POLITECNICO DI TORINO

Collegio di Ingegneria Informatica, del Cinema e Meccatronica

Master of Science Thesis

Advanced Persistent Threath Identification

Tutor/s: Prof. Marco Mellia

Candidate: Youness Bouchari

July 2025

Abstract

Advanced Persistent Threats (APTs) represent one of the most critical challenges in modern cybersecurity. Their stealthy and evolving nature makes them particularly difficult to detect within the massive volume of system logs generated by enterprise environments. This thesis investigates the use of machine learning for APT detection from log data, comparing shallow classifiers, deep learning approaches, and a tactic-aware ensemble of fine-tuned BERT heads.

The experiments demonstrate that while shallow models can achieve competitive performance under random data splits, they fail to generalize when evaluated chronologically, underscoring their limited ability to adapt to the evolving behaviors characteristic of APT campaigns. Deep learning models, especially fine-tuned BERT, provide stronger and more stable performance, benefiting from their ability to capture contextual relationships within logs.

The proposed ensemble of tactic-specific BERT heads highlights the potential of specialization by aligning detection capabilities with MITRE ATT&CK tactics. This ensemble achieved promising results under random splits and showed the value of tactic-aware learning, though limitations remain in terms of recall and robustness under chronological evaluation. Error analysis revealed that many missed malicious logs were dominated by obfuscated or semantically weak tokens, making them difficult to distinguish from benign activity.

Overall, this thesis contributes to the understanding of APT detection through log analysis, illustrating both the strengths and limitations of current machine learning approaches. The findings emphasize the importance of temporal evaluation for realistic assessments and suggest that adaptive, tactic-aware methods hold promise for improving the detection of advanced and evolving threats.

Contents

Abstract **1**

1 Introduction **4**
1.1 Advanced Persistent Threat 4
1.2 APT life cycle 5
1.3 APT Examples 6
1.4 Detection Challenges 6

2 State of the Art **8**
2.1 Signature-based SOTA 8
2.1.1 SR2APT 9
2.1.2 APTSHIELD 9
2.2 Anomaly-based SOTA 10
2.2.1 UNICORN 11
2.2.2 APT-LLM 11
2.3 Behavior-based SOTA 12
2.3.1 LogShield 13
2.3.2 SHIELD 13

3 Methodologies **15**
3.1 Datasets 15
3.2 Linux APT 2024 Dataset 16
3.2.1 Dataset Construction and Environment 16
3.2.2 Captured Data and Format 17
3.2.3 Labeling Strategy 17
3.2.4 Advantages and Limitations 18
3.3 Dataset Analysis 18
3.3.1 Dataset Composition 18
3.3.2 MITRE ATT&CK Tactics and Techniques 19
3.3.3 Attack Scenarios and Simulated Threats 20
3.4 Data Cleaning 20
3.5 Feature Representation 21
3.5.1 TF–IDF Representation 21
3.5.2 BERT Embeddings 21
3.5.3 Comparison and Selection 22
3.6 Shallow Learning Baseline 22
3.6.1 Logistic Regression 22
3.6.2 Support Vector Machine 23
3.6.3 Random Forest 23

3.7 Deep Learning Models . . . 23
3.7.1 Transformer-Based Log Classification . . . 23
3.7.2 Sequence-Aware Classifier: DeepLog-Inspired LSTM . . . 25
3.8 Proposed Solution: Ensemble of Tactic-Specific BERT Classifiers . . . 26
3.8.1 General Idea . . . 26
3.8.2 Why an Ensemble is Suitable . . . 27
3.8.3 Implementation Details . . . 27
3.9 Evaluation Metrics . . . 28
3.9.1 ROC Curve . . . 28
3.9.2 Classification Report . . . 29
3.9.3 Which to Focus On . . . 29

4 Results and Discussion **30**
4.1 Results on Shallow Models . . . 30
4.1.1 Experimental Setup . . . 30
4.1.2 Random 80/20 Split . . . 30
4.1.3 Temporal Split . . . 31
4.1.4 Comparative Analysis . . . 31
4.1.5 Cumulative Tables . . . 32
4.2 Results on Deep Learning Models . . . 32
4.2.1 Experimental Setup . . . 32
4.2.2 Random 80/20 Split . . . 32
4.2.3 Temporal Split . . . 33
4.2.4 Comparative Analysis . . . 34
4.2.5 Cumulative Tables . . . 34
4.3 Proposed solution: BERT Ensamble . . . 34
4.3.1 Experimental Setup . . . 34
4.3.2 Random 80/20 Split . . . 35
4.3.3 Temporal Split . . . 36
4.3.4 Comparative Analysis . . . 37
4.3.5 Cumulative Tables . . . 37
4.3.6 Analysis of Missed Malicious Records . . . 38

5 Conclusions and Future Work **40**
5.1 Conclusions . . . 40
5.2 Future Work . . . 41

Chapter 1

Introduction

1.1 Advanced Persistent Threat

Cybersecurity has become one of the most critical domains in the digital era, where vast amounts of sensitive data are constantly exchanged, stored, and processed online. Among the various cyber threats, *Advanced Persistent Threats (APTs)* stand out due to their sophistication, stealth, and long-term objectives. These attacks are typically orchestrated by well-resourced adversaries—such as nation-states or organized cybercriminal groups—and are designed to compromise specific targets over extended periods.

Feature	**Traditional Attacks**	**APT Attacks**
Attacker Type	Lone individuals or small groups	Organized, well-resourced teams
Target	Random systems or users	Specific organizations or institutions
Goal	Quick profit or fame	Strategic gain or espionage
Method	Fast, one-time attacks	Stealthy, long-term presence
Tools	Common malware	Custom exploits and zero-days
Access Duration	Temporary	Prolonged and persistent

Table 1.1: Comparison of Traditional Attacks and APTs

According to the National Institute of Standards and Technology (NIST), APTs are distinguished by their ability to use multiple attack vectors, adapt to defensive measures, and pursue their objectives persistently [1]. Unlike conventional attacks, which are often opportunistic and short-lived, APTs involve methodical planning and execution aimed at data exfiltration, espionage, or disruption of critical infrastructure. The defining characteristics of APTs are embedded in the term itself:

- **Advanced:** These attackers utilize sophisticated tools, techniques, and procedures (TTPs), often including zero-day vulnerabilities and multi-stage exploits.
- **Persistent:** APT groups aim to maintain long-term access within a target environment, using stealthy techniques to avoid detection and ensure operational longevity.
- **Threat:** These attacks pose a significant danger to national security, critical infrastructure, and enterprise data, often leading to long-term consequences even after the attack is mitigated.

1.2 APT life cycle

The following is a restructured explanation of the attack process employed by Advanced Persistent Threat (APT) groups based on security literature including InfosecTrain (2024) [2]. This framework outlines five distinct stages that describe how APT attacks are typically executed, regardless of the attackers' ultimate objective. It also highlights how the attack steps may be adjusted based on whether the goal is to steal information, disrupt operations, or achieve other outcomes.

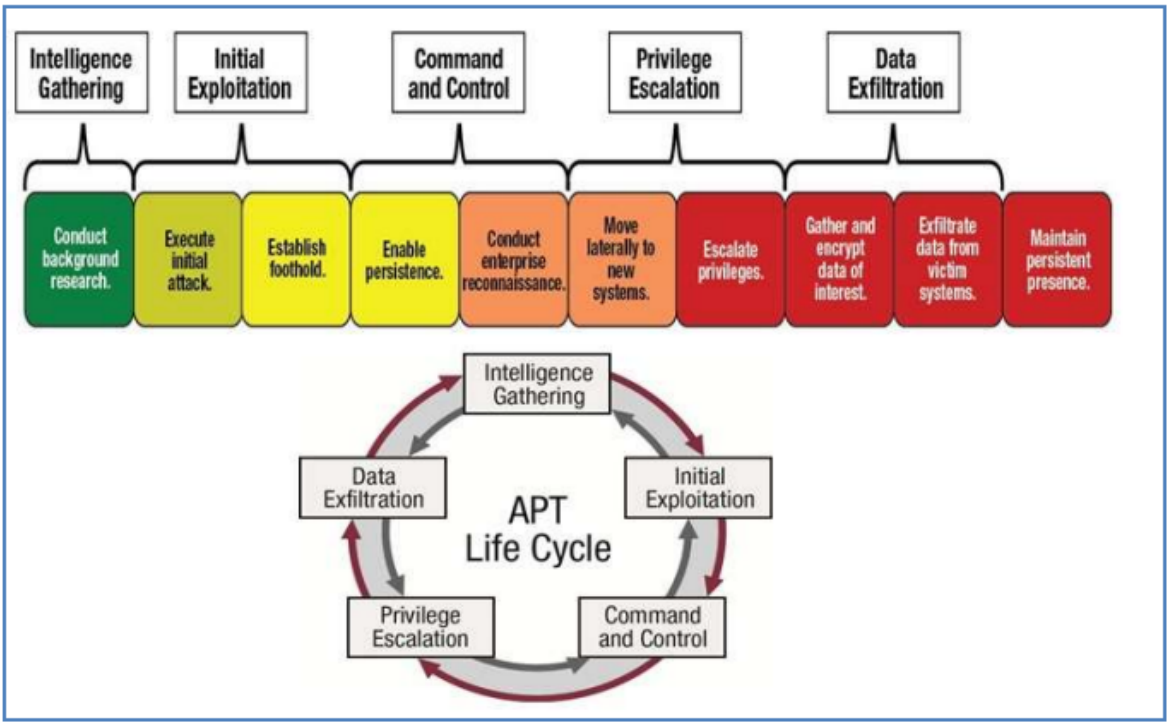

Figure 1.1: APT Attack Lifecycle Overview

Stage 1: Reconnaissance Every effective cyber attack starts with gathering intelligence. During this phase, attackers collect detailed information about the target, with the belief that a deeper understanding of the target's environment increases their chances of success.

Stage 2: Establishing a Foothold After the reconnaissance phase, attackers work to gain access to the target's systems or network. This step is critical, as it provides the entry point needed to carry out subsequent actions.

Stage 3: Lateral Movement/Remaining Undetected If the attacker's objective involves accessing sensitive data or compromising key components, they will maneuver within the target's network. During this phase, maintaining stealth is paramount, as moving laterally helps locate valuable assets while reducing the risk of detection.

Stage 4: Exfiltration/Disruption This phase is where the attack objectives begin to materialize. If the goal is data theft, attackers focus on retrieving and transmitting sensitive information back to their command center. Alternatively, if the aim is to disrupt operations, they may concentrate on disabling or destroying critical components of the target's infrastructure.

Stage 5: Post-Exfiltration/Post-Disruption In the final phase, attackers may con-

tinue to extract additional data or further impair critical systems. They also take measures to erase their tracks, ensuring a clean exit from the target's network.

1.3 APT Examples

In order to give a historical importance, following several real-life case studies of Advanced Persistent Threats (APTs) that demonstrate the growing complexity and impact of such attacks over the past decade. These case studies highlight different strategies and techniques employed by attackers to infiltrate systems, maintain persistence, and achieve their goals, which range from data theft to financial fraud

Titan Rain (2003–2006) Titan Rain was one of the earliest identified state-sponsored APT campaigns, originating from China and targeting U.S. defense contractors such as Lockheed Martin and NASA to exfiltrate sensitive but unclassified data. Although classified information was not confirmed stolen, the sustained coordination and duration of the campaign represented a significant precedent for modern APT operations [3].

Operation Aurora / Hydraq (2009) Operation Aurora—also referred to as Hydraq—was an APT targeting Google and other major corporations via zero-day exploits in Internet Explorer and Adobe Reader. Attackers deployed multi-layered malware and installed backdoors for prolonged access and espionage, marking one of the first high-profile attacks on commercial institutions [4].

Stuxnet (2010) Stuxnet was a sophisticated malware attack designed specifically to disrupt Iran's nuclear enrichment program by sabotaging industrial control systems. By exploiting multiple Windows zero-day vulnerabilities, it caused physical damage to centrifuges, demonstrating the potential for cyberattacks to translate into real-world kinetic impact [5].

RSA SecureID Attack (2011) In 2011, RSA was targeted via spear-phishing campaigns exploiting a zero-day Adobe Flash vulnerability. The attackers compromised credentials related to RSA's SecurID two-factor authentication system and exfiltrated data before the breach was contained, illustrating how supply-chain trust can be subverted for credential theft [6].

Carbanak (2013–2017) The Carbanak APT targeted financial institutions globally using spear-phishing emails to deliver custom malware. The attackers remained undetected for months, manipulating bank systems, siphoning money via fraudulent transactions and ATM controls, resulting in over €1billion in losses across dozens of countries [7].

