KnowML: Improving Generalization of ML-NIDS with Attack Knowledge Graphs

Xin Fan Guo[†‡], Xinran Zheng[‡], Albert Merono Penuela[⋆],
Lorenzo Cavallaro[‡], Sergio Maffeis[†], Fabio Pierazzi[‡]
[†]*Imperial College London,* [⋆]*King's College London,* [‡]*University College London*

***Abstract*—Anomaly-based ML-NIDS (A-NIDS) model normal network behavior from benign data and classify deviations from this baseline as anomalies, theoretically enabling the detection of evolving attack variants without labeled attack data. The ability of A-NIDS to generalize critically depends on the quality of the feature space representing network behavior. However, the requirement for feature spaces that encode attack-relevant semantics has received little attention and remains poorly understood. As a consequence, these systems still struggle to meet practical operational constraints (low false positive rates without compromising detection performance and generalization to attack variants). We identify two limitations in the current feature spaces. First, *Out-of-Dimension Blindness*, where features do not capture essential attack mechanism properties. Second, *Attack Strategy Aggregation Failure*, where features cannot encode composite attack behaviors. Moreover, we demonstrate that two SotA data-driven generalization frameworks (based on incremental and contrastive learning) cannot compensate for these feature-level shortcomings.**

To bridge this gap, we present KnowML, a framework that encodes attack domain knowledge directly into the feature space. For each attack family, our method employs LLMs to construct a corresponding Knowledge Graph (KG) from attack implementations. Symbolic reasoning is then applied over the KG to enumerate potential attack strategies and their compositions. The resulting *Knowledge-Augmented Feature Space* enables effective generalization even when trained exclusively on benign traffic, a capability beyond current approaches. Systematic empirical evaluations show that KnowML achieves up to 99% detection rates while maintaining false positive rates at or below 0.0137%, substantially outperforming contemporary feature-based baselines across diverse attack variants.

***Index Terms*—Anomaly Detection, Network Intrusion Detection, Knowledge Graphs, Feature Space, LLMs.**

1. Introduction

Among Machine Learning-based Network Intrusion Detection Systems (ML-NIDS), Anomaly-based ones (A-NIDS) have been widely studied for their potential to detect both evolving and previously-unseen attack variants without requiring extensive labeling to improve generalization [13], [101], [102]. Recent studies report strong performance on benchmark datasets, demonstrating progress in detection accuracy and generalization [80], [84], [101], [102], positioning A-NIDS as promising candidates for practical deployment.

Practicality, however, imposes two demanding constraints. First, a low False Positive Rate (FPR) is essential, unlike domains such as image classification: typical network traffic rates (10,000+ pps [33]) mean that even a 1% FPR yields an impractical volume of false alarms [15], [17], [85]. Second, A-NIDS must effectively detect both seen and unseen attack variants. Yet contemporary work often operates under permissive FPR thresholds (e.g., 5% in [100]) and evaluations are typically conducted on homogeneous datasets [38], which do not fully capture the diversity of attack strategies encountered in practice [71].

In this paper, we start by investigating whether current A-NIDS can satisfy the aforementioned operational constraints of low FPRs while detecting sophisticated attack variants. When we discuss A-NIDS, we focus on two building blocks of such systems: a feature space that represents traffic characteristics, and a machine learning model that performs anomaly-based detection. Within this setting, our initial empirical analysis indicates that the choice of feature space has a stronger influence on detection effectiveness than the choice of model architecture. This observation motivates a deeper examination of why existing feature spaces may fall short.

We hypothesize that the underlying cause is a fundamental *knowledge gap* that manifests in two forms, *Out-of-Dimension Blindness*, where features directly related to attack mechanisms are missing, and *Attack Strategy Aggregation Failure*, where features are not aggregated in a way that captures composite attack strategies. These gaps arise because current systems are built on incomplete and heuristic assumptions about attacker strategies. Unlike conventional ML applications such as image classification, where models operate on complete input data, that is, all pixels in an image (distorted or not) and learn representations from this entire space [19], [20], [78], intrusion detection operates on network traffic that contains a large and heterogeneous set of potential signals. Packet headers, timing patterns, connection states, and behavioral sequences across protocol layers are all plausible candidates for monitoring. In practice, human engineers must explicitly select which subset of these candidates to measure, and this design choice defines the feature space on which learning occurs. When the selected feature space lacks sufficient discriminative information, we observe two failure modes in our preliminary analysis (§2). First, models require higher FPRs to reach reported detection rates, which conflicts with operational requirements for low FPRs (§2.1). Second, attacks that primarily manifest

in unmonitored features of the traffic may appear benign in the monitored features and thus evade detection, a behavior reminiscent of mimicry style attacks [92] (§2.1).

