separate the data exfiltration step from the remaining attack phases by starting the exfiltration tool at the beginning of the simulation. The intent was to make the detection task more difficult for anomaly-based systems which rely on a training phase free of attacks [15]. As a result, it does not follow the logical step-by-step nature of a classical multi-step attack, since the final

objective occurs before any of the preceding phases. Therefore, we consider a multi-step attack sequence in which data exfiltration constitutes the initial step, which allows us to enforce a consistent temporal order.

As a result, we are effectively looking for the following attack phases in the specified order:

1. **Data Exfiltration**
2. **Scanning**
3. **Exploitation**
4. **Password Cracking**

6.2 Model Design

In this section, we describe the implementation of the DeepProbLog model for both the DARPA 2000 and AIT-LDS V2 datasets. In particular, we focus on the design of the neural component and the formulation of the temporal and attack-phase rules used in each dataset.

6.2.1 Neural Component

The neural component is implemented as a Long Short-Term Memory (LSTM) network. We select a temporal model because network flows are sequential, and malicious behavior often only becomes apparent when observing multiple related events over time. For instance, scanning activity is typically characterized by patterns across multiple packets rather than isolated events. An LSTM is therefore well-suited to capturing such temporal dependencies in network traffic.

The LSTM classifier consists of a single recurrent layer with 64 hidden units, followed by a fully connected layer with 32 neurons using the ReLU activation function. The output layer applies a softmax function to produce class probabilities. The architecture is intentionally kept simple to ensure that performance improvements can be more reliably attributed to the integration of symbolic reasoning rather than model complexity.

We train a single neural network that serves as the neural component of the model. Its role is to distinguish between *benign* and *malicious* flows. We adopt a binary classification setup, as different attack phases may exhibit similar low-level patterns and are therefore difficult to distinguish directly. For example, probing a specific vulnerability and exploitation may produce similar packet-level behavior. However, when combined with symbolic knowledge, these flows can be attributed to the correct attack phase based on higher-level contextual information.

The neural network is incorporated into the DeepProbLog model as a neural predicate, which outputs probabilities over the classes *benign* and *malicious*:

$$nn(lstm_{binary}, [Flow], Event, [benign, malicious]) :: neural_evidence(Flow, Event).$$

6.2.2 Temporal Rule

Temporal constraints in multi-step attack detection can be implemented with varying degrees of strictness. In particular, one may allow flexibility in the observed attack sequence (e.g., regression to earlier phases) or enforce a strictly forward progression through the attack stages.

This choice directly affects the trade-off between false alarms and missed detections: more permissive rules reduce number of missed attacks but may increase spurious transitions, while stricter rules reduce ambiguity at the cost of reduced flexibility.

In this work, we adopt a strict temporal ordering. Specifically, an attack phase p_i is only considered plausible if its prerequisite phase p_{i-1} has been observed, and none of the subsequent phases p_{i+1} have

yet occurred within the same context. Since the model operates on fully observed datasets with ground-truth attack progressions, the risk of missing earlier phases due to this constraint is not a concern in our setting.

Formally, we encode:

$$plausible_attack_phase(F_1, F_2, F_3, phase_1) : -F_1 = 0, F_2 = 0, F_3 = 0.$$

$$plausible_attack_phase(F_1, F_2, F_3, phase_2) : -F_1 = 1, F_2 = 0, F_3 = 0.$$

$$plausible_attack_phase(F_1, F_2, F_3, phase_3) : -F_1 = 1, F_2 = 1, F_3 = 0.$$

The temporal rules restrict inference to only two possible outcomes at a given step: the next attack phase or benign traffic. Consequently, the grounded program simplifies to

$$P(benign) = 1 - P(NextAttackPhase)$$

Hence, only two possible outputs.

6.2.3 Attack-Phase Rules

This subsection describes how attack-phase rules are constructed for the multi-step attacks in DARPA 2000 and AIT-LDS V2. The goal is to translate domain knowledge and existing intrusion detection rules into symbolic representations suitable for neuro-symbolic reasoning.

DARPA 2000

In DARPA 2000, attack steps exhibit well-defined behavioral patterns that can be directly encoded as symbolic rules. Rule construction is based on manual analysis of attack descriptions as well as conversion of existing intrusion detection rules from established rule sets [37]. We illustrate this process using Phase 1 and Phase 2 of the attack sequence.

Phase 1: Scanning. The initial attack phase corresponds to network scanning, where the attacker sends ICMP Echo Requests from an external source into the target network [18]. This behavior can be directly expressed as a symbolic rule:

$$\begin{aligned} phase_1_rule(Protocol, Src, Dst) \ : - \ & Protocol = icmp, \\ & Src = external, \\ & Dst = internal. \end{aligned}$$

Phase 2: Sadmind probing. Phase 2 corresponds to Sadmind "ping" requests (CVE-1999-0977 [28]). The corresponding detection logic is derived from the Snort community ruleset [37]. Snort rules consist of a rule header, which specifies protocol, address ranges, and ports, and a rule body, which defines payload-matching conditions [36].

Since this work assumes encrypted traffic, only header-level information is used. The relevant rule header is given by:

$$alert\ udp\ \$EXTERNAL_NET\ any \rightarrow \$HOME_NET\ any\ (\ldots)$$

Based on this abstraction, the corresponding symbolic representation in DeepProbLog is:

$$\begin{aligned} phase_2_rule(Protocol, Origin_{src}, Origin_{dst}) \ : - \ & Protocol = udp, \\ & Origin_{src} = external, \\ & Origin_{dst} = internal. \end{aligned}$$

AIT-LDS V2

In contrast to DARPA 2000, AIT-LDS V2 does not exhibit strongly structured attack phases. As a result, rule construction relies less on external rule bases and more on dataset-driven inspection of attack behavior.

According to the dataset documentation, the attacker gains initial access via VPN prior to the attack sequence [14, 15]. We therefore assume that both source and destination traffic originate within the internal network, and encode this as a shared constraint across rules.

Phase 1: Data Exfiltration. Phase 1 corresponds to data exfiltration via DNS traffic. DNS traffic runs over port 53. Hence, this behavior is encoded as:

$$\begin{aligned} phase_1_rule(Origin_{src}, Origin_{dst}, Port_{dst}) \ :- \ & Origin_{src} = internal, \\ & Origin_{dst} = internal, \\ & Port_{dst} = 53. \end{aligned}$$

Phase 2: Scanning. Phase 2 corresponds to scanning activity over TCP traffic. Based on dataset inspection, this behavior is consistently associated with internal-to-internal TCP communication, and is encoded as:

$$\begin{aligned} phase_2_rule(Protocol, Origin_{src}, Origin_{dst}) \ :- \ & Protocol = tcp, \\ & Origin_{src} = internal, \\ & Origin_{dst} = internal. \end{aligned}$$

The same principle is applied to the remaining attack phases.

Rule Encoding

In the implementation, symbolic attributes are mapped to numerical features. Specifically, protocols are represented using IP protocol numbers [42], while source and destination locations are encoded as binary indicators (1 for internal, 0 for external). To illustrate this encoding, the Phase 2 rule from the DARPA dataset can be represented as:

$$\begin{aligned} phase_2_rule(Protocol, Origin_{src}, Origin_{dst}) \ :- \ & Protocol = 17, \\ & Origin_{src} = 0, \\ & Origin_{dst} = 1. \end{aligned}$$

6.3 Data Preparation

The data preparation pipeline consists of four stages: (i) flow processing, (ii) labeling, (iii) neural input construction, and (iv) DeepProbLog query generation.

6.3.1 Flow Processing

Raw network traffic in PCAP format is transformed into flow-level representations using the Zeek network security monitor [44]. Zeek generates multiple log files describing different aspects of network activity. However, since this work assumes encrypted traffic, only the *conn.log* file is used, as it contains connection-level metadata derived solely from packet headers.

DARPA 2000. The DARPA 2000 dataset contains multiple PCAP subsets, including a full capture (benign and attack traffic) and five additional subsets corresponding to individual attack phases.

Flow labeling is performed by executing Zeek on all PCAP files and matching flows via a hash-based comparison. The hash is computed over all flow attributes except the unique flow identifier. Flows in the full capture are then assigned labels by matching against the phase-specific subsets. Flows without a match are labeled as benign (class 0). This results in the label distribution shown in Table 6.2.

Scenario	Benign	$Phase_1$	$Phase_2$	$Phase_3$	$Phase_4$	$Phase_5$
Inside	52258	20	22	34	19	73472
DMZ	10609	767	25	79	19	33942

Table 6.2: Label distribution across attack phases for DARPA 2000 Inside and DMZ datasets.

AIT-LDS V2. The AIT-LDS V2 dataset consists of multiple PCAP files collected from different sensors. All files are processed using Zeek and merged into a single chronologically ordered dataset based on `start_time`. Following the dataset documentation [14, 15], we restrict the data to the official simulation period and remove duplicate flows originating from the same sensor with identical feature values.

Unlike DARPA 2000, AIT-LDS V2 provides both PCAP-based data and pre-labeled tstat flows [14, 15]. Since tstat and Zeek produce different flow representations, direct matching is not possible. Instead, flows are aligned using a combination of the 4-tuple (src_ip, dst_ip, src_port, dst_port) and temporal proximity. The resulting label distribution is shown in Table 6.3.

The original labels are more fine-grained than the multi-step attack phases used in this work. Since no explicit mapping is provided in the dataset publication, we construct a mapping based on the accompanying labeling notebook.

Scenario	Benign	$Phase_1$	$Phase_2$	$Phase_3$	$Phase_4$
Fox	400024	14678	56443	28	61
Santos	333964	12861	5670	26	228

Table 6.3: Label distribution across attack phases for AIT-LDS V2 Fox and Santos datasets.

6.3.2 Neural Input Preparation

Before being provided to the neural component, the network flow data must be preprocessed and transformed into a suitable numerical representation. This includes selecting relevant flow features, computing statistical features, processing the features and creating windowed input for the temporal neural models.

Flow Features. Table 6.4 summarizes all available Zeek connection log features and how they are used in the system. The neuro-symbolic component makes use of most fields from the Zeek *conn.log* file. However, certain features are excluded due to irrelevance or computational overhead. Specifically, the *UID* field, a unique flow identifier, is removed due to its uniqueness per connection, *history* is excluded due to its high-dimensionality when encoded, and *tunnel_parents* is absent in all evaluated datasets.

Statistical Features. We compute the statistical features over a sliding time window of 60 seconds. The size of this time window is configurable. Similar to the window size of the neural component, the length of the time window decide which behaviors can be detected. A larger time window allow the model to learn long-ranging scanning, while a smaller window size. A window size of 60 was set based on experimental evaluation.

From a source perspective, we compute the number of active connections initiated within the window (*connections_per_src*), the number of unique destination IPs contacted (*unique_targets*), and the

Table 6.4: Zeek connection log features and their role in the system.

Feature	Description	Usage
ts	Timestamp of the connection	Symbolic
id.orig_h	Source IP address	Symbolic
id.resp_h	Destination IP address	Symbolic
id.orig_p	Source port	Symbolic
id.resp_p	Destination port	Symbolic
local_orig	Local origin (T/F)	Symbolic + Neural
local_resp	Local response (T/F)	Symbolic + Neural
proto	Transport protocol (TCP/UDP/ICMP)	Symbolic + Neural
service	Application protocol (e.g., HTTP, DNS)	Neural
duration	Connection duration	Neural
orig_bytes	Bytes sent by originator	Neural
resp_bytes	Bytes sent by responder	Neural
conn_state	Connection state	Neural
missed_bytes	Missed bytes in capture	Neural
orig_pkts	Packets sent by originator	Neural
orig_ip_bytes	IP bytes sent by originator	Neural
resp_pkts	Packets sent by responder	Neural
resp_ip_bytes	IP bytes sent by responder	Neural
ip_proto	IP protocol number	Neural
uid	Unique connection ID	Not used
history	Connection history flags	Not used
tunnel_parents	Tunnel parents	Not used

number of unique destination ports accessed (*unique_dports*). In addition, we compute connection-state ratios that capture abnormal connection outcomes. Specifically, the fraction of SYN failures (*syn_failure_ratio*), connection rejections (*reject_ratio*), and connection resets (*reset_ratio*).