SolarWinds (2019–2020) The SolarWinds Orion supply-chain compromise, attributed to APT29 (also known as Nobelium or Cozy Bear), involved injecting malicious code ("Sunburst") into trusted software updates. The attack affected thousands of organizations, including U.S. federal agencies and major corporations. The campaign demonstrated extreme stealth, long dwell time, and the danger of targeting widely used platforms [8].

1.4 Detection Challenges

The most critical issue with Advanced Persistent Threats (APTs) is that they are extremely difficult to detect in their early and stealthy stages. Many traditional security mechanisms fall short, allowing adversaries to linger undetected and execute complex

multi-stage campaigns. According to a recent comprehensive survey, the key obstacles in APT detection include [9]:

- **Evasion via Advanced and Living-off-the-Land Techniques**: Use of custom malware, polymorphism, and legitimate system tools to bypass signature-based defenses and avoid detection.
- **Low-and-Slow Behavior and Long Dwell Time**: Adversaries often maintain subtle, gradual activity over months, blending into normal operations and making anomalies hard to spot.
- **Multi-Stage Attack Chains**: APTs unfold across several stages—initial compromise, lateral movement, persistence, exfiltration—each individually benign, complicating holistic detection.
- **Encrypted Communications and Covert Channels**: Use of encryption (e.g. HTTPS, DNS tunneling) conceals command-and-control traffic and exfiltration, limiting visibility.
- **Alert Fatigue and Prioritization Difficulties**: Behavioral detection and anomaly systems generate high volumes of alerts with many false positives, overwhelming analysts and delaying response.
- **Resource and Expertise Constraints**: Effective APT detection—including threat hunting, log correlation, and machine learning methods—demands advanced tooling and skilled analysts, which many organizations lack.

Chapter 2

State of the Art

Advanced Persistent Threat (APT) detection has evolved into a multifaceted research area, where detection strategies are typically classified into three major categories: signature-based, anomaly-based, and behavior-based approaches. Each of these classes reflects a different perspective on how malicious activity is recognized and addressed within complex and often stealthy attack campaigns.
Signature-based methods rely on predefined indicators of compromise, such as known malware hashes, byte patterns, or network signatures. These techniques are widely used in antivirus engines and intrusion detection systems, offering high precision in identifying known threats. However, they are inherently limited against novel, polymorphic, or zero-day APT variants that do not match existing patterns.
In contrast, **anomaly-based detection** aims to identify deviations from a model of normal system, user, or network behavior. This category often leverages machine learning or statistical profiling to detect suspicious events that do not conform to historical norms. While effective at uncovering previously unseen attacks, these methods are typically more prone to false positives and require robust baselining and tuning.
Lastly, **behavior-based approaches**focus on detecting malicious intent by monitoring the sequence and context of actions, often aligned with attacker tactics and techniques. Rather than flagging individual anomalies, these systems evaluate broader behavioral patterns—such as lateral movement, privilege escalation, or data exfiltration—often referencing frameworks like MITRE ATTCK to interpret events in a threat-centric way.
This tripartite classification is commonly adopted in recent literature, offering a useful lens through which to evaluate the strengths, limitations, and complementary nature of existing APT detection solutions [10, 11].
In this chapter, the most relevant and diverse solutions from each category will be analyzed and discussed in detail.

2.1 Signature-based SOTA

Signature-based APT detection methods rely on predefined rules, indicators of compromise, or specific patterns to recognize malicious activity. A representative example is SR2APT [12], which applies a rule- and signature-based approach to detect multistage APT campaigns. By monitoring system logs for file operations such as `rename`, `copy`, or `move`, SR2APT employs string-matching techniques to flag suspicious activities associated with known attack stages. Another relevant work is APTSHIELD [13], a real-time host-based detection framework for Linux systems. It leverages kernel audit logs and matches

observed events against MITRE ATT&CK signatures to efficiently recognize malicious behaviors. Both frameworks are strong representatives of the signature-based family, as they demonstrate the core idea of detecting APTs through predefined and well-established patterns rather than adaptive anomaly or behavior modeling.

2.1.1 SR2APT

SR2APT is a host-based detection and response framework specifically designed to counter multistage advanced persistent threats (APTs). Its architecture is structured around two key components: a *Graph Convolutional Network (GCN)-based detection engine* and a *Deep Reinforcement Learning (DRL)-driven decision engine* [12].
The detection engine processes *system logs that are modeled as provenance graphs.* Instead of analyzing raw event sequences, SR2APT constructs subgraphs around nodes of interest (e.g., specific processes or commands) to capture contextual relationships. These subgraphs are then classified by a GCN into one of seven categories—six corresponding to distinct APT stages (reconnaissance, discovery, persistence, privilege escalation, asset discovery, and exfiltration) plus a benign class. By leveraging structural properties of graphs rather than isolated log entries, the system improves its ability to distinguish subtle attack behaviors that may span multiple system interactions. Experimental evaluation demonstrated that this approach achieves **94% classification accuracy**, outperforming traditional models such as SVM, CNN, and LSTM.
Once an alert is raised by the detection engine, it is passed to the decision engine, which determines the most suitable *defensive response action.* This module formulates the response strategy as a sequential decision-making problem under uncertainty, solved using a deep Q-network (DQN) algorithm. The engine considers not only the detected stage but also historical context, interdependencies among APT stages, and the trade-off between the cost of defense actions and potential attack damage. For instance, while aggressive defenses can halt intrusions, they risk disrupting benign activities; conversely, passive responses may allow attacks to progress. By modeling these dynamics, the DQN learns an adaptive policy that maximizes cumulative security benefits while minimizing unnecessary interventions.
The **innovative contributions** of SR2APT are twofold. First, it is the *first framework to apply GCNs for classifying provenance subgraphs of system logs*, enabling fine-grained detection of different APT stages rather than binary malicious/benign judgments. Second, it integrates *strategic alert response through reinforcement learning*, allowing defenses to be optimized dynamically based on attack progression and system state. This joint design ensures that APTs are often intercepted in earlier stages while reducing false alarms and costly mistaken responses to benign activity.
Overall, SR2APT represents a step forward from purely signature-based or anomaly-based methods by coupling structural graph learning for detection with adaptive reinforcement learning for response, thus offering a more agile and cost-effective defense against sophisticated, multistage intrusions.

2.1.2 APTSHIELD

APTSHIELD is a detection system specifically designed for Linux hosts, aiming to provide stable, efficient, and real-time defense against advanced persistent threats (APTs) [13]. The framework addresses several shortcomings of existing endpoint detection and response

solutions, such as the difficulty of selecting reliable data sources, the heavy computational burden caused by long-term monitoring, and the challenge of producing accurate real-time alerts. To tackle these issues, APTSHIELD adopts a three-stage architecture that integrates data collection, log compaction, and rule-based detection. For data collection, the authors performed an extensive evaluation of different kernel-level logging tools and concluded that `auditd` provides the most suitable balance of stability, performance, and semantic richness. Auditd records kernel audit logs that cannot easily be tampered with, ensuring that critical events such as file operations, process activities, and system calls are captured comprehensively. Since APT campaigns often last for weeks or months, raw system logs quickly become too large to handle efficiently. To address this, APTSHIELD introduces two compaction techniques: redundant semantics skipping, which eliminates repeated events that do not change the state of entities, and non-viable entity pruning, which removes terminated processes or inactive files that no longer influence the attack chain. These techniques maintain the semantic integrity of the provenance graph while significantly reducing storage and memory consumption, enabling real-time operation without sacrificing detection accuracy. The core detection engine of APTSHIELD is built upon the MITRE ATT&CK framework, where suspicious behaviors are defined through a system of labels assigned to processes and files. Labels reflect properties such as whether a process executed a sensitive command or whether a file contains untrusted data, and they are propagated across the dependency graph through well-defined transfer rules. By aggregating these labels, the system reconstructs the entire context of an attack and produces alerts when certain combinations match predefined adversarial tactics, techniques, and procedures. This approach allows APTSHIELD to recognize sophisticated threats including webshell intrusions, fileless attacks, and remote access trojans, while maintaining low false positive rates. The main contributions of the framework are its rigorous evaluation of Linux data sources for APT monitoring, its real-time semantics-aware data reduction methods that ensure constant memory overhead during long-term monitoring, and its integration of an ATT&CK-driven labeling system that provides interpretable whole-chain detection of complex campaigns. Despite its efficiency and robustness, APTSHIELD remains a signature-based system because it relies on predefined ATT&CK rules and expert-specified transfer mechanisms to identify malicious activity. Rather than adapting dynamically to unknown threats, it detects attacks by matching system events against established signatures of adversarial behavior, which firmly places it in the category of signature-driven APT detection methods.

2.2 Anomaly-based SOTA

Anomaly-based APT detection approaches identify suspicious activities by modeling normal behavior and flagging deviations. One representative framework is UNICORN [14], which uses provenance graph sketching and runtime analysis to capture long-term system behavior. By building compact representations of provenance data, UNICORN is able to efficiently detect stealthy and low-and-slow APT campaigns that evade traditional methods. More recently, APT-LLM [15] has explored the use of large language models and autoencoders to embed provenance traces into semantic representations, enabling the detection of subtle anomalies even in scenarios with extreme data imbalance. These two frameworks are strong representatives of the anomaly-based family, as they demonstrate how modeling deviations from normal system activity can effectively uncover sophisticated and previously unseen APTs.

2.2.1 UNICORN

UNICORN is a provenance-based intrusion detection framework that targets the unique challenges of advanced persistent threats (APTs) [14]. Unlike signature-driven systems, UNICORN adopts an anomaly-based approach: instead of relying on predefined attack rules, it learns models of normal system execution and flags deviations as potential intrusions. Its design specifically addresses the "low-and-slow" nature of APTs, which unfold gradually and often mimic benign activities to evade conventional detectors.

At its core, UNICORN leverages whole-system provenance, representing system execution as a directed acyclic graph that captures causal relationships among processes, files, and network connections. This provenance-centric view allows the system to reason about events that are temporally distant but causally related, a key advantage in tracking long-term attack chains. Since raw provenance graphs grow continuously and can become prohibitively large, UNICORN introduces a *graph sketching* mechanism. This technique incrementally summarizes streaming provenance data into fixed-size structures that preserve essential graph properties while keeping computation and storage overhead manageable.

Building on this sketching layer, UNICORN constructs *graph histograms* that describe recurring structural patterns in the provenance graph. By iteratively propagating labels across multiple hops in the graph, the system encodes contextual and temporal information about entities and their interactions. To account for evolving system states over time, UNICORN employs a "gradual forgetting" mechanism, discounting outdated behaviors while retaining causally relevant context. This ensures that models remain representative of legitimate system dynamics without being poisoned by long-running intrusions.

During training, UNICORN clusters graph sketches to build an *evolutionary model* of normal execution. This model captures different meta-states of a system (e.g., initialization, steady workload, failure recovery) and the valid transitions between them. At runtime, new sketches derived from streaming provenance data are compared against this evolutionary model. Anomalies are flagged either when sketches fail to fit any known cluster or when transitions deviate from the learned evolution of system behavior.

The key innovations of UNICORN include: (i) the introduction of sketch-based, time-weighted provenance encoding that can efficiently summarize long-running system executions, (ii) an evolutionary modeling strategy that captures system dynamics without updating models during deployment, thus preventing attacker-driven poisoning, and (iii) a scalable streaming architecture that processes provenance graphs in real time without requiring full in-memory graph storage. Evaluation results show that UNICORN achieves significant improvements in accuracy and precision compared to previous provenance-based detectors, while maintaining low computational and memory overheads.

UNICORN is clearly an anomaly-based method, since it does not depend on predefined signatures or ATT&CK-style rules to identify threats. Instead, it builds a baseline of benign behavior and treats deviations from this baseline as suspicious, which makes it well-suited to detect zero-day exploits and previously unseen APT strategies. Its ability to identify intrusions arises not from rule matching, but from recognizing when the causal structure of system execution diverges from established normal patterns.