To close this gap, a natural idea is to enumerate a rich set of features and learn representations over this enlarged space. In NIDS, however, this is impractical for two reasons. First, real-time processing at high-packet rates hinders monitoring every potential signal; unlike malware detection, where feature spaces with millions of dimensions are feasible for static analysis [16], a NIDS must balance attack-surface coverage with compactness for efficient processing. Second, data-driven feature engineering requires comprehensive datasets spanning diverse attack behaviors [9], [10], [42], [58], which remain scarce in network security [38], causing learned representations to generalize poorly to unseen variants.

We therefore seek a feature space that is both discriminative enough to detect attack variants, and compact enough for efficient monitoring. To achieve this, we require a proxy for approximating the space of possible attack strategies to serve as a foundation for feature design. To this end, we turn to open-source attack implementations as a concrete, analyzable record of attacker strategies and introduce KNOWML as a framework for systematic analysis of attack implementations to extract knowledge-augmented features. Given an attack family name (e.g., TCP DoS), KNOWML employs LLMs to construct a Knowledge Graph (KG) from code repositories. For example, the "TCP DoS" family encompasses any manipulation of TCP to achieve denial of service, where manipulating a specific field (e.g., window-size value [67]) constitutes a variant. KNOWML then applies reasoning to analyze attack strategies and their combinations, deriving a *Knowledge-Augmented Feature Space* (KAFS) grounded in attack semantics.

We validate our hypothesis and the utility of KNOWML through three complementary evaluations. First, we compare KAFS with three widely used feature spaces across two anomaly detection models on diverse network datasets, isolating the impact of feature design from model architecture and hyperparameter choices (to avoid the known "benchmark lottery problem" [27]). Second, we assess whether advanced data-driven generalization techniques can substitute for a knowledge-guided feature space by comparing KNOWML with state-of-the-art incremental and contrastive learning frameworks. Third, we conduct ablation and efficiency analyses to quantify individual component contributions and demonstrate that KAFS remains computationally feasible while delivering the strongest overall generalization.

In this paper, we make the following contributions:

- We systematize the weaknesses of A-NIDS approaches, identifying two knowledge gaps arising from reliance on non-discriminative features and from an incomplete understanding of attacker strategies (§2).
- We introduce KNOWML (§4.1), a framework that constructs a Knowledge Graph of attack strategies from code repositories using LLMs (§4.2) and derives a *Knowledge-Augmented Feature Space* grounded in attack semantics.
- We demonstrate through empirical evaluation that identified *knowledge gaps* cannot be bridged through machine learning techniques, and propose KAFS that achieves up to 99% F1-score while maintaining FPR at or below 0.0137% on both IoT and enterprise networks, substantially outperforming baseline approaches (§5.2).
- We release KNOWML's implementation and constructed KGs, and KAFS across CICIoT2023, CIC-IDS2017.[1] To address the scarcity of recent attack variants in existing benchmarks, we collect, annotate, and release 9 attack variants, including exploits of recently disclosed vulnerabilities absent from standard datasets (§5).

2. Motivation

In this section, we motivate our work by examining how current A-NIDS behave under two practical constraints, namely maintaining low false positive rates and detecting sophisticated attack variants. We show that detection performance degrades substantially when these constraints are enforced.

Anomaly Detection. Given benign traffic data $\boldsymbol{X}$ sampled from the benign distribution $\mathcal{D}$, unsupervised anomaly detection models learn normal behavior representations and assign anomaly scores to new samples. A threshold φ, determined from validation data, flags samples as anomalies when scores exceed φ (reconstruction-based) or fall below it (density-based). An effective A-NIDS must maintain high TPR under stringent thresholds ensuring low FPR. Formally, we define "reducing FPR" as adjusting φ on validation data such that FPR $\leq \varphi$ [54].

Feature-Space Baselines. To assess how the choice of feature space affects detection performance, we compare against three widely used methods that represent distinct paradigms in the field. The first baseline is the Kitsune feature space (KIT) [63], which provides a manually-defined set of 100 features designed from partial attack knowledge, such as using jitter in IP camera streams to detect man-in-the-middle attacks. This baseline represents the paradigm of expert-driven feature engineering based on domain-specific attack characteristics. The second baseline is the Standardized feature space (SFS) [80], which derives features from CISCO NetFlow standards [23] and extracts them using Argus [76]. This baseline is representative because it follows industry-standard network flow specifications and is commonly used to extract feature spaces for widely-adopted benchmark datasets such as UNSW-NB15 [65]. The third baseline is CICFlowMeter Features (CIC) [82] in its corrected version [34], which serves as the default feature space for prominent benchmark datasets including CIC-IDS2017 and CSE-CIC-IDS2018 [82].