From a destination perspective, we compute the number of incoming connections (*connections_per_dst*) and the number of unique source IPs contacting the destination (*unique_sources_per_dst*), which are indicative of targeted attacks or distributed probing.

Feature Processing. Neural input features are processed according to their data type, as summarized in Table 6.5. Categorical features are transformed using one-hot encoding, while numerical features are normalized using MinMax scaling to ensure a consistent input representation.

Table 6.5: Feature taxonomy used in the neural model.

Type	Features
Categorical	local_orig, local_resp, proto, service, conn_state, ip_proto
Numerical	duration, orig_bytes, resp_bytes, missed_bytes, orig_pkts, orig_ip_bytes, resp_pkts, resp_ip_bytes, connections_per_src, unique_targets, unique_dports, connections_per_dst, unique_sources_per_dst, syn_failure_ratio, reject_ratio, reset_ratio

Windowing. The final step of neural input preparation is the construction of temporal windows. Each window consists of a sequence of flows ordered by their timestamp and provides the input used to predict the attack phase of the current flow.

Window size determines how much historical context is available to the model. While larger windows may capture longer-term attack behavior, they also introduce additional noise and increase inference latency. To balance contextual information with real-time applicability, we evaluate window sizes of 10 and 100 flows.

6.3.3 Query Generation

Due to the absence of a fully integrated end-to-end system, several adaptations are required in the construction of DeepProbLog queries during evaluation.

Attack History. DeepProbLog queries require attack-history flags that indicate whether previous attack phases have been observed. In a fully deployed system, these flags would be derived from prior model predictions stored in an attack-history database. However, since the full system is not implemented, no such runtime history is available.

To isolate the effect of the temporal reasoning component, attack-history flags are instead generated from the ground-truth attack sequences provided by the datasets. Consequently, all queries are constructed under the assumption that preceding attack phases have been correctly identified. This design removes error propagation from earlier predictions and enables an independent evaluation of the temporal logic component.

Context Definition. Attack-history flags are associated with a context c, which is defined by the IP addresses involved in the attack. However, due to the structure and characteristics of certain attack steps in the evaluated datasets, this IP-based correlation does not hold consistently across all phases.

In AIT-LDS V2, the hosts involved in the initial data exfiltration phase are not the same as those involved in subsequent scanning activity. This intentionally non-chronological ordering is designed to increase the difficulty of the detection task [14, 15].

Similarly, in DARPA 2000, the distributed denial-of-service phase is executed using multiple, potentially spoofed, source IP addresses. As a result, the IP addresses observed in earlier phases do not correspond to those involved in the final attack stage.

To address this mismatch, attack-history flags are propagated globally over a fixed time horizon for both datasets once these phases are reached, ensuring that correlation does not break.

Chapter 7

Experimental Evaluation

This chapter presents the experimental evaluation of the proposed neuro-symbolic approach for multi-step attack detection. We conduct several experiments in order to address our research questions regarding attack step detection, temporal consistency, data requirements, generalization and system feasibility. Through these experiments we investigate whether the neuro-symbolic model can (i) reliably detect individual attack steps, (ii) enforce temporally valid predictions, (iii) learn effectively from limited training data, (iv) learn representations that generalize to unseen datasets, (v) remain robust in the presence of missed attack phases, and (vi) achieve computational performance suitable for practical deployment.

The chapter begins by introducing the evaluation metrics used throughout the study. It then presents the baselines and outlines six experiments, along with their corresponding results.

7.1 Evaluation Metrics

The neuro-symbolic component performs a multi-class classification task. Therefore, we record several metrics that evaluate the model's performance across individual attack steps rather than solely focusing on when a final multi-step attack alarm is raised.

7.1.1 Traditional Metrics

Several standard classification metrics were recorded, including accuracy, F1-scores (macro and weighted), false alarm rate (FAR), and detection rate (DR). In addition, per-class F1-scores were computed. Accuracy and F1-scores were evaluated over all classes, both benign traffic and attack phases. The false alarm rate was computed as FAR = $\frac{\text{\# benign predicted as attack}}{\text{\# true benign samples}}$, while the detection rate was computed as DR = $\frac{\text{\# correctly detected attacks}}{\text{\# true attack samples}}$.

Among these metrics, the macro F1-score is considered the most informative indicator of overall performance. It assigns equal weight to each class and therefore highlights the models performance on minority attack classes. However, since the DPL models are not trained with explicit class balancing, macro F1 may underrepresent performance in frequently occurring classes while overemphasizing rare ones. For this reason, it is interpreted in combination with the weighted F1-score, which accounts for class frequencies and provides a more realistic view of model performance.

7.1.2 Temporal Metrics

Traditional evaluation metrics do not capture a model's ability to learn the temporal dependencies of multi-step attacks. Therefore, we define two custom temporal metrics: causal violation rate (CVR) and regression violation rate (RVR).

Causal violations and regression violations represent two different types of temporal inconsistencies. A causal violation occurs when the model predicts attack phase p_{i+1} before phase p_i has been observed.

A regression violation occurs when the model has already progressed to phase p_i but subsequently predicts an earlier attack phase.

The causal violation rate was computed as $\text{CVR} = \frac{\#\text{ causal violations}}{\#\text{ total violations}}$, while the regression violation rate was computed as $\text{RVR} = \frac{\#\text{ regression violations}}{\#\text{ total violations}}$.

Causal violations are considered more severe than regression violations. From a logical perspective, causal violations should not occur, as each attack step depends on the prior occurrence of prerequisite steps. Regression violations, on the other hand, may occur in practice if an attacker revisits previous attack stages. Nevertheless, they still represent temporal inconsistencies which should be possible to minimize through the incorporation of logic.

7.2 Baselines

We implemented both neural and symbolic-based baselines for comparison. In addition, we compared our approach to Kitsune [22], a state-of-the-art anomaly-based intrusion detection system.

Neural Baselines We implemented two neural baselines: a multiclass LSTM and an ensemble LSTM. Both models use the same underlying architecture as the neural component of our system (see subsection 6.2.1). However, rather than performing binary classification, the baseline models directly predict the attack step through a multiclass output layer. The ensemble architecture is similar to the multiclass model but replaces the shared LSTM with multiple LSTMs (one per attack phase), whose outputs are combined through a linear prediction layer.

Symbolic Baselines The symbolic baselines are the DeepProbLog programs with the neural component removed. This isolates the contribution of the logical reasoning component and provides insight into the extent to which performance can be attributed to the logic alone. Comparing these baselines with the full model allows us to assess whether the neural network provides meaningful predictive capability beyond that achieved by the logical rules.

Kitsune Our implementation of Kitsune follows the original GitHub repository provided by the authors [22], with minor modifications to integrate it into our experimental pipeline. In particular, we replace Kitsune's original feature extraction and preprocessing pipeline with the feature representation used in our dataset to ensure a consistent input format across all compared methods. This allows us to isolate differences in detection performance rather than differences in feature engineering.

Given that Kitsune is an anomaly detector, all attack samples are removed from the training set. Furthermore, Kitsune processes one flow at a time and therefore does not require windowed input. Consistent with its online detection setting, flows are presented in their original temporal order rather than being shuffled, allowing the model to incrementally learn normal network behavior as intended by the original design.

Since Kitsune is an unsupervised anomaly detector, it does not produce explicit predictions for individual attack steps. Consequently, we evaluate it in terms of overall detection performance (i.e., benign vs. anomalous traffic). In addition, we analyze its detection behavior with respect to individual attack steps by mapping anomaly detections to the corresponding labeled phases, in order to gain insight into its behavior in multi-step attack scenarios.

7.3 Experiments

The datasets were divided into training and testing sets using a stratified 70/30 split, which ensured that samples from all attack phases were represented in both subsets. Both the baseline models and the DPL models were trained for a maximum of 50 epochs.

Table 7.1: Top 8 neural baseline configurations on AIT-LDS V2 ranked by macro F1-score. The best value per metric is shown in bold.

Model	Macro F1	Accuracy	DR	FAR
$LSTM_{\text{Ensemble}}(\text{base}, w = 100, lr = 10^{-3})$	**0.8650**	**0.9966**	**0.9874**	**0.0029**
$LSTM_{\text{Multiclass}}(\text{base}, w = 100, lr = 10^{-3})$	0.8213	0.9952	0.9865	0.0043
$LSTM_{\text{Ensemble}}(\text{base}, w = 100, lr = 10^{-4})$	0.7498	0.9907	0.9867	0.0090
$LSTM_{\text{Multiclass}}(\text{base}, w = 100, lr = 10^{-4})$	0.6919	0.9874	0.9846	0.0124
$LSTM_{\text{Multiclass}}(\text{base}, w = 10, lr = 10^{-3})$	0.6749	0.9923	0.9756	0.0067
$LSTM_{\text{Ensemble}}(\text{base}, w = 10, lr = 10^{-3})$	0.6741	0.9924	0.9760	0.0066
$LSTM_{\text{Multiclass}}(\text{flow-only}, w100, lr = 10^{-3})$	0.6367	0.9307	0.9706	0.0715
$LSTM_{\text{Ensemble}}(\text{base}, w = 10, lr = 10^{-4})$	0.6365	0.9880	0.9742	0.0112

Table 7.2: Top 8 neural baseline configurations on DARPA 2000 ranked by macro F1-score. The best value per metric is shown in bold.

Model	Macro F1	Accuracy	DR	FAR
$LSTM_{\text{Ensemble}}(\text{base}, w = 10, lr = 10^{-3})$	**0.9152**	**0.99958**	0.99957	0.00030
$LSTM_{\text{Multiclass}}(\text{base}, w = 10, lr = 10^{-3})$	0.9142	0.99957	0.99953	**0.00027**
$LSTM_{\text{Ensemble}}(\text{flow-only}, w = 10, lr = 10^{-3})$	0.9131	0.99955	**0.99959**	0.00041
$LSTM_{\text{Multiclass}}(\text{flow-only}, w = 10, lr = 10^{-3})$	0.8850	0.99949	0.99955	0.00046
$LSTM_{\text{Ensemble}}(\text{base}, w = 100, lr = 10^{-3})$	0.8843	0.99955	0.99953	0.00033
$LSTM_{\text{Ensemble}}(\text{flow-only}, w = 100, lr = 10^{-3})$	0.8817	0.99953	0.99951	0.00030
$LSTM_{\text{Multiclass}}(\text{flow-only}, w = 100, lr = 10^{-3})$	0.8793	0.99951	0.99953	0.00033
$LSTM_{\text{Ensemble}}(\text{flow-only}, w = 10, lr = 10^{-4})$	0.8783	0.99944	0.99953	0.00055

7.3.1 Ablation Study

To identify suitable hyperparameter settings for the subsequent experiments, we conducted an ablation study using the two neural baseline models. We experimented with feature sets, window sizes, and learning rates (see Table B.1 in the Appendix for all evaluated configurations). The flow-only configuration consists exclusively of flow features, while the base configuration augments these flow features with the statistical features.