2.2.2 APT-LLM

APT-LLM is a recent anomaly detection framework designed to tackle the difficulties of identifying advanced persistent threats (APTs) in highly imbalanced datasets [15]. Unlike

classical rule-based or handcrafted-feature systems, APT-LLM builds on pre-trained large language models (LLMs) to transform raw provenance traces into semantically meaningful embeddings. These embeddings are then modeled using autoencoders to establish a baseline of normal system behavior and detect deviations that suggest malicious activity. The framework starts by converting low-level system events—such as process execution, file operations, or network activity—into descriptive textual sentences. Pre-trained LLMs including BERT, ALBERT, RoBERTa, DistilBERT, and MiniLM are used to generate dense vector representations of these sentences, thereby capturing the contextual and semantic relationships between process actions. These high-dimensional embeddings act as rich behavioral signatures of system activity, allowing the detection system to pick up on subtle irregularities that might otherwise blend into benign background noise.
For anomaly detection, APT-LLM employs different types of autoencoders: a standard autoencoder for deterministic reconstruction, a variational autoencoder (VAE) that introduces a probabilistic latent space to generalize beyond training data, and a denoising autoencoder (DAE) that improves robustness by reconstructing clean embeddings from noisy inputs. Self-attention mechanisms are incorporated into the encoder to highlight dependencies between features, enhancing the model's ability to discriminate between normal and anomalous behaviors. At runtime, embeddings derived from new system traces are reconstructed by the trained autoencoder. Records with reconstruction errors above a threshold are flagged as potential APT activity.
APT-LLM's main contributions lie in: (i) reframing provenance traces as textual descriptions that can be effectively encoded with LLMs, (ii) combining multiple autoencoder architectures with attention mechanisms to capture different aspects of normal process behavior, and (iii) demonstrating strong performance on real-world DARPA Transparent Computing datasets, where malicious traces account for as little as 0.004% of the data. Experimental results show that the framework significantly outperforms traditional anomaly detectors such as OC-SVM, Isolation Forest, and DBSCAN, particularly in extreme class imbalance scenarios.
APT-LLM is firmly situated within the anomaly-based paradigm. It does not rely on fixed signatures or handcrafted attack rules; instead, it models typical system dynamics through embeddings and reconstruction, treating departures from this learned baseline as evidence of suspicious activity. This design makes it particularly adept at surfacing stealthy, previously unseen APT campaigns that elude conventional detection strategies.

2.3 Behavior-based SOTA

Behavior-based APT detection frameworks focus on monitoring the sequence and context of actions to identify malicious intent. LogShield [16] is a state-of-the-art framework that leverages transformer-based models with self-attention to analyze event sequences derived from provenance graphs. By capturing temporal and contextual dependencies, LogShield can detect complex APT attack patterns with high accuracy, outperforming traditional LSTM-based approaches. Similarly, SHIELD [17] integrates anomaly detection, graph-based analysis, and large language model-driven reasoning to uncover hidden attack behaviors while providing interpretable explanations. Both frameworks exemplify the behavior-based family, as they evaluate patterns of attacker actions over time rather than relying solely on individual anomalies or static signatures, making them particularly effective against sophisticated and stealthy APT campaigns.

2.3.1 LogShield

LogShield is a transformer-based detection framework that leverages self-attention mechanisms to identify advanced persistent threats (APTs) from system logs [16]. Unlike earlier provenance-driven or LSTM-based approaches, LogShield reframes the problem as a sequence modeling task, where system logs and provenance traces are treated as event sequences analogous to natural language. This design allows the model to capture both local and global dependencies among events, which is critical for detecting stealthy "low-and-slow" APT behaviors.

The framework begins with the construction of provenance graphs from raw system logs. These graphs are then decomposed into event traces, which represent causally linked activities across processes, files, and network operations. To prepare these traces for learning, LogShield introduces two specialized embedding strategies. The first is a *log embedding*, which encodes object-action pairs (e.g., "File Read", "Process Create") as tokens that can be processed by the transformer encoder. The second is a *temporal embedding*, which captures timing differences between parent and child events, thereby reflecting the slow, staged nature of APT campaigns. By integrating temporal context, the system is able to distinguish benign sequences from malicious ones that deliberately unfold over long periods.

At the modeling stage, LogShield employs a transformer architecture, where multi-head self-attention layers learn contextual relationships across log sequences. To optimize learning, a customized objective function—the *log key cross-entropy loss*—is applied. By randomly masking parts of the input and requiring the model to reconstruct them, the system learns to encode benign sequence patterns while improving its ability to isolate deviations caused by intrusions. This training strategy ensures that LogShield generalizes well even on large and imbalanced datasets.

The key innovations of LogShield include: (i) the introduction of temporal embeddings to capture the timing-sensitive characteristics of APT activity, (ii) the integration of provenance-informed log embeddings with a transformer architecture, enabling richer contextual modeling than RNN-based methods, and (iii) the use of a self-supervised objective function tailored to log data, which enhances separation between benign and malicious sequences. Experimental results on DARPA OpTC and TC E3 datasets confirm its effectiveness, with LogShield achieving F1-scores above 95%, consistently outperforming LSTM and other transformer baselines.

LogShield is best described as a behavioral-based detection method. Rather than relying on static rules or predefined attack signatures, it learns the normal temporal and structural patterns of system execution, and flags traces that deviate from these learned baselines. Its detection power arises from recognizing abnormal causal and temporal dynamics in system behavior, making it well-suited to uncovering subtle, stealthy, and previously unseen APT campaigns.

2.3.2 SHIELD

SHIELD is a provenance-based detection and investigation framework that integrates anomaly detection, graph analysis, and large language model (LLM) reasoning to uncover advanced persistent threats (APTs) [17]. Traditional provenance-based methods often struggle with high false positive rates or limited interpretability, while purely ML-driven approaches face challenges with concept drift and lack of explainability. SHIELD addresses these limitations by combining statistical anomaly detection with graph-based

event correlation and contextual reasoning powered by LLMs. This design not only detects intrusions but also provides human-readable attack narratives aligned with the cyber kill chain, significantly reducing analyst workload.

The pipeline begins with a *deviation analyzer* that uses statistical outlier detection (e.g., Local Outlier Factor) to flag unusual system events. These anomalies are then expanded into local provenance graphs capturing both the suspicious processes and their immediate lineage. A *graph analyzer* module refines these graphs by identifying infection entry points, propagating suspicious tags through causally related entities, pruning benign subgraphs, and clustering malicious nodes into dense communities using community detection algorithms such as Louvain. This ensures that only behaviorally meaningful clusters of suspicious activity are retained for deeper analysis.

At this stage, the *LLM analyzer* takes over, applying a multi-stage reasoning process to interpret suspicious communities. It examines event sequences for abnormal behaviors, analyzes temporal dependencies among processes, and synthesizes these into structured attack summaries. To improve trustworthiness, the system attaches dynamic confidence scores to its findings: benign patterns gradually decrease in confidence through a decay mechanism, while recurring suspicious behaviors are reinforced. These scores determine whether alerts are raised immediately or kept under observation. Finally, a *temporal correlation engine* integrates alerts across time windows, maintaining a long-term perspective on evolving attacks while efficiently managing memory through rolling provenance updates.

SHIELD introduces several innovations: (i) the integration of LLM-driven reasoning into provenance-based APT detection, enabling both high detection accuracy and interpretable explanations, (ii) a dynamic confidence scoring mechanism with reinforcement and decay, which balances real-time detection with long-term monitoring of stealthy intrusions, (iii) the use of graph pruning and community detection to reduce noise while retaining essential attack traces, and (iv) a framework capable of producing detailed attack summaries with indicators of compromise and explicit mapping to kill chain stages. Evaluations on DARPA CADETS, THEIA, Public Arena, and Blind Eagle datasets show that SHIELD consistently achieves high recall and precision, with perfect precision in several cases, while reducing false positives by orders of magnitude compared to baseline methods.

SHIELD exemplifies a behavioral-based detection approach. Instead of relying on predefined signatures or static rules, it models normal system activities and searches for deviations that indicate malicious intent. By combining statistical anomaly detection with causal graph analysis and LLM reasoning, SHIELD captures the behavioral essence of APT campaigns—slow progression, abnormal causal chains, and coordinated malicious communities—making it highly effective at surfacing stealthy, long-lived, and previously unseen attacks.

Chapter 3

Methodologies

3.1 Datasets

In the domain of Advanced Persistent Threat (APT) detection, the availability of comprehensive and realistic datasets is crucial for the development and evaluation of detection methodologies. Two of the most widely referenced datasets in this field are the CICAPT-IIoT and the DARPA Operationally Transparent Cyber (OpTC) datasets.
The CICAPT-IIoT dataset was developed by the Canadian Institute for Cybersecurity and is specifically tailored to reflect attacks in Industrial Internet of Things (IIoT) environments. It contains labeled network traffic capturing multi-stage APT scenarios across different phases of the cyber kill chain [18].
The DARPA OpTC dataset, released by the Defense Advanced Research Projects Agency, offers a rich collection of system and network telemetry captured from real enterprise-like environments. It is particularly valuable due to its detailed host-level logs and its focus on transparency and reproducibility in cyber defense research [19].
These datasets are widely used due to their realistic attack simulations, structured multi-stage threats, and comprehensive logging capabilities, which make them well-suited for training and evaluating APT detection models.
For the purposes of this thesis, the Linux APT 2024 Dataset has been selected as the main dataset for experimentation. This dataset is designed to simulate realistic APT scenarios targeting Linux-based systems, with a strong focus on capturing full attack chains including reconnaissance, privilege escalation, lateral movement, and data exfiltration.
One of the primary reasons for choosing this dataset is its emphasis on modern threat behaviors within Unix-like environments, which are often underrepresented in traditional cybersecurity datasets. Given the prevalence of Linux systems in both enterprise and cloud infrastructures, analyzing APT activity in such contexts is highly relevant for real-world applications.
Furthermore, the Linux APT 2024 Dataset provides detailed, labeled logs from multiple data sources, including shell command histories, system logs, and process monitoring tools. This diversity in telemetry makes it particularly suitable for developing and evaluating behavioral-based and anomaly-based detection strategies.
Its clear structure, ground truth labeling, and focus on advanced persistent behaviors make it a valuable resource for research aimed at detecting complex, multi-stage threats in modern computing environments [20].

3.2 Linux APT 2024 Dataset

The Linux APT 2024 Dataset is a recently developed benchmark specifically focused on evaluating intrusion detection systems and analyzing advanced persistent threats within Linux environments [20]. Unlike earlier datasets that are mostly Windows-based or network-centric, this dataset provides an in-depth view of Linux-specific threats and system behaviors.

3.2.1 Dataset Construction and Environment

The Linux APT 2024 dataset was created in a controlled and realistic laboratory environment. This environment was composed of three Linux-based computers organized across different internal networks to simulate how real systems communicate in an organization. To monitor and record all activities during the experiments, the researchers used a centralized logging platform called *WAZUH*, which is a security information and event management (SIEM) system. This system collects logs (i.e., detailed records of events) from all machines involved, including logs from an antivirus solution (Kaspersky Security Center) installed on one of the endpoints.
The antivirus plays an important role in this setup. It allows the researchers to capture how a typical endpoint protection system responds to various attack attempts. By forwarding its logs to the SIEM, it helps to track whether the malicious activity is detected, blocked, or allowed to proceed.
To build the dataset, the researchers executed various cyberattacks designed to mimic the behavior of real-world hacker groups known as Advanced Persistent Threats (APTs). These groups, such as APT28, APT29, APT41, and Turla, are known for carrying out long-term, stealthy attacks against high-value targets. The simulated attacks included techniques such as stealing administrator privileges (privilege escalation), hiding activity, logging keystrokes (keylogging), and exploiting known software vulnerabilities.
In the simulated environment, one of the three Linux machines was designated as the attack source, while the other two acted as normal, legitimate endpoints. The attacker machine was configured to mimic real-world intrusion scenarios, launching targeted operations against the other systems. Although the attacks were carried out internally within the lab environment, several of the simulated threat vectors reflected attacks that typically originate from the internet.
Importantly, each attack in the dataset was documented and categorized using the MITRE ATT&CK framework. This framework, developed by the MITRE Corporation, is a globally recognized knowledge base that systematically organizes known adversarial behaviors observed in real-world cyberattacks. Instead of focusing on specific tools or malware, it classifies attacks according to the goals (tactics) and the specific methods (techniques) used by threat actors to achieve them. For example, common tactics include "Initial Access," "Privilege Escalation," or "Data Exfiltration," and each is associated with various techniques such as exploiting a vulnerability or using stolen credentials. By mapping attacks to this structure, the dataset becomes more interpretable, enabling researchers to analyze and compare threat behaviors in a standardized way. It also facilitates the development of detection strategies aligned with industry practices and threat modeling.