The models were evaluated using 5-fold cross-validation with early stopping. Because the datasets are highly imbalanced, a weighted sampler was employed during training to mitigate bias toward the benign class and majority attack classes. Full neural baseline results are provided in Appendix B.1, Section B.1.

AIT-LDS V2. Table 7.1 presents the top-performing configurations on the AIT-LDS V2 dataset. The best-performing configuration across all evaluation metrics was the ensemble LSTM with the base feature set, a window size of 100, and a learning rate of 10^{-3}. The second-best configuration was the multiclass LSTM using the same feature set, window size, and learning rate. Overall, configurations using the base feature set and larger window size consistently achieved the strongest performance.

DARPA 2000. Table 7.2 presents the top-performing configurations on the DARPA 2000 dataset. The best-performing configuration across most evaluation metrics was the ensemble LSTM with the base feature set, a window size of 10, and a learning rate of 10^{-3}. The second-best configuration was the multiclass LSTM using the same feature set, window size, and learning rate. In contrast to AIT-LDS V2 results, the highest performing DARPA configurations consistently employed the shorter window size.

Table 7.3: Performance and temporal consistency on AIT-LDS V2. Results are averaged over five runs (and three runs for $DPL_{Pretrained}$ T). Standard deviations are reported only for macro F1. Models are ranked by macro F1-score. Best results for CVR and RVR are highlighted in bold.

Model	Macro F1 (± std)	Weighted F1	Accuracy	DR	FAR	CVR	RVR
$DPL_{Pretrained}$ T+R	0.944 ± 0.024	0.999	0.999	0.990	0.0003	**0.000**	**0.002**
DPL_{E2E} T+R	0.940 ± 0.010	0.999	0.999	0.991	0.0005	**0.000**	0.014
$LSTM_{Multiclass}$	0.880 ± 0.015	0.998	0.998	0.988	0.0018	0.019	0.021
$DPL_{Pretrained}$ T	0.878 ± 0.020	0.998	0.998	0.983	0.0008	**0.000**	0.152
$LSTM_{Ensemble}$	0.858 ± 0.024	0.998	0.998	0.991	0.0019	0.027	0.017
DPL_{E2E} T	0.842 ± 0.013	0.998	0.998	0.981	0.0008	**0.000**	0.180
Kitsune	0.645 ± None	0.938	0.947	0.231	0.01249	None	None
$Logic_{Baseline}$ T+R	0.631 ± 0.038	0.917	0.925	0.714	0.0627	**0.000**	0.047
$Logic_{Baseline}$ T	0.092 ± 0.000	0.100	0.053	1.000	1.000	**0.000**	0.200

The best-performing configurations identified in the ablation study are used for all subsequent experiments. These settings are applied both to the neural baseline models and the neural component of the DPL models.

Takeaways

Feature set and window size. The base feature set consistently outperformed the flow-only feature set, and the highest-performing configurations generally used a learning rate of 10^{-3}. While the ensemble LSTM achieved the strongest overall performance on both datasets, the optimal window size was dataset-dependent: AIT-LDS V2 benefited from longer temporal context ($w = 100$), whereas DARPA 2000 achieved better performance with shorter windows ($w = 10$).

7.3.2 Symbolic Knowledge Study

This experiment investigates whether incorporating symbolic knowledge can help the model reliably detect individual attack steps and enforce temporally valid attack progressions. We evaluate two DPL variants: one that incorporates temporal constraints only (T), and one that incorporates both temporal constraints and attack-phase rules (T+R). The DPL models are compared against the neural baselines, symbolic baselines, and Kitsune.

For each DPL variant, we evaluate two training strategies: end-to-end (E2E) training and pretraining. In the end-to-end setting, the neural and symbolic components are optimized jointly from scratch. In the pretraining setting, the neural component is first trained independently on the training data and subsequently integrated into the DPL framework, where it is further optimized together with the symbolic component using the same training set. This comparison allows us to assess the impact of joint neuro-symbolic optimization on detection performance.

AIT-LDS V2. Table 7.3 summarizes the results of the symbolic knowledge study on AIT-LDS V2. The best-performing configuration is the pretrained DPL model incorporating both temporal and attack-phase knowledge. However, the end-to-end variant achieves comparable performance across most evaluation metrics. Both models substantially outperform the remaining approaches in terms of macro F1-score, even though the other DPL variants and neural baselines all achieve macro F1-scores above 0.84.

In contrast, Kitsune and the symbolic baselines achieve considerably lower macro F1-scores and higher false alarm rates. Inspecting the confusion matrix of Kitsune (Figure 7.1a) shows that it is only able to detect phase 2 of the attack, while the remaining phases are largely missed.

Table 7.4: Performance and temporal consistency on DARPA dataset. Results are averaged over three runs. Standard deviations are reported only for macro F1. Models are ranked by macro F1-score. Best results for CVR and RVR are highlighted in bold.

Model	Macro F1 (± std)	Weighted F1	Accuracy	DR	FAR	CVR	RVR
Kitsune	0.984 ± None	0.984	0.984	0.999	0.03719	None	None
DPL_{E2E} T+R	0.940 ± 0.014	0.999	0.999	0.999	0.00015	**0.000**	**0.000**
$DPL_{Pretrained}$ T+R	0.892 ± 0.006	0.999	0.999	0.999	0.00009	**0.000**	**0.000**
$Logic_{Baseline}$ T+R	0.891 ± 0.079	0.963	0.962	0.999	0.09021	**0.000**	0.001
$DPL_{Pretrained}$ T	0.872 ± 0.013	0.999	0.999	0.999	0.00021	**0.000**	**0.000**
DPL_{E2E} T	0.839 ± 0.019	0.999	0.999	0.999	0.00015	**0.000**	0.014
$LSTM_{Multiclass}$	0.827 ± 0.016	0.999	0.999	0.999	0.00030	0.027	0.081
$LSTM_{Ensemble}$	0.820 ± 0.014	0.999	0.999	0.999	0.00032	0.053	0.092
$Logic_{Baseline}$ T	0.558 ± 0.000	0.655	0.585	1.000	0.99834	**0.000**	0.144

DARPA 2000. Table 7.4 summarizes the results on DARPA 2000. Kitsune achieves a strong macro F1-score of 0.984. The confusion matrix in Figure 7.1b shows that Kitsune is able to reliably detect all attack classes. However, this comes at the cost of a higher false alarm rate, indicating a tendency to misclassify benign traffic as anomalous.

Among the DPL models, those incorporating both temporal and attack-phase knowledge achieve the strongest overall performance across most metrics. However, the end-to-end variant achieves a noticeably higher macro F1-score, suggesting that jointly learning neural representations and symbolic constraints is more effective than relying solely on pretrained representations.

The corresponding symbolic baseline also achieves competitive macro F1-scores, demonstrating that the symbolic knowledge itself captures useful attack characteristics. However, its lower accuracy and higher false alarm rate indicate weaker discrimination of majority classes and less stable overall performance.

Temporal Consistency. Across both datasets, the neural baselines consistently underperform compared to the best-performing DPL configurations (see Table 7.3 and Table 7.4). In addition, they are the only models that exhibit non-zero causal violations, suggesting weaker temporal consistency in their predictions. In contrast, the DPL models substantially reduce both causal and regression violations, indicating improved temporal consistency under the proposed modeling constraints.

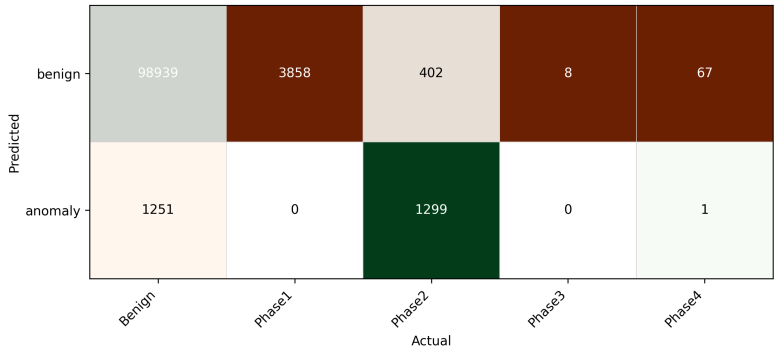
Figures 7.2 and 7.3 further illustrate these effects. Temporal logic enforces strict ordering constraints that eliminate causal violations across all settings. The inclusion of attack-phase rules further reduces regression violations, resulting in more temporally consistent predictions overall.

Since Kitsune is an anomaly detector that only distinguishes between benign and anomalous traffic, temporal consistency metrics are not applicable and are therefore omitted.

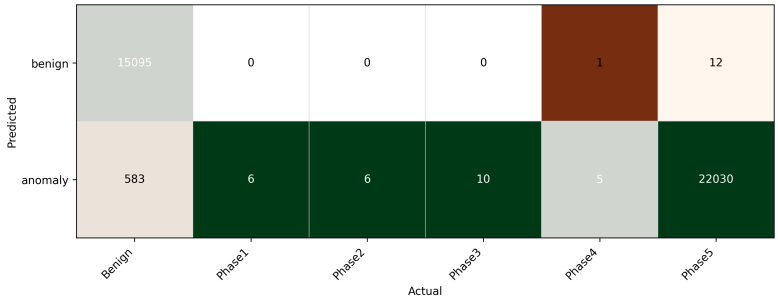
Takeaways

Temporal and attack-phase knowledge. Incorporating temporal and attack-phase knowledge improves attack-step detection performance while substantially reducing temporal inconsistencies. The strongest results are consistently obtained when both knowledge sources are combined.

End-to-end learning. Joint optimization of the neural and symbolic components generally yields the strongest performance, suggesting that the neural representations benefit from being learned under symbolic constraints rather than incorporated only after pretraining.

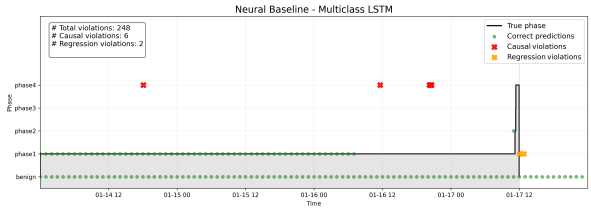


(a) AIT-LDS V2

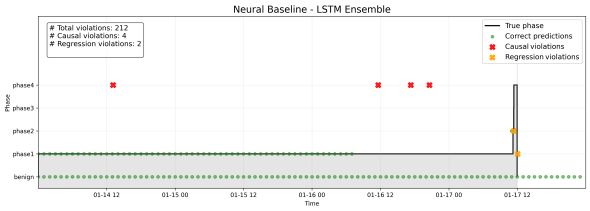


(b) DARPA 2000

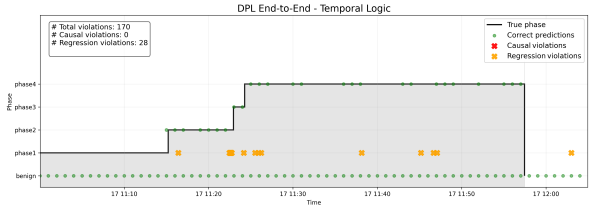
Figure 7.1: Confusion matrices for Kitsune on the AIT-LDS V2 and DARPA 2000 datasets. The matrices illustrate the distribution of correct predictions and misclassifications across the evaluated classes, highlighting Kitsune's performance per attack-phase.



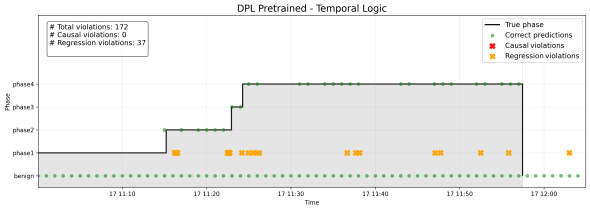
(a) Multi-class LSTM – Full attack window



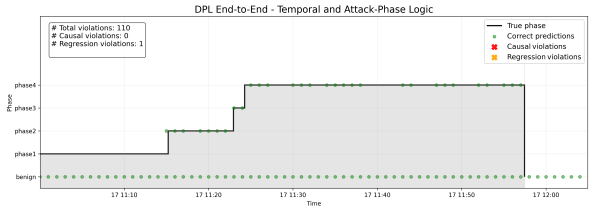
(b) Ensemble LSTM – Full attack window



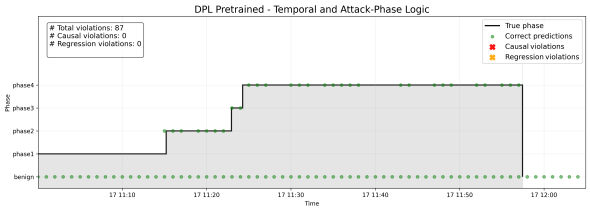
(c) Temporal Logic Only – End-to-End



(d) Temporal Logic Only – Pre-trained

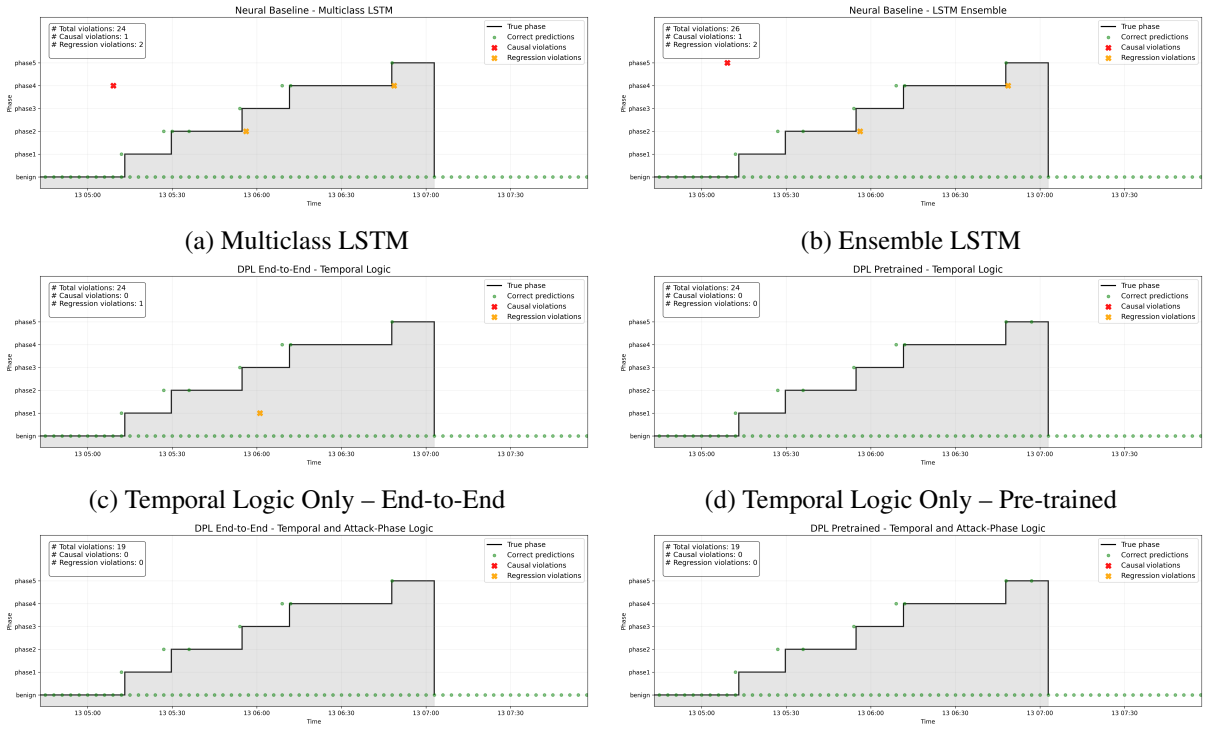


(e) Temporal and Attack-Phase Logic – End-to-End



(f) Temporal and Attack-Phase Logic – Pre-Trained

Figure 7.2: Temporal consistency comparison across neural and neuro-symbolic models on AIT-LDS V2, highlighting differences between baseline LSTMs and DPL models.



(a) Multiclass LSTM

(b) Ensemble LSTM

(c) Temporal Logic Only – End-to-End

(d) Temporal Logic Only – Pre-trained

(e) Temporal and Attack-Phase Logic – End-to-End

(f) Temporal and Attack-Phase Logic – Pre-Trained

Figure 7.3: Temporal consistency comparison across neural and neuro-symbolic models on DARPA 2000, highlighting differences between baseline LSTMs and DPL models.

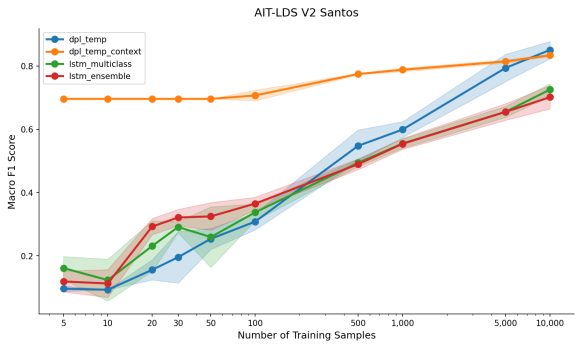
7.3.3 Number of Training Samples

This experiment evaluates how the performance of DeepProbLog models and neural baselines varies under different levels of training data availability. To ensure balanced learning conditions, we construct training subsets with an equal number of benign and attack samples per attack step. For each dataset size, results are averaged over five independent runs per model.

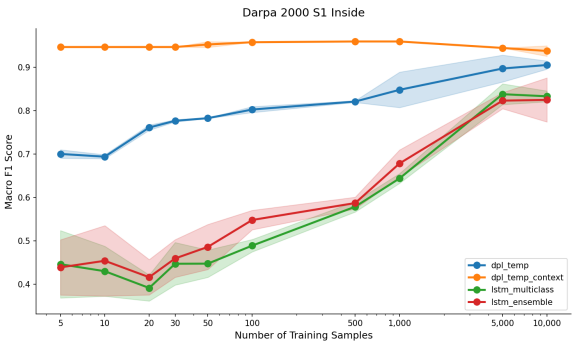
Low-Data Regime Performance. Figure 7.4 shows results for AIT-LDS V2 (subfigures a, c, and e) and DARPA 2000 (subfigures b, d, and f). Across both datasets, the DPL model with temporal and attack-phase knowledge consistently achieves strong performance in low-data regimes. In contrast, temporal knowledge alone exhibits dataset-dependent behavior: on DARPA it yields strong initial performance, while on AIT-LDS V2 it leads to slower early convergence.

Scaling with Increasing Data. As the number of training samples increases, the neural baselines gradually approach the performance of the DPL model with temporal and attack-phase knowledge. This convergence occurs at approximately 10,000 samples for AIT-LDS V2 and 5,000 samples for DARPA 2000. However, even in higher data regimes, models incorporating symbolic knowledge maintain a consistent advantage in macro F1.

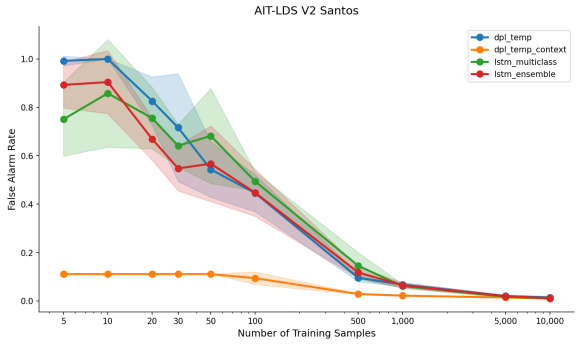
Learning Stability. The error bars indicate that the DPL model with temporal and attack-phase knowledge exhibits the most stable learning behavior across runs. In contrast, neural baselines show higher variance across all data regimes. On AIT-LDS V2, minor fluctuations are observed for the DPL model with temporal and attack-phase knowledge at around 100 samples. A similar effect appears on DARPA at approximately 50 samples. The temporal-only DPL model shows increased variability across most sample sizes on AIT-LDS V2, while on DARPA higher variance in macro F1 emerges beyond approximately 1,000 samples.



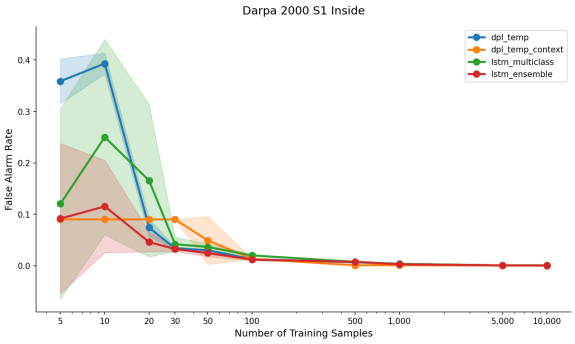
(a) AIT: Macro F1



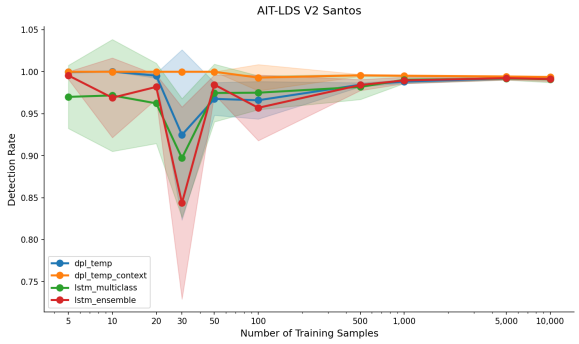
(b) DARPA: Macro F1



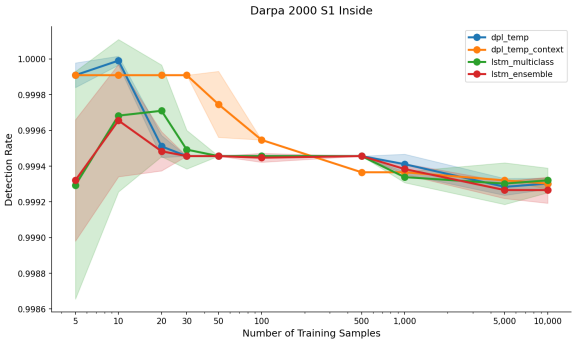
(c) AIT: False alarm rate



(d) DARPA: False alarm rate



(e) AIT: Detection rate



(f) DARPA: Detection rate

Figure 7.4: Effect of training set size on model performance across AIT-LDS V2 (left) and DARPA 2000 (right). Results show how macro F1, FAR, and DR vary as the number of training samples increases for different model configurations.

Takeaways

Data efficiency. Temporal and attack-phase symbolic knowledge improves data efficiency, enabling strong performance in low-data regimes and reducing the number of training samples required compared to purely neural baselines.

7.3.4 Cross-Dataset Generalizability

This experiment evaluates whether DeepProbLog generalizes beyond the multi-step attack scenarios it is trained on. In particular, we investigate whether incorporating symbolic knowledge enables the model to learn representations that transfer across attack variants or network vantage points, rather than overfitting to dataset-specific patterns. We focus on the DPL model with both temporal and attack-phase

knowledge.

We consider two cross-domain evaluation settings. For AIT-LDS V2, the model is trained on one multi-step attack scenario (Santos) and evaluated on a different scenario (Fox), allowing us to assess generalization across variations of the same MSA setting. For DARPA 2000, the model is trained on the Inside dataset and evaluated on the DMZ dataset, enabling evaluation of generalization across different network vantage points.