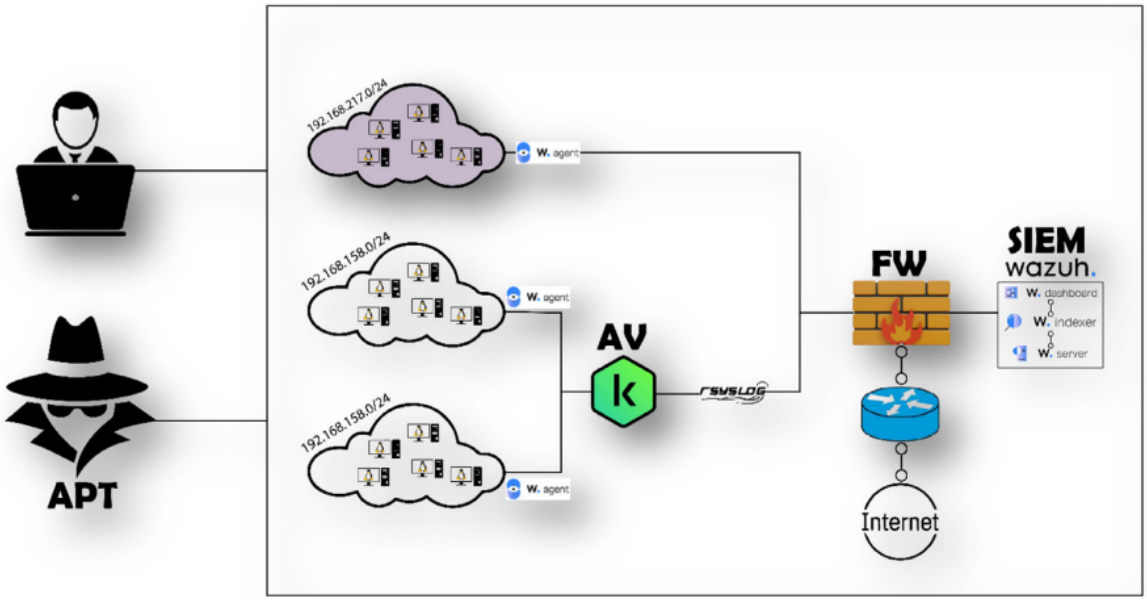

Figure 3.1: Linux APT 2024 Dataset building enviroment

3.2.2 Captured Data and Format

The logs are stored in structured CSV files, with a processed version also available in XLSX format. The data spans from October 1, 2023, to January 7, 2024, and contains over 125,000 entries. Each record includes detailed metadata such as timestamps, command logs, file hashes, source logs, and MITRE-aligned tags. A configuration XML file (`local_rules.xml`) defines detection rules and system-specific logging policies.

3.2.3 Labeling Strategy

The dataset includes both benign and malicious logs, which are essential for supervised machine learning tasks such as classification. The labeling process was based on a combination of rule-based detection and anomaly detection techniques. Specifically, the researchers used custom-defined detection rules within the WAZUH SIEM, as well as pattern-matching tools like YARA, to flag suspicious activity. These rules were designed to detect specific commands, known malware behaviors, exploit signatures, and unusual deviations from normal system activity.
Although the labeling was not done manually, it benefits from a multi-layered detection pipeline. The combination of static rules, dynamic behavioral analysis, and correlation with known indicators of compromise (IoCs) helps ensure a relatively high degree of confidence in the accuracy of the labels. For example, activity matching a known APT technique (such as privilege escalation through a specific CVE exploit) would be flagged as malicious, whereas regular system operations like file browsing or benign shell commands would be labeled as normal.
Furthermore, the logs are linked with MITRE ATT&CK identifiers, reinforcing the semantic meaning behind each labeled instance. This structured approach reduces the risk of arbitrary or noisy labeling and makes the dataset suitable for training machine learning models that require clear distinctions between classes. In the final version of the dataset, approximately 24,852 records are labeled as malicious and 101,046 as benign, offering

a substantial and balanced base for experimentation with both anomaly detection and supervised learning techniques.

3.2.4 Advantages and Limitations

Advantages:

- Realistic simulation of APT activity targeting Linux systems, including modern attack payloads and methods.
- Comprehensive logging from both system and network layers, integrating 33 different Linux log resources.
- Strong alignment with the MITRE ATT&CK framework facilitates standardization and cross-study comparisons.
- Dual availability of raw and processed formats supports a range of analytical approaches.

Limitations:

- The dataset is Linux-specific, which may limit its applicability to Windows or hybrid environments.
- Some of the simulations, while realistic, may not fully capture the unpredictability and stealthiness of real-world APT actors.
- The reliance on a preconfigured SIEM (WAZUH) and custom rules might introduce a detection bias in labeling.

3.3 Dataset Analysis

The Linux APT 2024 dataset provides a rich foundation for studying intrusion attempts and advanced persistent threats in Linux environments.

3.3.1 Dataset Composition

The dataset is composed of more than 125,000 log entries gathered over a three-month period (October 2023 to January 2024). To handle storage constraints, the logs were distributed across 17 CSV files, each containing up to 10,000 records, in addition to a merged file that consolidates all records. A processed version in XLSX format is also provided for structured analysis.
Each record contains a diverse set of attributes: timestamps, agent identifiers, raw command logs, rule descriptions, file hashes (MD5, SHA-256), system paths, IP addresses, and mappings to MITRE ATT&CK tactics and techniques. This multidimensional structure enables researchers to analyze both low-level technical details and high-level adversarial strategies.
The dataset distinguishes between benign and malicious activity, a critical feature for supervised learning tasks. Out of the total 125,898 records, 101,046 are categorized as benign, while 24,852 are flagged as malicious or suspicious.

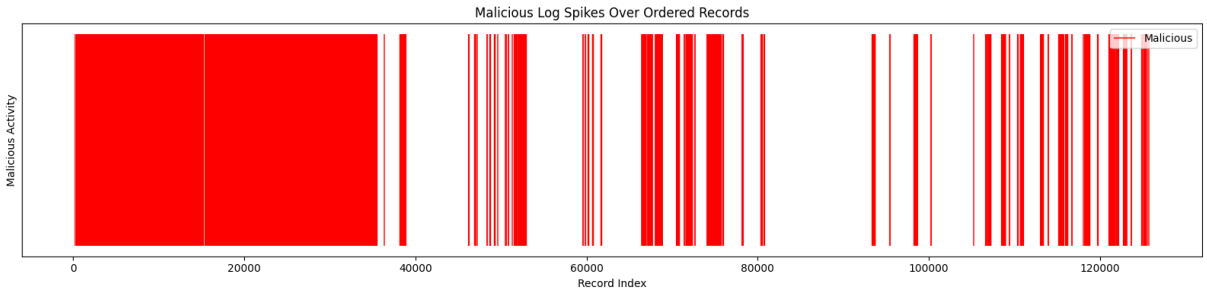

Figure 3.2: Linux APT 2024 Dataset labels cronological distribution

3.3.2 MITRE ATT&CK Tactics and Techniques

By aligning records with the MITRE ATT&CK framework, the dataset provides a standardized view of adversarial behavior. Among the tactics, the most frequent are *Defense Evasion* (18,124 records), *Initial Access* (16,624), and *Privilege Escalation* (17,164), followed by *Discovery* (14,058). This distribution highlights how attackers prioritize stealth, entry vectors, and privilege abuse in Linux environments.

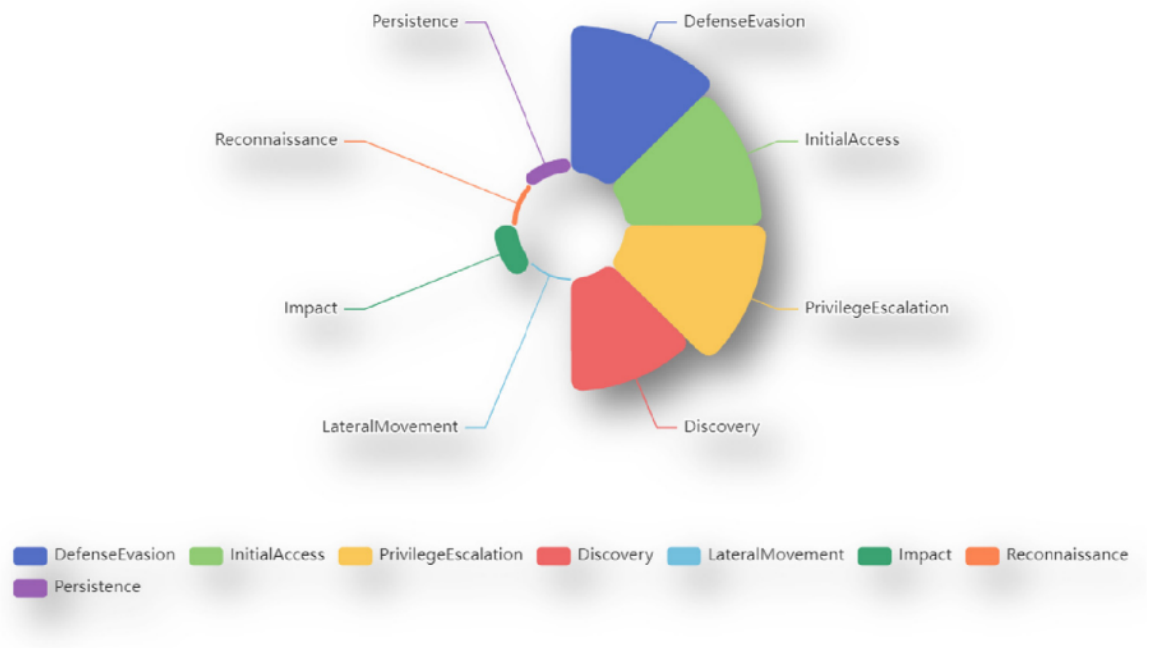

Figure 3.3: Linux APT 2024 Dataset tatics distribution

At the technique level, the most common include exploitation of public-facing applications (14,919 instances), file and directory discovery (13,999), and the use of valid accounts (6,820). Other techniques, such as privilege escalation exploits and process discovery, are also represented, though less frequently. This mapping illustrates the persistence of classic intrusion vectors while also capturing the evolution of more recent exploitation methods.

3.3.3 Attack Scenarios and Simulated Threats

The dataset incorporates multiple APT groups and real-world attack emulations. Well-known groups such as APT28, APT29, APT41, and Turla were simulated, alongside targeted exploits like the Apache Struts vulnerability, keyloggers, and Linux privilege escalation payloads. Each scenario was designed to reflect realistic adversarial objectives, such as credential theft, lateral movement, and data exfiltration. The combination of APT-specific behaviors with general intrusion attempts ensures a varied and representative sample of modern threats.

3.4 Data Cleaning

Before conducting any machine learning or anomaly detection experiments, it is essential to preprocess the raw logs in order to avoid misleading results. The Linux APT 2024 dataset, in its original form, contains a large amount of metadata that, if left untreated, could artificially inflate the performance of detection models without actually improving their ability to generalize.
The cleaning process is implemented through a custom function (`clean_log`), which systematically removes elements that might introduce bias or data leakage. In particular:

- **Email and IP addresses:** These values are unique identifiers that trivially differentiate benign from malicious activity, but they do not reflect generalizable attack patterns. Keeping them would risk models simply memorizing addresses rather than learning behavioral signatures.
- **Hexadecimal strings and numeric values:** Log entries often contain hashes, inode IDs, user IDs, ports, or timestamps. These are not meaningful features for APT detection, since they vary from one environment to another and cannot be relied upon as discriminative indicators across different contexts.
- **File paths, system tokens, and libraries:** Paths such as `/usr/bin/` or references to `drivers`, `modules`, or `tmp` folders may be environment-specific. If not removed, models might incorrectly associate these with malicious activity due to their frequency in simulated attacks, resulting in overfitting.
- **Usernames and architecture-specific tokens:** Usernames (e.g., `root`, `admin`) or architecture labels (e.g., `x86_64`, `arm64`) are again too specific to the experimental setup. Their presence may cause models to learn dataset artifacts rather than attacker behavior.
- **Formatting and punctuation:** Removing non-alphanumeric characters and collapsing whitespace ensures textual uniformity, facilitating tokenization and vectorization in later steps.

The cleaned logs are stored in a new column (`full_log_clean`), which provides a standardized textual representation of each event. This process helps ensure that models trained on the dataset are not biased by spurious identifiers but instead learn to detect attack patterns that are semantically meaningful and transferable to other environments. Without this cleaning phase, the accuracy of APT detection models would be unrealistically high. In other words, models might appear to perform extremely well simply

because they exploit low-level artifacts unique to the dataset (e.g., a specific IP address always linked to malicious traffic), rather than learning generalizable adversarial behaviors. By enforcing this preprocessing step, the experiments are made more robust and the evaluation results better reflect real-world applicability.