In both settings, models are trained on the full source scenario (Santos or Inside) and evaluated on the complete target scenario (Fox or DMZ). We compare both end-to-end training and a pretrained configuration for the neural component, and benchmark against a multiclass neural baseline.

Due to computational constraints, a window size of 10 was used for AIT-LDS V2 in this experiment instead of the default value of 100. This introduces a reduced temporal context compared to previous experiments. As a result, absolute performance levels should be interpreted with caution, and comparisons should primarily focus on relative differences between models within this setting.

Generalization Across Scenarios Figure 7.5 presents the results on AIT-LDS V2. The neural baseline exhibits limited generalization across attack scenarios, whereas incorporating temporal and attack-phase knowledge leads to substantial improvements across all evaluation metrics. This indicates that symbolic constraints introduce useful inductive biases that transfer across attack variants. The end-to-end DPL model achieves the strongest overall performance, suggesting that learned representations benefit from direct interaction with symbolic constraints during training.

A phase-level breakdown is provided in Table 7.5. All models struggle to detect attack phase 4 when trained on Santos and evaluated on Fox, indicating that this phase exhibits substantial variation across scenarios. Nevertheless, the DPL models consistently outperform both the neural and symbolic baselines across most attack phases.

Generalization Across Network Vantage Points Figure 7.6 presents the results of the cross-vantage point experiment. Similar to the cross-scenario setting, the neural baseline shows limited generalization capability, whereas the DPL models achieve substantially higher macro F1 and weighted F1-scores.

The highest macro F1-score is obtained by the symbolic baseline. However, this comes at the cost of a high false alarm rate, as reflected in the poor F1-score for the benign class (see Table 7.6). In contrast, the DPL models achieve a more balanced performance profile, while maintaining competitive overall performance.

Table 7.6 reports the per-class results. The DPL models show the greatest difficulty in correctly identifying phase 1, whereas the neural baseline struggles across multiple classes, indicating less stable performance across attack stages.

Takeaways

Generalization. Symbolic temporal and attack-phase knowledge improves generalization across both attack scenarios and network vantage points. The strongest results are consistently obtained when symbolic constraints are integrated jointly with neural learning, indicating that interaction during training is beneficial for transferability across attack variants.

7.3.5 Missed Attack Steps

All previous experiments assume perfect knowledge of previously observed attack phases. In practice, however, intrusion detection systems may fail to detect certain attack steps, resulting in incomplete attack histories.

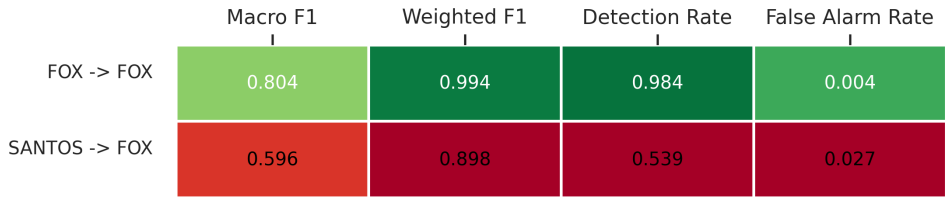
In this experiment, we investigate whether the temporal constraints can be revised to tolerate such missing observations. More specifically, we evaluate whether the model can still infer $phase_i$ even when the preceding phase $phase_{i-1}$ has not been observed.

Table 7.5: Per-phase generalizability performance on AIT-LDS V2. The models are trained on Santos and tested on Fox. The best performing score per metric is in bold.

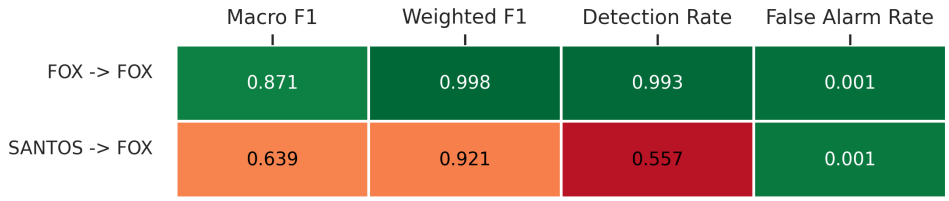
Model	Macro F1	Weighted F1	$F1_{Benign}$	$F1_{P1}$	$F1_{P2}$	$F1_{P3}$	$F1_{P4}$
DPL_{E2E}	**0.721**	**0.945**	**0.997**	0.977	**0.752**	**0.877**	0.026
$DPL_{Pretrained}$	0.639	0.921	0.962	**0.981**	0.620	0.585	**0.046**
$LSTM_{Multiclass}$	0.596	0.898	**0.997**	0.965	0.534	0.511	0.024
$Logic_{Baseline}$	0.561	0.953	0.982	0.000	0.999	0.824	0.000

Table 7.6: Per-phase generalizability performance on DARPA 2000. The models are trained on INSIDE and tested on DMZ. The best performing score per metric is in bold.

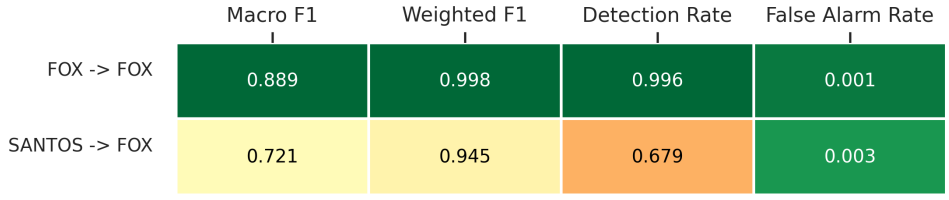
Model	Macro F1	Weighted F1	$F1_{Benign}$	$F1_{P1}$	$F1_{P2}$	$F1_{P3}$	$F1_{P4}$	$F1_{P5}$
$Logic_{Baseline}$	**0.931**	0.871	0.686	**0.998**	**1.000**	**1.000**	**0.974**	0.926
DPL_{E2E}	0.912	**0.974**	0.951	0.586	1.000	1.000	0.947	0.990
$DPL_{Pretrained}$	0.886	0.971	**0.947**	0.431	**1.000**	0.974	0.973	**0.991**
$LSTM_{Multiclass}$	0.410	0.104	0.380	0.468	0.367	0.904	0.333	0.007



(a) Multiclass LSTM

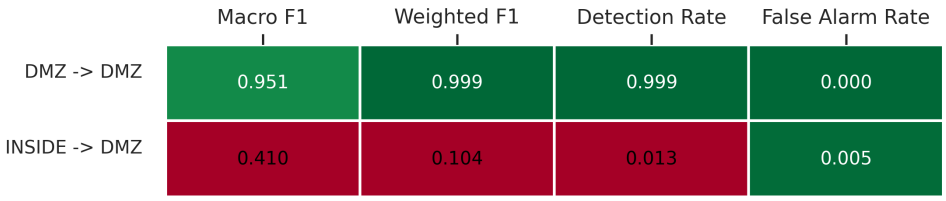


(b) DeepProbLog Pretrained

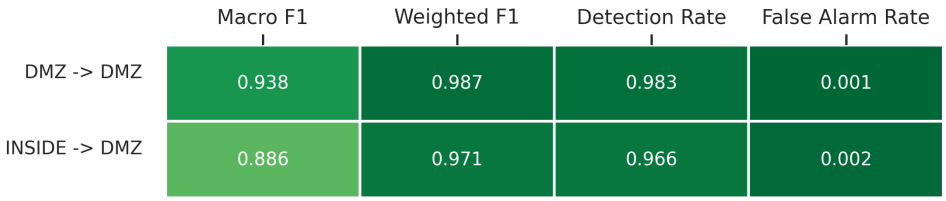


(c) DeepProbLog End-to-End

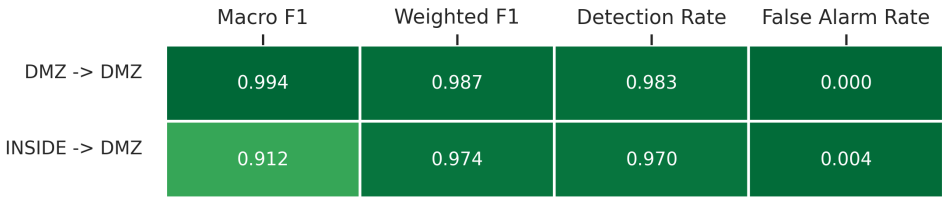
Figure 7.5: Confusion matrices illustrating cross-dataset generalization on AIT-LDS V2. Each matrix compares model performance when trained and tested on the same scenario with performance when evaluated on a different dataset.



(a) Neural Baseline (Multiclass LSTM)



(b) DeepProbLog Pretrained



(c) DeepProbLog End-to-End

Figure 7.6: Confusion matrices illustrating cross-dataset generalization on DARPA 2000. Each matrix compares model performance when trained and tested on the same scenario with performance when evaluated on a different dataset.

To demonstrate this flexibility, we perform an experiment on AIT-LDS V2 where we relax the temporal constraint for $phase_2$. Concretely, the model is allowed to predict the second attack phase both when the first phase has been observed and when it has not.

$$plausible_attack_phase(F_1, F_2, F_3, phase_2) : -F_2 = 0, F_3 = 0.$$

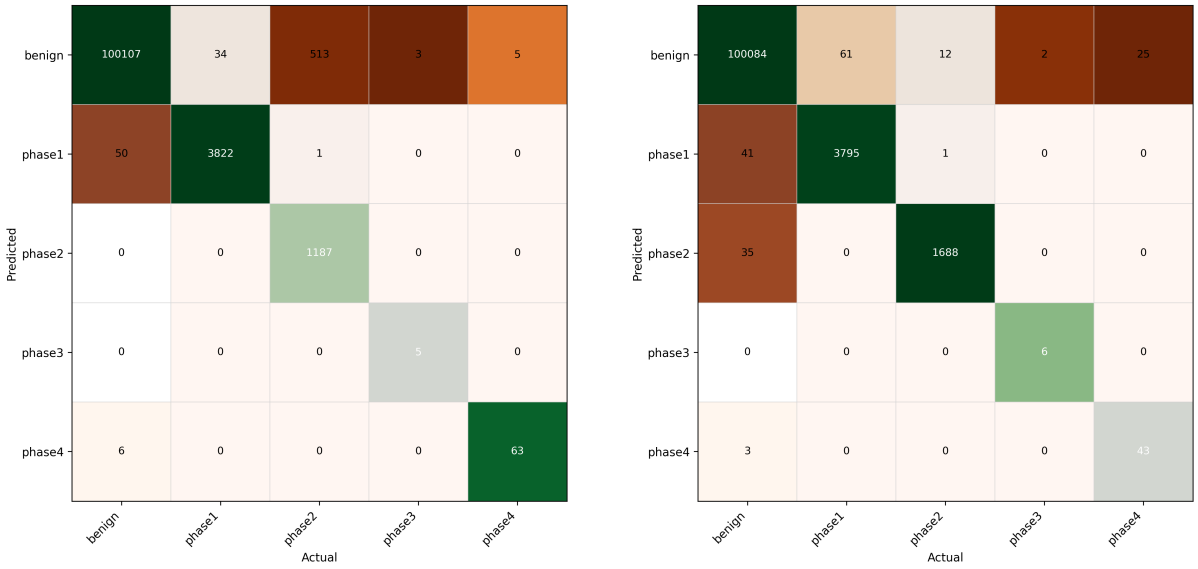
To simulate missed detections, we introduce noise by randomly resetting all attack-phase flags to 0 for 30% of the true $phase_2$ samples in both the training and test sets. This removes the temporal evidence that would normally permit the prediction of $phase_2$.

Robustness to Missing Observations. Table 7.7 summarizes the overall and per-phase performance. Compared to the original setting, the revised logic evaluated on the corrupted dataset exhibits a modest reduction in overall performance. Nevertheless, is substantially outperforms the original logic when applied to the same corrupted data, improving both the macro F1-score and the F1-score of $phase_2$.

The confusion matrices provide further insight into this behavior. As shown in Figure 7.7(b), the revised logic misses only 13 $phase_2$ flows despite the corrupted attack phase flags. In contrast, the original logic fails to recover these missing phases, resulting in 514 misclassified $phase_2$ flows (Figure 7.7(a)).

Takeaways

Robustness to missing observations. Relaxing temporal constraints improves robustness to missing attack observations, while preserving overall detection performance.



(a) Corrupted dataset with the original logic.

(b) Corrupted dataset with the revised temporal logic.

Figure 7.7: Confusion matrices for the end-to-end DPL model on AIT-LDS V2. Subfigure (a) shows the effect of removing attack phases when using the original logic. Subfigure (b) shows performance on the corrupted dataset after introducing the revised logic. The comparison illustrates the robustness of the proposed logic to missing attack phases.

Table 7.7: Performance under missing attack phases in AIT-LDS V2. The best-performing value for each metric is highlighted in bold.

Configuration	Macro F1	Weighted F1	$F1_{Benign}$	$F1_{P1}$	$F1_{P2}$	$F1_{P3}$	$F1_{P4}$
$Logic_{Orig}, Data_{Orig}$	**0.931**	**0.999**	**0.999**	0.988	**1.000**	0.769	0.900
$Logic_{Orig}, Data_{Corr}$	0.899	0.994	0.997	**0.989**	0.822	0.769	**0.920**
$Logic_{Rev}, Data_{Corr}$	0.917	0.998	**0.999**	0.987	0.986	**0.857**	0.754

7.3.6 System-Level Metrics

To evaluate the practical feasibility of the proposed system, we measure average inference latency per prediction, throughput, and peak memory usage. Inference time corresponds to the full execution of a DeepProbLog query, including both logical grounding and neural forward passes when required. System metrics were collected using `psutil` [32] during training and inference.

AIT-LDS V2. The results in Table 7.8 show that mean inference latency ranges from 0.14 ms to 2.48 ms across models, corresponding to throughput values between approximately 400 and 7,000 predictions per second. The symbolic baselines achieve the lowest latency, followed by the neuro-symbolic models with temporal and attack-phase knowledge. These DPL models achieve substantially lower average latency compared to their temporal-only counterparts.

Peak memory usage remains relatively stable across all configurations, ranging from approximately 25.9 GB to 30.2 GB. The small variation across models further suggests that memory consumption is largely driven by the shared execution infrastructure rather than the specific reasoning configuration.

DARPA 2000. As shown in Table 7.9, mean inference latency ranges from 2.19 ms to 5.60 ms. As in the AIT-LDS V2 setting, the symbolic baselines are the fastest, while neuro-symbolic models incur additional computational overhead due to neural execution. Overall, DARPA 2000 exhibits higher inference latency and a smaller performance gap between models compared to AIT-LDS V2.

Table 7.8: Runtime efficiency on the AIT-LDS V2 test set. Latency is averaged per prediction across several runs, and throughput is computed as $1/\text{latency}$. The best-performing value for each metric is highlighted in bold.

Model	Latency (ms)	Throughput (pred/s)	Peak Memory (GB)
$DPL_{Pretrained}$ T	2.48 ± 40.28	403	25.90 ± 0.57
DPL_{E2E} T	2.41 ± 36.93	415	29.83 ± 0.56
$DPL_{Pretrained}$ T+R	0.46 ± 3.32	2155	30.19 ± 0.01
DPL_{E2E} T+R	0.46 ± 1.09	2152	30.19 ± 0.00
$Logic_{Baseline}$ T+R	0.23 ± 15.03	4354	30.13 ± 0.01
$Logic_{Baseline}$ T	$\mathbf{0.14 \pm 0.35}$	**6985**	29.95 ± 0.03

Table 7.9: Runtime efficiency on the DARPA 2000 test set. Latency is averaged per prediction across several runs, and throughput is computed as $1/\text{latency}$. The best-performing value for each metric is highlighted in bold.

Model	Latency (ms)	Throughput (pred/s)	Peak Memory (GB)
$DPL_{Pretrained}$ T	5.28 ± 36.53	189	15.32 ± 0.02
DPL_{E2E} T	5.34 ± 34.69	187	15.31 ± 0.00
$DPL_{Pretrained}$ T+R	5.60 ± 38.83	178	15.19 ± 0.00
DPL_{E2E} T+R	5.29 ± 40.86	189	15.20 ± 0.01
$Logic_{Baseline}$ T+R	3.06 ± 31.49	327	12.65 ± 0.64
$Logic_{Baseline}$ T	$\mathbf{2.19 \pm 22.00}$	**457**	12.99 ± 0.01

Peak memory usage is again stable across models, ranging from approximately 12.6 GB to 15.3 GB, with only minor differences between configurations. This suggests that memory consumption scales primarily with dataset size rather than the inclusion of symbolic reasoning components.

Takeaways

Inference latency and memory consumption. The proposed neuro-symbolic approach achieves low inference latency and stable memory consumption.

Chapter 8

Discussion

In this chapter, we interpret the experimental results presented in Chapter 7 and place them in a broader context. Rather than reporting additional empirical findings, we focus on explaining observed patterns, comparing results across experiments, and relating them to existing work. We also discuss the limitations of the proposed framework and identify directions for future research.

8.1 The Role of Symbolic Knowledge

A central objective of this thesis was to investigate whether symbolic knowledge can improve multi-step attack detection. Across most experiments, the results indicate that incorporating symbolic knowledge is beneficial. The observed improvements suggest that symbolic constraints not only enhance predictive performance but also guide the model toward predictions that are more consistent with the temporal structure of multi-step attacks.

In addition to improving detection performance, symbolic knowledge appears to improve the model's ability to generalize across attack scenarios and network environments. This observation is consistent with the findings of [11], who report that symbolic constraints can improve both generalization and the detection of attack behaviors that are difficult to identify from throughput-based features alone. Together, these results suggest that symbolic knowledge provides useful inductive biases that remain beneficial across a range of learning settings.

8.1.1 Temporal Constraints vs. Attack-Phase Rules

The experiments indicate that temporal constraints and attack-phase rules play complementary roles in the detection process. Temporal constraints enforce valid transitions between attack stages, ensuring consistency in the modeled attack progression by restricting which steps can follow from previously observed events.

Attack-phase knowledge, in contrast, operates at the level of individual flows by filtering events based on domain-specific characteristics. This reduces the set of candidate observations that must be considered by the neural component, which otherwise struggles to reliably distinguish fine-grained attack steps from raw network data alone.

The strongest performance improvements are observed when both forms of knowledge are combined. This suggests that temporal constraints provide guidance over attack progression, while attack-phase rules emphasize characteristics associated with specific attack steps. Together, they introduce complementary inductive biases that improve both detection performance and consistency across attack stages.

8.1.2 Symbolic Reasoning and Neural Learning

The results suggest that the relative importance of symbolic reasoning and neural learning depends on the characteristics of the dataset and the extent to which attack behavior can be described through domain

knowledge.

On DARPA 2000, the symbolic baseline achieves competitive macro F1-scores, indicating that a substantial portion of the attack behavior can be represented explicitly through symbolic constraints. In contrast, the symbolic baseline performs considerably worse on AIT-LDS V2, where attack phases are more behavior-driven and cannot be captured reliably using handcrafted rules alone. In this setting, the neural component becomes essential for learning patterns that are difficult to express symbolically.

These findings suggest that symbolic reasoning and neural learning should be viewed as complementary rather than competing approaches. Symbolic knowledge is particularly effective when attack behavior can be described precisely, whereas neural learning becomes increasingly important when attack-steps are noisy, variable, or only partially observable.

Nevertheless, across both datasets, the combination of neural learning and symbolic reasoning consistently achieves the strongest performance, indicating that each component compensates for limitations of the other.

8.2 End-to-End Learning vs. Pretraining

An important question is whether symbolic knowledge should be incorporated only during inference or whether it should also influence representation learning.

The strong performance of the pretrained DPL models compared to the neural baselines demonstrates that symbolic rules can be applied effectively after neural training. Even without directly influencing representation learning, symbolic reasoning improves prediction quality by filtering outputs that violate expert knowledge.

However, the results also indicate advantages of end-to-end optimization. While the pretrained and end-to-end models achieve comparable performance on AIT-LDS V2, the end-to-end models consistently outperform their pretrained counterparts in the cross-dataset experiments. This suggests that incorporating symbolic constraints during training encourages the neural component to learn representations that better reflect the structure of multi-step attacks.

Overall, these findings indicate that symbolic reasoning is beneficial both during training and during inference, with the largest gains occurring when both components are optimized jointly.

8.3 Data Efficiency

The experiments show a clear relationship between model performance and the amount of available training data. In low-data settings, the DPL models incorporating temporal and attack-phase knowledge consistently outperform all other approaches in terms of macro F1-score. This suggests that symbolic constraints provide a strong inductive bias when only limited training data is available.

As the number of training samples increases, the performance gap between the temporal and attack-phase DPL model and the temporal-only variant gradually decreases. On AIT-LDS V2, the temporal-only model approaches comparable performance after approximately 10,000 training samples. In contrast, the neural baselines do not reach similar performance within the evaluated range, indicating that symbolic structure continues to provide benefits even as additional data becomes available.

Macro F1-scores remain relatively low during the earliest training stages for all models on AIT-LDS V2, suggesting that a minimum amount of data is required before stable performance can be achieved. This effect is less pronounced on DARPA 2000, where strong performance is obtained with fewer samples, likely because the symbolic component captures a larger proportion of the relevant attack behavior.

An additional observation is a temporary decrease in performance on AIT-LDS V2 after approximately 30 training samples for all models except the DPL model incorporating both temporal and attack-phase knowledge. One possible explanation is that early updates of the neural component introduce unstable representations before sufficient data has been observed. The symbolic constraints may partially compensate for this effect by providing additional structure during learning. However, further investigation is required to confirm this hypothesis.

Overall, these results suggest that symbolic constraints act as a stabilizing inductive bias, particularly during the early stages of training when data is scarce.

8.4 Temporal Context

Unlike previous approaches that use LSTMs to model complete attack sequences directly [31, 30], the proposed framework separates attack-step classification from attack progression reasoning. The neural component focuses on classifying individual attack steps, while symbolic rules model long-range dependencies between those steps.

As a result, the window size only determines how much local context is available for classifying the current flow. The results show that the optimal window size differs between datasets. For AIT-LDS V2, the best performance is obtained with a window size of 100, whereas DARPA 2000 performs best with a window size of 10. This suggests that different attack behaviors require different amounts of local temporal context. Behavior-driven activities such as scanning and exfiltration appear to benefit from larger windows, while more localized attack patterns can be detected effectively using shorter contexts.

This separation of responsibilities represents a practical advantage of the proposed architecture. Long-range attack progression can be modeled explicitly through symbolic reasoning, while the neural component can focus on learning local attack characteristics from data. Consequently, the framework avoids relying entirely on neural architectures to learn long-range temporal dependencies implicitly.

8.5 Computational Performance

An important consideration for multi-step attack detection is whether the additional symbolic reasoning introduces substantial computational overhead.

The runtime experiments show that the proposed neuro-symbolic models maintain low inference latency and stable memory consumption across both datasets. Although the symbolic baselines are consistently faster, the overhead introduced by neural execution remains modest.