3.5 Feature Representation

In order to transform raw log data into numerical representations suitable for machine learning, two different embedding approaches were tested: a traditional *Term Frequency–Inverse Document Frequency* (TF–IDF) representation and contextual embeddings obtained with a transformer-based language model (BERT).

3.5.1 TF–IDF Representation

The TF–IDF approach is a classical method in text mining that assigns weights to terms according to their importance within a corpus [21]. The weight increases proportionally with the frequency of a word in a document but is offset by the frequency of that word across all documents, thus reducing the impact of very common tokens.
In this work, TF–IDF vectors were extracted using `scikit-learn`'s `TfidfVectorizer`, configured with a maximum of 100 features, bi-grams, and English stop-word removal:

```
vectorizer = TfidfVectorizer(
    max_features=100,
    ngram_range=(1, 2),
    stop_words='english')
X_train_vec = vectorizer.fit_transform(X_train)
X_test_vec = vectorizer.transform(X_test)
```

This representation produces sparse vectors that capture surface-level lexical information, but it does not account for semantic similarity between different words or contexts.

3.5.2 BERT Embeddings

To capture richer contextual information, we also employed embeddings derived from a pre-trained transformer model, namely BERT (Bidirectional Encoder Representations from Transformers) [22]. Unlike TF–IDF, BERT generates dense, contextualized embeddings where the meaning of each token depends on its surrounding words. This property is particularly useful in the analysis of log data, where the same token (e.g., a command) may have different implications depending on its context.
The embedding process was implemented by extracting the hidden state of the special `[CLS]` token, which is commonly used as a representation of the entire sequence:

```
def get_bert_embeddings(text_list, tokenizer, model, batch_size=32):
    embeddings = []
    model.eval()
    with torch.no_grad():
        for i in tqdm(range(0, len(text_list), batch_size), desc="Embedding logs"):
            batch = text_list[i:i+batch_size]
```

```
            inputs = tokenizer(batch, return_tensors='pt',
                               padding=True, truncation=True,
                               max_length=64)
            outputs = model(**inputs)
            cls_embeddings = outputs.last_hidden_state[:, 0, :]
            embeddings.append(cls_embeddings)
    return torch.cat(embeddings).numpy()
```

The resulting embeddings are dense vectors that encode semantic and syntactic relationships in the log messages.

3.5.3 Comparison and Selection

While both approaches produced viable features for classification tasks, the results of preliminary experiments indicated that BERT embeddings significantly outperformed TF–IDF vectors in terms of classification accuracy and generalization. This performance gap can be attributed to BERT's ability to model contextual meaning rather than relying solely on token frequency statistics.
For this reason, subsequent experiments and deep learning models in this work will primarily rely on BERT-based embeddings.

3.6 Shallow Learning Baseline

All three models were trained on preprocessed log data represented by both TF–IDF vectors and BERT embeddings. Two different train-test split strategies were applied throughout the project to ensure robustness of the evaluation: (i) a random 80/20 split, and (ii) a chronological 80/20 split, where the training set consists of earlier records and the test set of later ones. The chronological split was introduced to better mimic real-world deployment scenarios, where models are trained on past events and then applied to future unseen activity. It was not possible to isolate a completely unknown APT scenario as the test set, since the attacks in the dataset were conducted in parallel and interleaved within the same time window. Both split strategies were therefore used consistently across all experiments in this work. Evaluation metrics included accuracy, precision, recall, F1-score, and ROC AUC, providing a comprehensive view of classifier performance. Class imbalance was addressed by computing balanced class weights, ensuring that the models did not become biased toward the majority class.
To establish a benchmark for intrusion detection on the Linux APT 2024 dataset, a set of shallow learning models was implemented. These baseline models serve two purposes: first, to provide an initial reference point against which more advanced deep learning architectures can later be compared; and second, to test whether traditional classifiers, when combined with different feature representations, are sufficient to capture the behavioral patterns of malicious activity.

3.6.1 Logistic Regression

Logistic Regression is a widely used linear model for binary classification tasks [23]. It estimates the probability of a class by applying a logistic (sigmoid) function to a linear combination of input features. Despite its simplicity, logistic regression often provides

competitive results in text classification tasks, especially when combined with TF–IDF embeddings. In this work, the model was trained with balanced class weights to account for class imbalance in the dataset and was optimized with up to 1000 iterations to ensure convergence.

3.6.2 Support Vector Machine

Support Vector Machines (SVM) are powerful discriminative classifiers that aim to find the optimal hyperplane separating data points from different classes with the maximum margin [24]. In this study, a linear SVM was used via the `LinearSVC` implementation, which is particularly suitable for high-dimensional feature spaces such as those generated by TF–IDF. Class weights were also adjusted to mitigate the effects of class imbalance. SVMs are robust to overfitting in sparse representations, making them a strong candidate for text-based anomaly detection tasks.

3.6.3 Random Forest

Random Forest is an ensemble learning technique that builds multiple decision trees during training and outputs the majority vote of the individual trees [25]. Each tree is trained on a random subset of the data and features, which helps reduce overfitting and improve generalization. Although Random Forests are not typically the first choice for high-dimensional text embeddings, they provide an interpretable and diverse baseline, capable of capturing non-linear decision boundaries that linear models may miss.

3.7 Deep Learning Models

This section documents the modeling choices implemented for building deeplearning/SOTA solutions for the APT identification task. We focus on the learning algorithms, training procedures, inference strategies, and evaluation logic applied to log-level APT detection.

3.7.1 Transformer-Based Log Classification

Recent advances in Natural Language Processing (NLP) have been driven by Transformer architectures [26].. Unlike recurrent or convolutional networks, Transformers rely on a *self-attention* mechanism that allows each token in the input to dynamically focus on all other tokens in the sequence. This property makes Transformers particularly effective at capturing both short- and long-range dependencies, while also enabling efficient parallel training.

Bidirectional Encoder Representations from Transformers (BERT)

BERT [22] is one of the most influential Transformer-based models. Its novelty lies in its bidirectional encoding: when representing a word or subword, the model takes into account both its left and right context simultaneously. This is achieved through self-attention, which computes contextualized embeddings where the meaning of a token adapts to its surroundings. BERT is pre-trained on two tasks: *Masked Language Modeling* (predicting randomly masked tokens from context) and *Next Sentence Prediction* (predicting if two sentences follow each other). These pretraining objectives provide the

model with a strong general understanding of language. For classification tasks, a special token `[CLS]` is added at the beginning of each sequence, and its final representation is fed into a classification head.

DistilBERT

While BERT is powerful, its large size makes it computationally demanding. DistilBERT (Sanh et al., 2019) is a compressed version obtained through knowledge distillation, where a smaller model (the student) is trained to reproduce the behavior of the original BERT (the teacher). DistilBERT retains about 97% of BERT's performance while being significantly faster and smaller. This trade-off makes it highly suitable for log analysis, where large volumes of data must be processed efficiently without sacrificing too much accuracy. In this work, DistilBERT is fine-tuned for binary classification of log entries (benign vs. malicious). The implementation uses Hugging Face's `DistilBertForSequenceClassification` with `num_labels=2`. Tokenization is performed with `DistilBertTokenizerFast`, which applies WordPiece subword segmentation and maps text to integer IDs. Sequences are padded or truncated to a maximum length of 64 tokens to ensure consistent batch sizes. Fine-tuning is carried out using the cross-entropy loss, with Hugging Face's `Trainer` API managing batching, GPU acceleration, and optimizer scheduling. Both the classification head and the underlying transformer layers are updated during training, enabling the model to adapt pre-trained knowledge to the domain of Linux system logs.

SecBERT

General-purpose models like BERT and DistilBERT are trained on corpora such as Wikipedia and BookCorpus. Although powerful, these sources do not capture the highly technical and domain-specific vocabulary typical of system logs and cybersecurity text. To address this limitation, SecBERT was introduced as a variant of BERT pre-trained specifically on cybersecurity-related data, including log files, threat reports, and technical documents. This domain adaptation allows the model to better capture the semantics of command-line arguments, system calls, and malware signatures, which are rarely seen in general text but are crucial for intrusion detection.
SecBERT is fine-tuned for the same binary classification task as DistilBERT. It is initialized via Hugging Face's `AutoTokenizer` and `AutoModelForSequenceClassification`, with a maximum sequence length of 512 tokens to accommodate longer log entries. Training is performed with explicitly defined hyperparameters: a learning rate of `5e-6`, 10 epochs, batch sizes of 32 (training) and 64 (evaluation), and weight decay of `0.01` for regularization. These values are chosen to balance convergence speed and stability while minimizing the risk of overfitting or catastrophic forgetting of the cybersecurity-specific pretraining. Evaluation is done using `trainer.predict`, where model logits are converted into probabilities via softmax and mapped to class labels with `argmax`.

Why Using Transformers?

The motivation for using Transformer-based models is their ability to integrate both local and global information from log entries. Shallow models such as TF–IDF combined with Logistic Regression can only capture surface-level word frequencies, while Transformers generate contextualized embeddings that adapt token meaning depending on the

surrounding context. This is especially important in cybersecurity, where the same command may be benign in one context but malicious in another. By fine-tuning both a general-purpose (DistilBERT) and a domain-specific (SecBERT) model, we explore the trade-off between broad linguistic understanding and specialized domain knowledge in detecting Advanced Persistent T

3.7.2 Sequence-Aware Classifier: DeepLog-Inspired LSTM

Many security-relevant behaviors only emerge when analyzing *sequences* of logs over time. For example, a single command may appear benign in isolation, but when repeated in a certain temporal pattern it can indicate malicious activity. To address this, we experiment with a recurrent neural model inspired by DeepLog [27], which was one of the first approaches to apply sequence learning for log anomaly detection.
Recurrent Neural Networks (RNNs) are designed to process sequential data by maintaining a hidden state that evolves over time. However, classical RNNs suffer from the *vanishing gradient problem*, which makes it difficult for them to learn long-range dependencies. Long Short-Term Memory (LSTM) networks mitigate this issue by introducing gating mechanisms — the input gate, forget gate, and output gate — that control the flow of information through a memory cell. This architecture allows the network to decide which information to retain, which to update, and which to discard at each time step. As a result, LSTMs can capture both short-term fluctuations and longer-term temporal patterns, making them particularly suitable for modeling log sequences.

DeepLog Framework

DeepLog [27] proposed to model system logs as natural language sequences and to learn their temporal structure using LSTMs. By observing normal execution logs, the LSTM learns to predict the next log entry given a history window. Anomalies — such as malicious activity — are then detected when the observed log diverges from the expected prediction. This approach highlights the importance of sequence modeling in intrusion detection, as it enables the identification of suspicious behaviors that cannot be recognized by analyzing isolated log entries.

Practical Implementation

In our implementation, we adopt the general principle of DeepLog but adapt it for binary classification of log windows (benign vs. malicious). The pipeline is as follows:

Token Vocabulary. A custom `Vocab` class maps each distinct log token to an integer ID, producing compact numerical sequences. This ensures that the LSTM can process logs as discrete time-series rather than free text.

Sliding-Window Encoding. Log streams are segmented into overlapping windows of fixed size $w = 10$. Each training sample thus consists of a short sequence of 10 consecutive log tokens. The associated label indicates whether the last log in the window is benign or malicious. This encoding allows the model to learn temporal correlations between neighboring logs.

Dataset and Batching. Sequences and labels are wrapped into a PyTorch `Dataset` and fed into `DataLoader`s with `batch_size=128`. Training batches are shuffled to promote generalization, while evaluation batches preserve chronological order for consistency.

Model Architecture. The `DeepLogClassifier` is a lightweight sequence model:

- **Embedding Layer:** `nn.Embedding(vocab_size, 64)` maps discrete token IDs into dense 64-dimensional vectors.
- **LSTM Encoder:** a 2-layer LSTM with hidden size 64 processes the window of embeddings. Only the final hidden state is retained as a representation of the entire window.
- **Classification Head:** a linear layer maps the hidden state to a scalar, followed by a Sigmoid activation that outputs the probability of the sequence being malicious.

Training Setup. The model is trained for 20 epochs using Adam optimization with a learning rate of 1×10^{-4}. The objective function is binary cross-entropy (`nn.BCELoss`), which penalizes the divergence between predicted probabilities and true binary labels. Each training iteration consists of moving a batch to the GPU (if available), performing a forward pass, computing the loss, backpropagating gradients, and updating parameters. Average loss per epoch is monitored to track convergence.

Per-Log Inference. At test time, the same sliding window is applied to the log stream. Each window yields a probability score, which is thresholded at 0.5 to classify the final log of the window as benign or malicious. Predictions are aligned with ground-truth labels by shifting indices by $w - 1$, enabling accurate per-log evaluation of detection performance.