An interesting observation is that models incorporating attack-phase knowledge achieve lower latency than temporal-only variants on AIT-LDS V2. Temporal-only models evaluate the neural component for every candidate flow, whereas attack-phase rules filter out many flows before neural execution is required. As a result, fewer neural evaluations are performed during inference.

This finding highlights a practical advantage of symbolic reasoning. Rather than introducing only additional computation, symbolic constraints can also reduce the amount of neural evaluations that is required. Consequently, the symbolic component not only improves detection performance but can also contribute to computational efficiency.

8.6 Limitations

As with any scientific study, this work has several limitations, stemming both from the assumptions underlying our multi-step attack model and from the system design itself.

8.6.1 Adversarial Manipulation

Our approach assumes that attacks exhibit patterns that can be captured through expert-defined knowledge and temporal correlations. Consequently, the system may be less effective against adversaries who deliberately deviate from known attack sequences or avoid the temporal and relational patterns on which the correlation mechanism relies. Such limitations are inherent to correlation-based approaches, and mitigation strategies have been studied in the literature [26]. While we acknowledge these limitations, addressing them falls outside the scope of this thesis.

Another limitation concerns adversarial manipulation of the machine learning components. While attackers may attempt to evade or poison the detection model, Sommer et al. [38] argue that such attacks are often more difficult to execute in practice than commonly assumed, as they require substantial knowledge of the target system and considerable effort. Moreover, adversarial machine learning constitutes a research field of its own. For this reason, defending against such attacks falls outside the scope of this thesis.

Nevertheless, we hypothesize that the inclusion of symbolic knowledge may increase the difficulty of manipulating the system compared to purely data-driven approaches, as its reasoning is not solely derived from learned patterns. This intuition is consistent with recent work suggesting that knowledge-based components can constrain the space of feasible adversarial manipulations [11]. However, evaluating the resilience of neuro-symbolic systems against adversarial attacks remains an open research question and is beyond the scope of this thesis.

8.6.2 ”Perfect” Attack History

A key limitation of this study is that it assumes a ”perfect” attack history. In a realistic deployment, the attack history would be constructed incrementally from the model's own predictions. However, in our experiments is instead updated using the ground-truth attack phases associated with each flow.

As a consequence, the attack history never contains incorrectly inferred attack steps and never misses attack steps that should have been recorded. This eliminates the possibility of error propagation, where false positives could introduce spurious entries into the history and false negatives could prevent relevant attack steps from being added. The resulting attack history is therefore more accurate than what would be available in practice.

To partially address this limitation, we evaluated the impact of missing information through the missing-flags experiments. However, these experiments only model the effect of omitted attack steps and do not capture the consequences of incorrect attack history updates caused by false detections. In particular, we cannot assess how the system would behave if a large number of false positives accumulated over time and influenced subsequent reasoning.

Nevertheless, we expect the impact of individual prediction errors to be mitigated by the multi-step nature of the attack model. Since an attack scenario is only recognized after several attack phases have been observed, isolated misclassifications are less likely to result in a complete attack scenario being inferred. Evaluating this assumption in a fully online setting remains an important direction for future work.

8.6.3 Evaluation in an Experimental Setting

The results of this study should not be interpreted as evidence that the proposed approach is ready for deployment in an operational environment. Rather, the evaluation was conducted in an experimental setting that abstracts away several practical challenges associated with real-time deployment.

In particular, the current implementation relies on DeepProbLog's batch-oriented query generation, whereas a deployed system would need to process flows incrementally and update its attack history online. Supporting such behavior would require modifications to the query generation process and additional mechanisms for managing uncertainty and missing information during inference.

Furthermore, while our experiments indicate low inference latency, we have not evaluated the complete system under realistic deployment conditions. Factors such as online query generation, larger rule sets, sustained traffic volumes, and long-term maintenance of attack histories may increase computational overhead. Consequently, although the results suggest that the approach has the potential to support real-time detection, further engineering effort and empirical evaluation are required before conclusions about operational feasibility can be drawn.

8.7 Future Work

While the proposed neuro-symbolic approach demonstrates strong performance across multiple evaluation settings, several directions remain for future research.

8.7.1 Enhanced Attack Correlation

The current correlation mechanism combines prerequisite–consequence relationships with similarity-based matching between alerts. Although effective in the evaluated scenarios, it does not explicitly model the structure of the underlying environment. Future work should investigate richer representations of system context, such as network topology, host dependencies, and service-vulnerability relationships. Incorporating such information may improve the reconstruction of attack traces and provide additional context for downstream reasoning.

8.7.2 Alternative Neural Architectures

The proposed approach relies on a single neural network to model all attack phases and attack variants. An interesting direction is the use ensemble-based architectures, where specialized neural components are trained for specific attack stages, attack classes, or environments. Such architectures may allow the system to adapt more effectively to diverse attack behavior.

8.7.3 Neuro-Symbolic Anomaly Detection

This thesis focuses on supervised detection of known multi-step attacks. However, the proposed framework is not limited to classification-based models. Future work could investigate replacing the neural classifier with an anomaly detection component capable of identifying previously unseen attacks. Symbolic knowledge could then be incorporated either during learning or post-hoc to improve detection quality, temporal consistency, and explainability.

8.7.4 Probabilistic Reasoning Under Uncertainty

Although the proposed system is implemented in DeepProbLog, the symbolic component primarily acts as a deterministic constraint mechanism. As a result, the framework does not fully exploit the probabilistic reasoning capabilities offered by DeepProbLog. Future research should explore learnable probabilistic rules and explicit reasoning under uncertainty, enabling the system to handle situations where multiple attack explanations or attack progressions are simultaneously plausible.

Chapter 9

Conclusions

Multi-step attacks are challenging to detect due to their temporal structure and the dependencies between individual attack steps. As attacks unfold over time, effective detection requires not only identifying isolated malicious events, but also reasoning about how these events form a coherent progression toward an adversary's objective. Detecting such attacks during their early stages is particularly important, as it enables defenders to intervene before an attack reaches its final objective.

To address this challenge, this thesis introduced a neuro-symbolic framework for multi-step attack detection based on DeepProbLog [20, 21]. The proposed approach combines neural attack-step classification with symbolic temporal reasoning, integrating knowledge of attack progression directly into the learning and inference process. By enforcing temporal constraints, the system aims to improve both attack-step detection and the consistency of predicted attack progressions.

The system was evaluated on two multi-step attack datasets: DARPA 2000 [18] and AIT-LDS V2 [14, 15]. Across both datasets, the neuro-symbolic models consistently achieved stronger overall performance than neural and symbolic baselines. In particular, incorporating temporal and attack-phase knowledge improved macro F1-score and reduced false alarm rates. Kitsune [22] outperformed the neuro-symbolic approach on DARPA 2000, while on the more complex AIT-LDS V2 the proposed method performed considerably better, achieving a macro F1-score of 0.944 compared to 0.645. The experiments further demonstrated that symbolic constraints improve temporal consistency by ensuring that predictions follow valid attack progressions.

Beyond predictive performance, the results indicate that symbolic knowledge improves data efficiency. The DeepProbLog models required fewer training samples to achieve strong performance compared to purely neural baselines, suggesting that symbolic constraints provide useful inductive biases during learning. Furthermore, the runtime analysis shows that the proposed approach achieves low inference latency and stable memory consumption despite the additional reasoning component.

Finally, the cross-dataset experiments indicate that the proposed approach generalizes beyond the specific attack scenarios on which it was trained. Although performance decreases under distribution shift, the neuro-symbolic models consistently outperform neural baselines, suggesting that the learned representations capture transferable characteristics of multi-step attacks.

Overall, the results suggest that combining neural learning with symbolic temporal and attack-phase knowledge provides an effective framework for multi-step attack detection. By integrating attack progression knowledge directly into the detection process, the proposed approach improves predictive performance, temporal consistency, data efficiency, and generalization while maintaining low inference latency. Thereby, supporting the timely detection of multi-step attacks.

AI Usage Statement

During the preparation of this work, I used ChatGPT, Claude, Copilot, and NotebookLM to refine grammar and paragraph flow, create tables and plots, and assist in the literature review process. After using

these tools, I reviewed, edited, and made the content my own and validated the outcome as needed, and I take full responsibility for the content of the thesis report.

Bibliography

[1] Marco Angelini, Silvia Bonomi, Emanuele Borzi, Antonella Del Pozzo, Simone Lenti, and Giuseppe Santucci. An attack graph-based on-line multi-step attack detector. pages 1–10, 01 2018. doi: 10.1145/3154273.3154311.

[2] 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. doi: 10.1109/ICCCN61486.2024.10637618.

[3] Alice Bizzarri, Chung-En Yu, Brian Jalaian, Fabrizio Riguzzi, and Nathaniel D. Bastian. A synergistic approach in network intrusion detection by neurosymbolic ai, 2024. URL https://arxiv.org/abs/2406.00938.

[4] Bing Chen, J. Lee, and A.S. Wu. Active event correlation in bro ids to detect multi-stage attacks. In *Fourth IEEE International Workshop on Information Assurance (IWIA'06)*, pages 16 pp.–50, 2006. doi: 10.1109/IWIA.2006.2.

[5] Nadjah Chergui and Narhimene Boustia. Contextual-based approach to reduce false positives. *IET Information Security*, 14, 01 2020. doi: 10.1049/iet-ifs.2018.5479.

[6] Cloudflare. What is stuxnet?, 2026. URL https://www.trellix.com/security-awareness/ransomware/what-is-stuxnet/. Accessed: 2026-06-15.

[7] 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.

[8] Emerging Threats. Emerging threats, n.d. URL https://community.emergingthreats.net/.

[9] FOX IT. Declassified technical report. Technical Report REP-Armstrong-221856, Eindhoven University of Technology, 2025. URL https://assets.w3.tue.nl/w/fileadmin/content/pers/2025/05%20Mei/REP_Armstrong_221856_TechnicalReport_TLP-CLEAR_v2.0_DECLASSIFIED.pdf. Accessed: 2026-06-15.

[10] Ibrahim Ghafir, Mohammad Hammoudeh, Vaclav Prenosil, Liangxiu Han, Robert Hegarty, Khaled Rabie, and Francisco J. Aparicio-Navarro. Detection of advanced persistent threat using machine-learning correlation analysis. *Future Generation Computer Systems*, 89:349–359, 2018. ISSN 0167-739X. doi: https://doi.org/10.1016/j.future.2018.06.055. URL https://www.sciencedirect.com/science/article/pii/S0167739X18307532.

[11] Xin Fan Guo, Albert Merono Penuela, Sergio Maffeis, and Fabio Pierazzi. Knowml: Improving generalization of ml-nids with attack knowledge graphs, 2025. URL https://arxiv.org/abs/2506.19802.

[12] 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, 2026. URL https://arxiv.org/abs/2509.06921.

[13] jumpcloud. What is a network flow?, 2025. URL https://jumpcloud.com/it-index/what-is-a-network-flow.

[14] M. Landauer, F. Skopik, M. Wurzenberger, W. Hotwagner, and A. Rauber. Have it your way: Generating customized log datasets with a model-driven simulation testbed. *IEEE Transactions on Reliability*, 70(1):402–415, 2021. doi: 10.1109/TR.2020.3031317.

[15] M. Landauer, F. Skopik, M. Frank, W. Hotwagner, M. Wurzenberger, and A. Rauber. Maintainable log datasets for evaluation of intrusion detection systems. *IEEE Transactions on Dependable and Secure Computing*, 20(4):3466–3482, 2022. doi: 10.1109/TDSC.2022.3201582.