3.8 Proposed Solution: Ensemble of Tactic-Specific BERT Classifiers

3.8.1 General Idea

A key observation from the Linux APT 2024 dataset is that malicious activity is not homogeneous: attacks are structured into distinct *tactics* according to the MITRE ATT&CK framework, such as Initial Access, Privilege Escalation, Defense Evasion, or Command and Control. Each tactic exhibits characteristic log patterns that may differ significantly from one another. A single global classifier, trained to cover all tactics at once, risks underfitting rare tactics or being dominated by high-frequency ones.
To address this, we propose an **ensemble of fine-tuned BERT classifiers**, each trained on log entries associated with a specific MITRE tactic. This modular design allows each model to specialize in the linguistic and structural features that are most discriminative for its assigned tactic, while the ensemble as a whole provides robust coverage of the complete APT lifecycle. At inference time, predictions from the tactic-specific models are aggregated to decide whether a log should be considered malicious.

3.8.2 Why an Ensemble is Suitable

The ensemble approach brings several advantages:

- **Specialization:** Different attack stages leave very different traces in logs. For example, privilege escalation often involves suspicious system calls, while command and control may involve repeated outbound connections. By training one model per tactic, each classifier can focus on its domain-specific patterns.
- **Improved Recall:** Rare tactics are less likely to be overshadowed by frequent ones, since they are handled by dedicated models. This reduces the risk of missing subtle but important malicious behaviors.
- **Flexibility:** The modular structure allows easy retraining or replacement of specific tactic models if new data becomes available, without affecting the rest of the system.
- **Parallelization:** Since each classifier is trained independently on its corresponding tactic, the training process can be parallelized across multiple GPUs or computing nodes. This not only reduces total training time but also makes the approach scalable when extending the ensemble with new tactic-specific models.

3.8.3 Implementation Details

The ensemble is built around `DistilBERT`, a lighter version of BERT that retains most of its representational power while being computationally more efficient. The pipeline proceeds as follows:

Data Partitioning by Tactic. The malicious subset of the dataset is grouped according to its associated MITRE tactic. High-volume tactics (e.g., Defense Evasion, Initial Access, Privilege Escalation, Discovery) are allocated their own dedicated classifiers. Less frequent tactics are either combined into a group model or omitted if data is too scarce. Benign logs are mixed into each tactic dataset to provide balanced positive/negative training examples.

Model Training. For each tactic dataset, the following setup is applied:

- Tokenization via `DistilBertTokenizerFast`, with padding and truncation to a maximum sequence length of 64 tokens.
- A `DistilBertForSequenceClassification` model with two output labels (benign/malicious).
- Training parameters: batch size of 32 (training) and 64 (evaluation), three epochs of fine-tuning, and standard cross-entropy loss. Models are trained independently, one per tactic.

This results in a collection of tactic-specific classifiers, each stored and reloaded for evaluation.

Hyperparameter Configuration. Each tactic-specific classifier was fine-tuned with a set of carefully chosen hyperparameters to balance efficiency and performance. The sequence length was capped at 64 tokens, which is sufficient to cover the majority of log entries while avoiding excessive padding. Training was performed for `3 epochs`, a standard compromise that allows the model to adapt to the log domain without overfitting on the relatively small per-tactic datasets. Batch sizes were set to `32` for training and `64` for evaluation, making effective use of GPU memory while ensuring stable gradient updates. The optimizer and learning rate scheduling were handled automatically by the Hugging Face `Trainer`, with the default AdamW optimizer and weight decay for regularization. The loss function was binary cross-entropy implemented via the standard cross-entropy objective for two classes. These hyperparameters provide a lightweight yet robust setup, enabling multiple tactic-specific models to be trained in parallel without prohibitive computational costs.

Ensemble Inference. During testing, every log entry is passed through all tactic-specific models. Each model outputs both a class prediction and a probability score. Predictions are aggregated with a **max-pooling strategy**: the ensemble assigns a malicious label if any tactic model strongly flags the log as malicious. This conservative approach prioritizes recall, ensuring that an attack is not missed simply because it falls outside the focus of a single classifier.

3.9 Evaluation Metrics

To assess the performance of the proposed ensemble of fine-tuned BERT models for Advanced Persistent Threat (APT) detection, two complementary metrics are used: **Receiver Operating Characteristic (ROC) Curve** and the **Classification Report** (precision, recall, F1-score). Both provide different but valuable insights into the detection capability of the models.

3.9.1 ROC Curve

The ROC curve is a graphical representation of the trade-off between the **True Positive Rate (TPR, or recall)** and the **False Positive Rate (FPR)** across different classification thresholds. The **Area Under the Curve (AUC)** is often used as a summary statistic, where a value closer to 1 indicates a strong ability to distinguish between malicious and benign logs.

For APT detection, the ROC curve is particularly useful because:

- It evaluates performance independently of any single decision threshold.
- It highlights how well the system can rank malicious events higher than benign ones.
- Since APT attacks are rare but critical, the ability to reduce false negatives (missed attacks) without excessively increasing false positives is crucial, and this trade-off is visible in the ROC curve.

3.9.2 Classification Report

The classification report provides a breakdown of **precision, recall, and F1-score** per class:

- **Precision**: Of the logs predicted as malicious, how many were truly malicious. High precision means fewer false alarms.
- **Recall**: Of all truly malicious logs, how many were correctly identified. High recall means fewer missed attacks.
- **F1-score**: The harmonic mean of precision and recall, balancing the trade-off.

For APT detection, the classification report is essential because it directly quantifies the **impact of misclassifications**. A missed detection (false negative) can mean failing to stop a breach, while too many false positives (low precision) can overwhelm analysts with alerts.

3.9.3 Which to Focus On

While both metrics are valuable, the **classification report should be emphasized more in the context of APT detection**. This is because operational security teams need actionable insight into:

- How often the system misses malicious logs (recall).
- How trustworthy the alerts are (precision).

The ROC curve remains useful to compare models and evaluate overall discriminative ability, but the classification report better reflects the practical requirements of intrusion detection, where **recall is usually prioritized** to ensure threats are not overlooked, even at the cost of some false positives.

Chapter 4

Results and Discussion

4.1 Results on Shallow Models

4.1.1 Experimental Setup

Three classical machine learning classifiers were evaluated as shallow baselines: **Logistic Regression (LR)**, **Linear SVM (LinearSVC)**, and **Random Forest (RF)**. Models were trained under two different feature representations:

- **TF–IDF features**: unigram and bigram representation (max features = 100, English stopword removal).
- **BERT embeddings**: dense vector representations computed from the log text.

All models were trained with `class_weight=balanced` (or equivalent) to mitigate class imbalance. Two evaluation protocols were used:

1. **Random 80/20 split:** stratified hold-out (positives $\approx 20\%$ in the test set).
2. **Temporal split:** first 80% of data for training, last 20% for testing (positives $\approx 1.9\%$ in the test set).

The random split simulates in-distribution performance, while the temporal split is a closer approximation of real deployment, where distribution shift and label imbalance are prominent.

4.1.2 Random 80/20 Split

Under random sampling, all models performed strongly, with very similar outcomes across classifiers. The feature representation was more influential than the choice of classifier.

TF–IDF features. Logistic Regression, SVM, and Random Forest all achieved a ROC–AUC of ~ 0.915. Accuracy was approximately 0.76, with macro–F1 scores around 0.72. The models exhibited a strong preference for *recall* of the positive class (~ 0.99), at the expense of precision (~ 0.45).

BERT embeddings. BERT-based features consistently improved ROC–AUC to $\sim$ 0.935, representing a gain of ≈ 0.02 compared to TF–IDF. Accuracy was 0.77, and macro–F1 increased slightly to 0.73. The positive class maintained very high recall (0.98–0.99) with modest precision (0.46–0.47).

Summary. In the random split, all three classifiers were essentially tied in performance. Feature representation dominated: BERT embeddings consistently outperformed TF–IDF by ~ 0.02 ROC–AUC. The weighting scheme yielded recall-oriented models, which may be desirable in domains where missing positives is costlier than false alarms.

4.1.3 Temporal Split

Performance degraded substantially when models were trained on early data and tested on later data, reflecting temporal drift and stronger imbalance.

TF–IDF features. All three models collapsed to near-chance performance with ROC–AUC scores between 0.551 and 0.555. Accuracy (~ 0.74) was dominated by the majority class. Positive recall fell to ~ 0.36, with very poor precision (~ 0.03).

BERT embeddings.

- Logistic Regression was the most robust, reaching ROC–AUC = 0.666, with accuracy 0.75 and positive recall 0.36.
- Linear SVM degraded more severely (ROC–AUC = 0.588).
- Random Forest collapsed almost entirely, producing near-trivial predictions: accuracy 0.98 (by predicting almost all samples as negative), but positive recall only 0.02.

Summary. Under temporal evaluation, all models degraded, but Logistic Regression with BERT embeddings was comparatively the least affected. Random Forest was particularly brittle, effectively reducing to a majority-class predictor.

4.1.4 Comparative Analysis

The results highlight three main findings:

1. **Feature representation outweighs model choice in-distribution.** BERT embeddings consistently improve ROC–AUC compared to TF–IDF.
2. **Temporal generalization is the key challenge.** AUC drops from ~ 0.935 (random split) to as low as 0.59 (temporal split). This indicates substantial covariate and label shift in the dataset.
3. **Accuracy is misleading under imbalance.** In the temporal split, an "always negative" baseline already achieves 98.1% accuracy, underscoring the importance of ROC–AUC and precision–recall metrics.

4.1.5 Cumulative Tables

Table 4.1: Performance on the random 80/20 split (in-distribution).

Model + Features	ROC–AUC	Accuracy	Macro-F1	Pos. Recall
LR + TF–IDF	0.915	0.76	0.72	0.99
SVM + TF–IDF	0.915	0.76	0.72	0.99
RF + TF–IDF	0.916	0.76	0.72	0.99
LR + BERT	0.935	0.77	0.73	0.98
SVM + BERT	0.933	0.77	0.73	0.99
RF + BERT	0.934	0.77	0.73	0.98

Table 4.2: Performance on the temporal split (out-of-distribution).

Model + Features	ROC–AUC	Accuracy	Macro-F1	Pos. Recall
LR + TF–IDF	0.555	0.74	0.45	0.36
SVM + TF–IDF	0.551	0.74	0.45	0.33
RF + TF–IDF	0.552	0.74	0.45	0.36
LR + BERT	0.666	0.75	0.45	0.36
SVM + BERT	0.588	0.75	0.45	0.33
RF + BERT	0.614	0.98	0.45	0.02

Key Numbers. The best in-distribution result was achieved by LR + BERT (ROC–AUC = 0.935), while the best temporal performance was also obtained by LR + BERT (ROC–AUC = 0.666). Random Forest, in particular, exhibited extreme sensitivity to temporal drift, despite appearing competitive under random splits.

4.2 Results on Deep Learning Models

4.2.1 Experimental Setup

Three deep learning models were evaluated on the Lunex APT 2024 dataset: **fine-tuned BERT**, **fine-tuned SecBERT**, and **DeepLog** (LSTM-based sequence model). As with shallow baselines, two evaluation protocols were applied:

1. **Random 80/20 split:** stratified hold-out, positives $\approx 20\%$ of the test set.
2. **Temporal 80/20 split:** first 80% of the logs for training, last 20% for testing, positives $\approx 1.9\%$ of the test set.

The random split reflects in-distribution performance, while the temporal split better approximates real-world deployment where class imbalance and drift are present.

4.2.2 Random 80/20 Split

All three models performed strongly under random sampling, surpassing the shallow baselines in recall-precision balance and macro–F1.

BERT fine-tuned. Fine-tuned BERT achieved the highest overall ROC–AUC (0.935), matching the shallow baselines in AUC but offering stronger per-class performance. The positive class was identified with high precision (1.00) but recall was moderate (0.56), resulting in an F1 of 0.72. Accuracy was 0.91, and macro–F1 averaged 0.83.

SecBERT fine-tuned. The domain-adapted SecBERT model reached ROC–AUC 0.783. Precision was high (0.99) for positives, though recall remained similar to BERT (0.57). Overall accuracy (0.91) and macro–F1 (0.83) were nearly identical to BERT, suggesting that in-distribution performance is less sensitive to pretraining domain differences.

DeepLog. The sequence-based DeepLog model achieved ROC–AUC 0.831, lower than fine-tuned BERT but still outperforming shallow models. Positive recall (0.67) exceeded that of BERT and SecBERT, though precision (0.96) was slightly lower. Macro–F1 reached 0.87, the highest among deep learning models under random splits.