[16] Max Landauer, Florian Skopik, Maximilian Frank, Wolfgang Hotwagner, Markus Wurzenberger, and Andreas Rauber. Ait log data set v2.0, 2022. URL https://zenodo.org/records/5789064. Accessed: 2026-06-15.

[17] Diana Levshun and Igor Kotenko. A survey on artificial intelligence techniques for security event correlation: models, challenges, and opportunities. *Artificial Intelligence Review*, 56, 01 2023. doi: 10.1007/s10462-022-10381-4.

[18] Lincoln Laboratory. 2000 darpa intrusion detection scenario specific datasets, n.d. URL https://www.ll.mit.edu/r-d/datasets/2000-darpa-intrusion-detection-scenario-specific-datasets.

[19] Lockheed Martin. Cyber kill chain, n.d. URL https://www.lockheedmartin.com/en-us/capabilities/cyber/cyber-kill-chain.html. Accessed: 2026-06-15.

[20] Robin Manhaeve, Sebastijan Dumancic, Angelika Kimmig, Thomas Demeester, and Luc De Raedt. Deepproblog: Neural probabilistic logic programming. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper_files/paper/2018/file/dc5d637ed5e62c36ecb73b654b05ba2a-Paper.pdf.

[21] 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. doi: https://doi.org/10.1016/j.artint.2021.103504. URL https://www.sciencedirect.com/science/article/pii/S0004370221000552.

[22] Yisroel Mirsky, Tomer Doitshman, Yuval Elovici, and Asaf Shabtai. Kitsune: An ensemble of autoencoders for online network intrusion detection, 2018. URL https://arxiv.org/abs/1802.09089.

[23] MITRE ATT&CK. Enterprise matrix, n.d.. URL https://attack.mitre.org/matrices/enterprise/. Accessed: 2026-06-15.

[24] MITRE ATT&CK. Reconnaissance, n.d.. URL https://attack.mitre.org/tactics/TA0043/. Accessed: 2026-06-15.

[25] Julio Navarro, Aline Deruyver, and Pierre Parrend. A systematic survey on multi-step attack detection. *Computers & Security*, 76:214–249, 2018. ISSN 0167-4048. doi: https://doi.org/10.1016/j.cose.2018.03.001. URL https://www.sciencedirect.com/science/article/pii/S0167404818302141.

[26] Peng Ning and Dingbang Xu. Hypothesizing and reasoning about attacks missed by intrusion detection systems. *ACM Trans. Inf. Syst. Secur.*, 7(4):591–627, November 2004. ISSN 1094-9224. doi: 10.1145/1042031.1042036. URL https://doi.org/10.1145/1042031.1042036.

[27] Peng Ning, Yun Cui, and Douglas S. Reeves. Constructing attack scenarios through correlation of intrusion alerts. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, CCS '02, page 245–254, New York, NY, USA, 2002. Association for Computing Machinery. ISBN 1581136129. doi: 10.1145/586110.586144. URL https://doi.org/10.1145/586110.586144.

[28] NIST. Cve-1999-0977 detail, n.d.. URL https://nvd.nist.gov/vuln/detail/cve-1999-0977.

[29] NIST. Cyber attack, n.d.. URL https://csrc.nist.gov/glossary/term/cyber_attack.

[30] Trung V. Phan and Thomas Bauschert. Learning the apt kill chain: Temporal reasoning over provenance data for attack stage estimation, 2026. URL https://arxiv.org/abs/2603.07560.

[31] Jian Qu, Yuchen Zhang, Jialong Zhang, Jianfeng Li, and Xiaobo Ma. Tracegram: Framing trace-level traffic analysis with temporally-aware multiple instance learning. In *35th USENIX Security Symposium (USENIX Security 26)*. USENIX Association, 2026. URL https://www.usenix.org/conference/usenixsecurity26/presentation/qu.

[32] Giampaolo Rodola. psutil: Cross-platform lib for process and system monitoring, 2009. URL https://github.com/giampaolo/psutil.

[33] 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. doi: https://doi.org/10.1016/j.bdr.2022.100359. URL https://www.sciencedirect.com/science/article/pii/S2214579622000533.

[34] Syed Usman Shaukat, Saad Khan, and Simon Parkinson. A review on multi-step attack detection. *IEEE Access*, 13:161779–161805, 2025. doi: 10.1109/ACCESS.2025.3607497.

[35] Snort. Snort, n.d.. URL https://www.snort.org/.

[36] Snort. The basics, n.d.. URL https://docs.snort.org/rules/.

[37] Snort. Downloads, n.d.. URL https://www.snort.org/downloads/#rule-downloads.

[38] Robin Sommer and Vern Paxson. Outside the closed world: On using machine learning for network intrusion detection. *Proceedings - IEEE Symposium on Security and Privacy*, pages 305–316, 01 2010. doi: 10.1109/SP.2010.25.

[39] Suricata. Observe. protect. adapt., n.d. URL https://suricata.io/.

[40] Trellix. What is stuxnet?, n.d. URL https://www.trellix.com/security-awareness/ransomware/what-is-stuxnet/. Accessed: 2026-06-15.

[41] U.S. House Committee on Oversight and Government Reform. The equifax data breach. Technical report, U.S. House of Representatives, December 2018. URL https://oversight.house.gov/wp-content/uploads/2018/12/Equifax-Report.pdf.

[42] WIKIPEDIA. List of ip protocol numbers, 2025. URL https://en.wikipedia.org/wiki/List_of_IP_protocol_numbers.

[43] Dongran Yu, Bo Yang, Dayou Liu, Hui Wang, and Shirui Pan. A survey on neural-symbolic learning systems, 2023. URL https://arxiv.org/abs/2111.08164.

[44] Zeek Project. Zeek network security monitor, n.d. URL https://zeek.org/.

Appendix A

Glossary

In this appendix we give an overview of frequently used terms and abbreviations.

MSA: Multi-Step Attack

IDS: Intrusion Detection System

NIDS: Network Intrusion Detection System

DPL: DeepProbLog

Neuro-Symbolic AI: Neuro-Symbolic Artificial Intelligence

AIT-LDS V2: Austrian Institute of Technology Log Data Set Version 2

Appendix B

Extended Experimental Results

B.1 Ablation Study

Table B.1: Experimental configurations evaluated in neural baseline models.

Configuration	**Evaluated Options**
Feature Sets	Base features, Flow-only features
Window Sizes	10, 100
Learning Rates	10^{-3}, 10^{-4}

Table B.2: Detection performance of neural baseline configurations on AIT-LDS V2 sorted by macro F1-score. The best score per column is in bold.

Model	Macro F1	Accuracy	DR	FAR
$LSTM_{\text{Ensemble}}(\text{base}, w = 100, lr = 10^{-3})$	**0.8650**	**0.9966**	**0.9874**	**0.0029**
$LSTM_{\text{Multiclass}}(\text{base}, w = 100, lr = 10^{-3})$	0.8213	0.9952	0.9865	0.0043
$LSTM_{\text{Ensemble}}(\text{base}, w = 100, lr = 10^{-4})$	0.7498	0.9907	0.9867	0.0090
$LSTM_{\text{Multiclass}}(\text{base}, w = 100, lr = 10^{-4})$	0.6919	0.9874	0.9846	0.0124
$LSTM_{\text{Multiclass}}(\text{base}, w = 10, lr = 10^{-3})$	0.6749	0.9923	0.9756	0.0067
$LSTM_{\text{Ensemble}}(\text{base}, w = 10, lr = 10^{-3})$	0.6741	0.9924	0.9760	0.0066
$LSTM_{\text{Multiclass}}(\text{flow-only}, w = 100, lr = 10^{-3})$	0.6367	0.9307	0.9706	0.0715
$LSTM_{\text{Ensemble}}(\text{base}, w = 10, lr = 10^{-4})$	0.6365	0.9880	0.9742	0.0112
$LSTM_{\text{Multiclass}}(\text{base}, w = 10, lr = 10^{-4})$	0.6227	0.9859	0.9778	0.0136
$LSTM_{\text{Ensemble}}(\text{flow-only}, w = 100, lr = 10^{-3})$	0.5973	0.9209	0.9756	0.0822
$LSTM_{\text{Ensemble}}(\text{flow-only}, w = 100, lr = 10^{-4})$	0.5191	0.9083	0.9703	0.0952
$LSTM_{\text{Multiclass}}(\text{flow-only}, w = 100, lr = 10^{-4})$	0.4831	0.8889	0.9807	0.1162
$LSTM_{\text{Ensemble}}(\text{flow-only}, w = 10, lr = 10^{-3})$	0.4300	0.8780	0.9579	0.1263
$LSTM_{\text{Multiclass}}(\text{flow-only}, w = 10, lr = 10^{-3})$	0.4208	0.8659	0.9661	0.1396
$LSTM_{\text{Ensemble}}(\text{flow-only}, w = 10, lr = 10^{-4})$	0.4204	0.8685	0.9653	0.1368
$LSTM_{\text{Multiclass}}(\text{flow-only}, w = 10, lr = 10^{-4})$	0.4153	0.8623	0.9661	0.1434

Table B.3: Detection performance of neural baseline configurations on DARPA 2000 sorted by macro F1-score. The best score per column is highlighted in bold.

Model	Macro F1	Accuracy	DR	FAR
$LSTM_{\text{Ensemble}}(\text{base}, w = 10, lr = 10^{-3})$	**0.9152**	**0.99958**	0.99957	0.00030
$LSTM_{\text{Multiclass}}(\text{base}, w = 10, lr = 10^{-3})$	0.9142	0.99957	0.99953	**0.00027**
$LSTM_{\text{Ensemble}}(\text{flow-only}, w = 10, lr = 10^{-3})$	0.9131	0.99955	**0.99959**	0.00041
$LSTM_{\text{Multiclass}}(\text{flow-only}, w = 10, lr = 10^{-3})$	0.8850	0.99949	0.99955	0.00046
$LSTM_{\text{Ensemble}}(\text{base}, w = 100, lr = 10^{-3})$	0.8843	0.99955	0.99953	0.00033
$LSTM_{\text{Ensemble}}(\text{flow-only}, w = 100, lr = 10^{-3})$	0.8817	0.99953	0.99951	0.00030
$LSTM_{\text{Multiclass}}(\text{flow-only}, w = 100, lr = 10^{-3})$	0.8793	0.99951	0.99953	0.00033
$LSTM_{\text{Ensemble}}(\text{flow-only}, w = 10, lr = 10^{-4})$	0.8783	0.99944	0.99953	0.00055
$LSTM_{\text{Multiclass}}(\text{base}, w = 100, lr = 10^{-3})$	0.8744	0.99950	0.99950	0.00041
$LSTM_{\text{Ensemble}}(\text{base}, w = 100, lr = 10^{-4})$	0.8636	0.99948	0.99953	0.00044
$LSTM_{\text{Ensemble}}(\text{base}, w = 10, lr = 10^{-4})$	0.8627	0.99944	0.99957	0.00052
$LSTM_{\text{Ensemble}}(\text{flow-only}, w = 100, lr = 10^{-4})$	0.8599	0.9994	0.99951	0.00063
$LSTM_{\text{Multiclass}}(\text{flow-only}, w = 100, lr = 10^{-4})$	0.8479	0.99924	0.99957	0.00110
$LSTM_{\text{Multiclass}}(\text{base}, w = 100, lr = 10^{-4})$	0.8428	0.99933	0.99955	0.00079
$LSTM_{\text{Multiclass}}(\text{flow-only}, w = 10, lr = 10^{-4})$	0.8225	0.99932	0.99951	0.00077
$LSTM_{\text{Multiclass}}(\text{base}, w = 10, lr = 10^{-4})$	0.7836	0.99911	0.99953	0.00123