Summary. On random splits, deep learning methods achieved strong results, with fine-tuned BERT leading in ROC–AUC and DeepLog providing the best recall and macro–F1. Compared to shallow baselines, deep models demonstrated a more balanced tradeoff between recall and precision.

4.2.3 Temporal Split

When evaluated under temporal drift, all models degraded significantly, though differences emerged in robustness.

BERT fine-tuned. Performance dropped to ROC–AUC 0.680. While overall accuracy remained high (0.98), this was due to strong majority-class prediction. Positive recall collapsed to 0.02, producing an F1 of only 0.03. This highlights severe brittleness to temporal distribution shift.

SecBERT fine-tuned. SecBERT was even more affected, with ROC–AUC 0.515. Positive recall fell to 0.03, yielding trivial F1 scores despite accuracy of 0.98. The model effectively reduced to a negative-class predictor under drift.

DeepLog. DeepLog retained somewhat better robustness, with ROC–AUC 0.528. Positive recall was 0.06, slightly above BERT and SecBERT, though still insufficient for deployment. As with other models, accuracy (0.98) was inflated by class imbalance.

Summary. All deep learning models struggled with temporal generalization. Despite strong random-split performance, their ability to detect positives in chronologically shifted data collapsed. DeepLog retained marginally better recall than BERT or SecBERT, but overall performance was unsatisfactory.

4.2.4 Comparative Analysis

Three main findings emerge from the deep learning evaluation:

1. **Deep models improve in-distribution balance.** Fine-tuned BERT, SecBERT, and DeepLog achieve higher precision–recall tradeoffs than shallow baselines in random splits.

2. **Temporal robustness is severely lacking.** ROC–AUC plummets from 0.935 (BERT, random split) to 0.680 (BERT, temporal split), and as low as 0.515 (SecBERT). DeepLog also drops to 0.528, underscoring the difficulty of detecting APT events under temporal drift.

3. **Accuracy alone is misleading.** All models maintained ~ 0.98 accuracy in the temporal split by predicting almost all samples as negative, emphasizing the importance of AUC and recall metrics in imbalanced, real-world settings.

4.2.5 Cumulative Tables

Table 4.3: Performance of deep learning models on the random 80/20 split (in-distribution).

Model	ROC–AUC	Accuracy	Macro-F1	Pos. Recall
BERT (fine-tuned)	0.935	0.91	0.83	0.56
SecBERT (fine-tuned)	0.783	0.91	0.83	0.57
DeepLog (LSTM)	0.831	0.93	0.87	0.67

Table 4.4: Performance of deep learning models on the temporal split (out-of-distribution).

Model	ROC–AUC	Accuracy	Macro-F1	Pos. Recall
BERT (fine-tuned)	0.680	0.98	0.51	0.02
SecBERT (fine-tuned)	0.515	0.98	0.52	0.03
DeepLog (LSTM)	0.528	0.98	0.54	0.06

Key Numbers. The strongest in-distribution performance was obtained by fine-tuned BERT (ROC–AUC = 0.935), while DeepLog achieved the best recall (0.67) and macro–F1 (0.87). In temporal evaluation, all models deteriorated severely, with fine-tuned BERT scoring highest (ROC–AUC = 0.680), though recall dropped to 0.02.

4.3 Proposed solution: BERT Ensamble

4.3.1 Experimental Setup

In addition to individual deep learning classifiers, we evaluated an **ensemble of fine-tuned BERT heads**, where each head was trained to specialize on a subset of ATT&CK tactics (e.g., Defense Evasion, Initial Access, Persistence, Execution, Privilege Escalation).

During inference, the ensemble aggregates predictions across these specialized heads, enabling tactic-aware detection. To ensure that each ensemble component was properly optimized, we monitored the per-tactic training loss curves. All heads exhibited smooth convergence without signs of oscillation or divergence. Importantly, the loss curves decayed steadily and flattened toward the later epochs, without evidence of overfitting (no sudden increase in loss or instability). This indicates that the ensemble models were trained sufficiently and reached a stable minimum, making their results reliable for evaluation.

Figure 4.1: Training loss convergence for the BERT ensemble models across tactics. The consistent downward trends without overfitting confirm that the tactic-specific heads are well-trained and stable.

As with previous experiments, two evaluation protocols were applied:

1. **Random 80/20 split:** stratified hold-out with $\approx 20\%$ positives.
2. **Temporal 80/20 split:** first 80% of logs for training, last 20% for testing, with positives $\approx 1.9\%$.

4.3.2 Random 80/20 Split

The ensemble achieved the strongest in-distribution performance across all tested methods.

Performance. ROC–AUC reached 0.955, the highest among all evaluated models. Accuracy was 0.97, and macro–F1 reached 0.82. The ensemble achieved high precision on the positive class (0.99) but recall was moderate (0.49), leading to an F1 of 0.66. Compared to single BERT models, the ensemble improved ROC–AUC by approximately 0.02, though recall remained similar.

Why better? The gain can be attributed to *task specialization*: by training separate heads on tactic-specific subsets, the ensemble captured finer-grained attack patterns that a single model might overlook. This specialization improves discrimination in-distribution, particularly for subtle tactic signals, leading to higher AUC and precision.

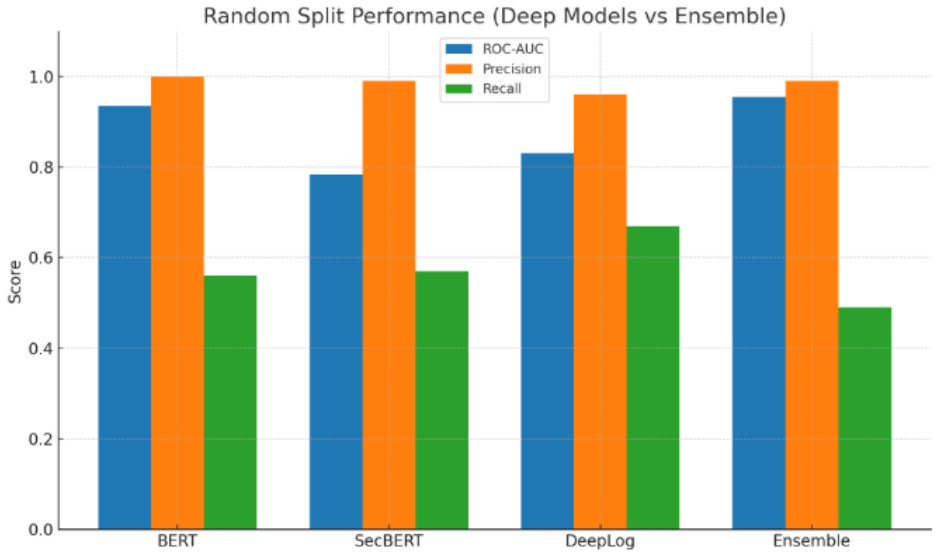

Figure 4.2: Comparison of ROC–AUC, Precision, and Recall for deep learning models and the BERT ensemble under the random 80/20 split.

Summary. By leveraging tactic-specific heads, the ensemble produced a strong precision-oriented model with superior ROC–AUC and balanced macro–F1. Its high accuracy and robust detection under random splits make it the best-performing in-distribution method.

4.3.3 Temporal Split

Under temporal drift, ensemble performance degraded sharply, similar to other deep models.

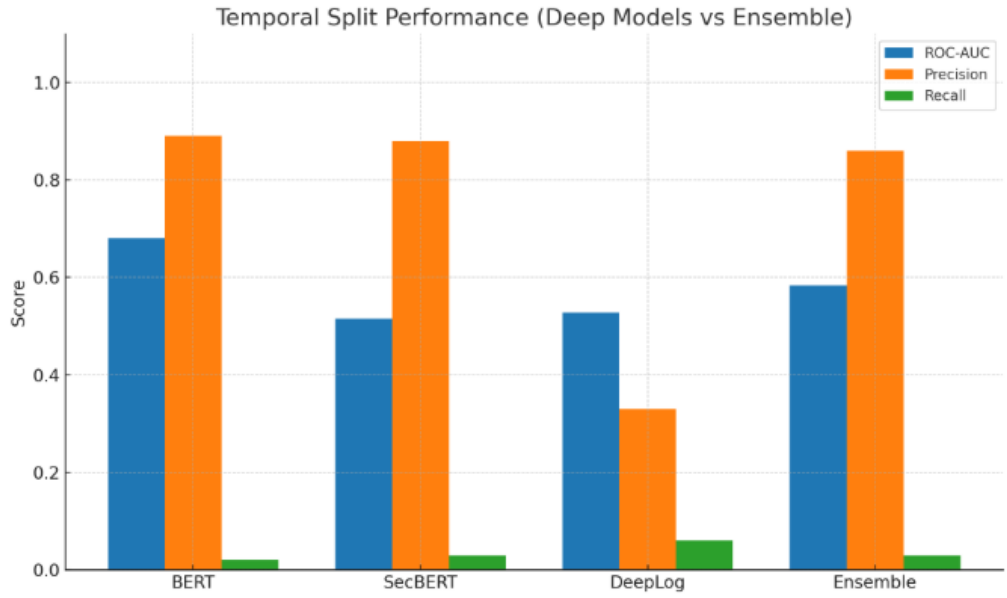

Figure 4.3: Comparison of ROC–AUC, Precision, and Recall for deep learning models and the BERT ensemble under the temporal split.

Performance. ROC–AUC dropped to 0.584, close to chance level. Accuracy remained artificially high (0.98) due to majority-class dominance. Positive recall collapsed to 0.03, producing an F1 of only 0.05, despite precision remaining high (0.86). This indicates that the ensemble overfit to training distribution and struggled with temporal generalization.

Why worse? The same *specialization* that boosted in-distribution performance became a liability under drift: tactic-specific heads overfit to the temporal characteristics of their training window. As the log distribution shifted, these specialized detectors failed to generalize, collapsing to majority-class predictions.

Summary. Despite strong in-distribution results, the ensemble was not robust under chronological testing. Like fine-tuned BERT and SecBERT, it reduced to a majority-class predictor with very limited recall.

4.3.4 Comparative Analysis

The ensemble results yield three main insights:

1. **Best in-distribution performance.** With ROC–AUC 0.955, the ensemble outperformed both shallow and deep single-model baselines under random splits.
2. **Limited temporal robustness.** Temporal AUC dropped to 0.584, with recall as low as 0.03, showing no advantage over single BERT models under drift.
3. **Specialization improves static performance, but harms robustness.** Tactic-specific heads enhance feature sensitivity and precision in-distribution, but this comes at the cost of adaptability. When attack behavior evolves, specialized models fail more severely than general ones.

4.3.5 Cumulative Tables

Table 4.5: Performance of ensemble models on the random 80/20 split (in-distribution).

Model	ROC–AUC	Accuracy	Macro-F1	Pos. Recall
BERT Ensemble (tactic heads)	0.955	0.97	0.82	0.49

Table 4.6: Performance of ensemble models on the temporal split (out-of-distribution).

Model	ROC–AUC	Accuracy	Macro-F1	Pos. Recall
BERT Ensemble (tactic heads)	0.584	0.98	0.52	0.03

Key Numbers. The ensemble achieved the best in-distribution score (ROC–AUC = 0.955) but collapsed under temporal testing (ROC–AUC = 0.584, recall 0.03). In other words: the ensemble was **better in static evaluation** due to tactic specialization, but **worse in realistic temporal evaluation** because over-specialization reduced adaptability.

4.3.6 Analysis of Missed Malicious Records

Although the ensemble of BERT heads achieved strong performance overall, a detailed error analysis was conducted to better understand the nature of the missed malicious records (false negatives). Two complementary perspectives were used: a linguistic inspection of the missed logs and a statistical view of the prediction probabilities.

Lexical signal weakness. Figure 4.4 compares the word clouds of detected malicious logs against the ones that were missed. It is evident that the missed malicious logs lack distinctive attacker-related tokens or semantic cues. Instead, they are dominated by sparse and generic tokens (e.g., identifiers, placeholders, or system-generated strings), which do not provide sufficient discriminative signal for the BERT heads. By contrast, the detected malicious logs are enriched with security-relevant terms (*cmd*, *java*, *memberaccess*, etc.), making them easier for the models to classify. This indicates that the false negatives correspond to attacks expressed in subtle or obfuscated forms, where the malicious intent is not strongly reflected in the text.

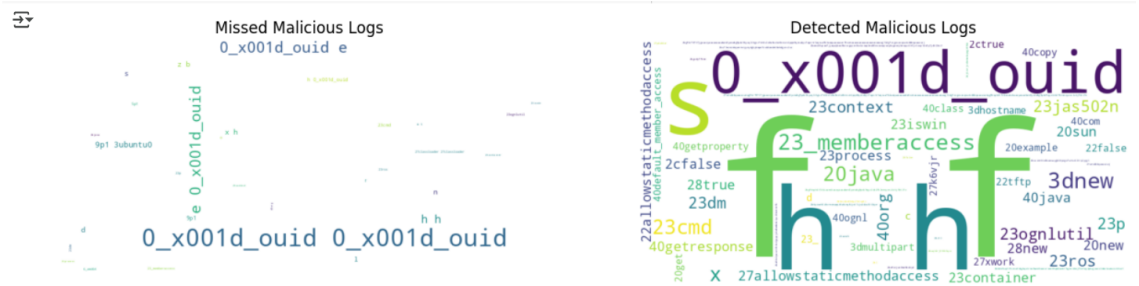

Figure 4.4: Comparison between missed malicious logs (left) and detected malicious logs (right). Missed records are lexically sparse and dominated by generic identifiers, while detected records include richer attacker-related patterns.

Confidence distribution. Figure 4.5 presents the probability distribution of the model's predictions for malicious samples. Here, we clearly see that missed malicious records concentrate at very low confidence levels (close to 0), meaning that the ensemble was not uncertain but rather confidently misclassified them as benign. This suggests that the missed cases do not only lack strong attacker signals but may also overlap lexically or structurally with benign log patterns, causing the classifier to treat them as normal behavior.

Conclusion. The error analysis demonstrates that the false negatives stem from a combination of (i) weak lexical or semantic cues in the missed logs, and (ii) structural similarity to benign behavior. This highlights the need for complementary detection strategies (e.g., context-aware features or sequence-level correlations) to reduce the blind spots of text-only classifiers.

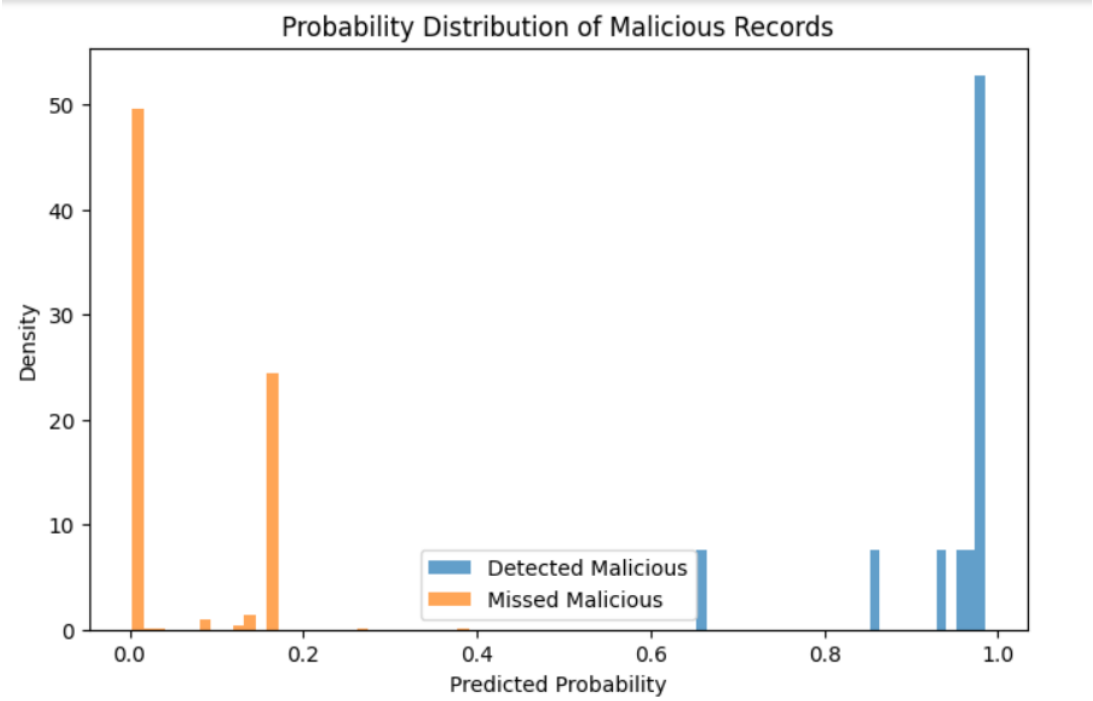

Figure 4.5: Probability distribution of malicious records. Missed malicious logs (orange) cluster near zero, indicating confident misclassification, while detected logs (blue) concentrate near one.

Chapter 5

Conclusions and Future Work

5.1 Conclusions

This thesis explored shallow models, deep learning classifiers, and a tactic-aware BERT ensemble for malicious log detection. The results reveal both the potential and the limitations of these approaches, shaped strongly by the evaluation protocol.

Shallow models, although efficient and simple, captured mostly surface-level patterns. With BERT embeddings they reached competitive results under random splits, but their generalization collapsed once data was ordered chronologically, showing their limited ability to adapt to new and unseen attack traces.

Deep learning classifiers provided a stronger foundation. Among them, fine-tuned BERT stood out as the most robust, maintaining reasonable performance even under chronological evaluation, while other architectures such as DeepLog or SecBERT proved less reliable. These results confirm that deep contextual representations are better suited to handle the complexity of log data than shallow statistical methods.

The **ensemble of tactic-specific BERT heads** added an interesting layer of specialization. By dividing the detection task across tactics, the ensemble showed strong ranking ability in the random split, confirming that tactic-aware decomposition helps the model capture certain nuanced behaviors. At the same time, its limitations became clear: recall was lower than in some individual models, and chronological evaluation exposed weaknesses similar to those of other approaches. This shows that while the ensemble is not a universal solution, it is a promising strategy to enrich detection pipelines by complementing single-model approaches.

A deeper look at the **missed malicious logs** helps explain these results. The word cloud analysis highlighted that many false negatives were dominated by obfuscated tokens, identifiers, or strings with little semantic content. These features offer the models almost no discriminative signal, making the logs appear deceptively similar to benign activity. The probability distribution analysis further confirmed that such logs were often assigned very low malicious probabilities — the models were not merely uncertain, but confidently wrong. This suggests that the hardest cases are not random errors, but systematic blind spots caused by the nature of adversarial obfuscation.

The central lesson of this work is the dramatic gap between random and chronological evaluations. While random splits can paint an overly optimistic picture of model performance, chronological splits reveal the real challenge: adapting to temporal drift and evolving attack techniques. Accuracy alone, inflated by the majority class, is misleading; recall and ROC-based metrics give a more faithful view of robustness.

In conclusion, the ensemble demonstrates the promise of tactic-aware learning, but also underscores that the hardest part of malicious log detection lies in coping with evolving, obfuscated patterns that deliberately mimic benign activity. Future progress will depend not only on building stronger models, but also on designing systems that can adapt continuously and remain effective against the ever-changing landscape of adversarial behavior.

5.2 Future Work

The findings of this thesis suggest several directions for future research.
First, the significant performance drop observed under the chronological split highlights the need for models that can adapt to temporal drift. Future work should therefore investigate **incremental or continual learning strategies**, where models are updated regularly as new data becomes available, reducing the risk of concept drift over time.
Second, the error analysis revealed that many missed malicious logs are characterized by obfuscation, identifiers, or strings with little semantic content. Addressing this limitation will likely require **richer feature representations** that go beyond token semantics, for example by incorporating structural properties of logs, execution context, or frequency-based signals.
Third, the ensemble of tactic-specific heads demonstrated the value of specialization but also showed limitations in recall. A natural extension would be to explore **hybrid ensembles**, where tactic-specific models are combined with strong general-purpose classifiers. This could allow the system to balance specialization with broader coverage, reducing systematic blind spots.
Finally, while this work focused on supervised learning, future research could explore **semi-supervised or unsupervised methods**, which may leverage large amounts of unlabeled log data. This would be especially relevant in security contexts, where obtaining reliable labels is difficult and attackers constantly introduce novel behaviors.
Overall, future work should continue to pursue models that are both **specialized and adaptive**, combining the strengths of tactic-aware learning with mechanisms to remain robust against the evolving nature of adversarial activity.

Bibliography

[1] R. Ross, G. McGraw, S. Youst, D. Bodeau, and R. McQuaid, "Managing information security risk: Organization, mission, and information system view," Tech. Rep. SP 800-39, National Institute of Standards and Technology (NIST), 2011. `https://doi.org/10.6028/NIST.SP.800-39`.

[2] R. Bisht, "Phases of advanced persistent threat (apt) lifecycle," *InfosecTrain Blog*, 2024. Defines a seven-phase APT lifecycle framework.

[3] S. Bodmer, M. Kilger, G. Carpenter, and J. Jones, *Reverse Deception: Organized Cyber Threat Counter-Exploitation*. McGraw-Hill Osborne Media, 2012. Includes Titan Rain case as early APT.

[4] N. Thornburgh, "The invasion of the chinese cyberspies (and the man who tried to stop them)," *Wired*, 2010. Details Operation Aurora / Hydraq targeting Google and others.

[5] K. Zetter, "Countdown to zero day: Stuxnet and the launch of the world's first digital weapon," 2011. Comprehensive account of Stuxnet attacks and impact.

[6] D. Goodin, "Rsa breach linked to compromised tokens in securid two-factor," *The Register*, 2011. Covers the RSA SecureID phishing-based APT incident.

[7] D. E. Sanger, "Inside the takedown of the alleged €1billion cyber bank robbery," *Wired*, 2018. Carbanak APT campaign overview and financial impact.

[8] Kaspersky ICS CERT, "Apt attacks on industrial organizations in h12021," tech. rep., Kaspersky ICS CERT, 2021. Includes analysis of SolarWinds SUNBURST supply-chain APT case.

[9] Krishnapriya and Singh, "A comprehensive survey on advanced persistent threat (apt) detection techniques," *Computers, Materials & Continua*, vol. 80, no. 2, pp. 2675–2719, 2024. Survey reviews detection methods and highlights primary limitations.

[10] S. Krishnapriya and S. Singh, "A comprehensive survey on advanced persistent threat (apt) detection techniques," *Computers & Security*, vol. 141, p. 103381, 2024.

[11] D. Salim, R. Hassan, and A. Zainal, "A systematic literature review for apt detection and effective cyber situational awareness (ecsa) conceptual model," *Sensors*, vol. 23, no. 13, p. 5966, 2023.

[12] V. Vinoth Kumar, R. Prabha, and V. Thirumal, "Sr2apt: A detection and strategic alert response model against multistage advanced persistent threat attacks," *Security and Privacy*, vol. 6, no. 2, p. e6802359, 2023.

[13] K. Wen, J. Li, X. Sun, J. Yan, X. Zhang, and M. Xu, "Aptshield: A stable, efficient and real-time advanced persistent threat detection system for linux hosts," 2021.

[14] X. Han, K. Khandelwal, X. Du, Z. Chen, and R. Sekar, "Unicorn: Runtime provenance-based detector for advanced persistent threats," in *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2020.

[15] M. Rahman, S. Wang, K. Kalkan, and J. Zhan, "Apt-llm: Embedding-based anomaly detection of cyber advanced persistent threats using large language models," *arXiv preprint arXiv:2502.09385*, 2025.

[16] L. Zhang, R. Sun, Y. Chen, and M. Xu, "Logshield: A transformer-based apt detection system leveraging self-attention," *arXiv preprint arXiv:2311.05733*, 2023.

[17] M. Rahman, S. Wang, K. Kalkan, and J. Zhan, "Shield: Apt detection and intelligent explanation using large language models," *arXiv preprint arXiv:2502.02342*, 2025.

[18] C. I. for Cybersecurity, "Cic-apt2021 dataset," 2021. Accessed: 2025-08-08.

[19] D. A. R. P. A. (DARPA), "Darpa operationally transparent cyber dataset (optc)," 2021. Accessed: 2025-08-08.

[20] C. T. I. Lab, "Linux apt 2024 dataset," 2024. Accessed: 2025-08-08.

[21] K. S. Jones, "A statistical interpretation of term specificity and its application in retrieval," *Journal of Documentation*, vol. 28, no. 1, pp. 11–21, 1972.

[22] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of NAACL-HLT*, 2019.

[23] D. W. Hosmer, S. Lemeshow, and R. X. Sturdivant, *Applied Logistic Regression*. John Wiley & Sons, 2013.

[24] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

[25] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[26] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems (NeurIPS)*, pp. 5998–6008, 2017.

[27] M. Du, F. T. Li, G. Zheng, and V. Srikumar, "Deeplog: Anomaly detection and diagnosis from system logs through deep learning," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pp. 1285–1298, 2017.