2.1. The FPR–TPR Trade off

In operational deployments, Anomaly-based ML-NIDS (A-NIDS) must sustain very low false positive rates to avoid causing alert fatigue. Following prior work [22], [63], [81], [95], we regard an FPR of at most 0.1% as operationally practical.

1. https://github.com/s2labres/KnowML

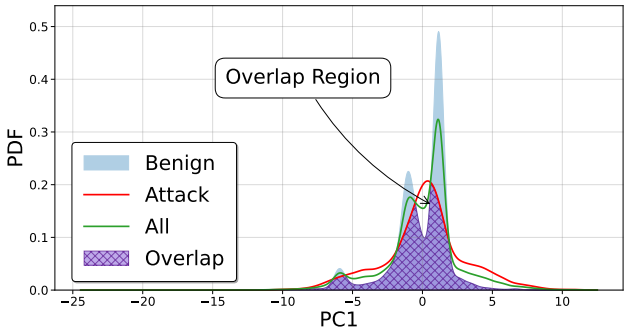
TABLE 1: **Detection Performance (%) of Two Baseline Anomaly Detectors on CIC-IDS2017 Dataset (CIDS-17).** FPR_{tr} denotes the target False Positive Rate (FPR) used for threshold tuning, and FPR_{te} the rate measured on the test set.

FPR_{tr}	Anomaly Model	Metric	DoS GoldenEye	DoS Hulk	Benign FPR_{te}
≤ 10.0	GMM	TPR	93.38	93.72	11.02
		F1-score	92.71	96.26	
	EoA	TPR	72.62	57.15	9.98
		F1-score	78.99	72.16	
≤ 0.1	GMM	TPR	17.59	10.15	0.10
		F1-score	29.90	18.43	
	EoA	TPR	63.87	38.94	0.10
		F1-score	77.92	56.04	

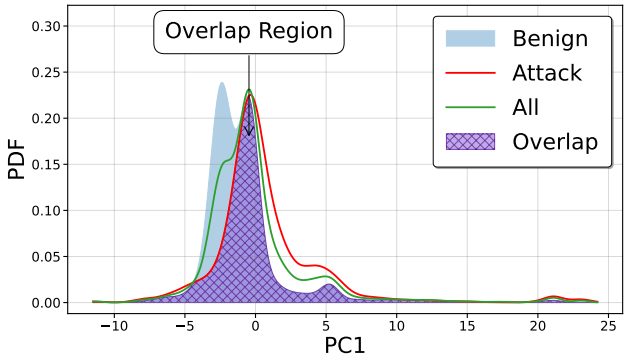
In this section we examine how enforcing low FPR affects True Positive Rate (TPR) for A-NIDS. We focus on two representative anomaly detectors, Gaussian Mixture Models (GMM) [11] and Ensembles of Autoencoders (EoA) [63], which we refer to collectively as baselines (for additional details on the experimental setup and results, see §5). Our objective is to systematically assess the respective roles of the two core components of an A-NIDS, namely the machine learning model and its feature space. We proceed in two stages. In the first stage, we fix the feature space and vary attack difficulty. Using KIT, we evaluate the baselines on two progressively more challenging attacks whose key behaviors are not explicitly represented in the monitored features. The first is a volumetric attack whose throughput increases during the attack (Observation 1). The second is a non-volumetric variant collected specifically for this study (Observation 2), both variants aim to achieve the same outcome–DoS. In the second stage, we fix the ML-model and vary the feature space to assess the contribution of feature design. We compare three widely used feature spaces (CIC, SFS, KIT) (Observation 3). The complete set of results for all feature spaces is presented in §5.

Observation 1: TPR degrades when attack traffic partially overlaps with benign under constrained FPR. When enforcing operationally realistic FPR constraints ($\leq$0.1%), testing on attack variants that partially overlap with benign traffic (Figure 1a) leads to substantial TPR degradation. This effect is not always visible in prior evaluations, which often adopt higher FPR thresholds, for example, 5% in [100], or emphasize other metrics such as Precision (for example, Precision up to 97.25% in [103]) without explicitly constraining FPR [37], [53], [77], [97]. For illustration, consider GMM [11] and EoA [63], both trained on benign CIC-IDS2017 traffic [82] and evaluated on DoS-GoldenEye and DoS-Hulk (HTTP DoS variants). Under an FPR constraint of at most 0.1%, GMM TPR on DoS-GoldenEye decreases from 93.38% to 17.59%, and EoA TPR on DoS-Hulk decreases from 57.15% to 38.94% (Table 1). Similar patterns under strict FPR constraints are reported in [46], suggesting that maintaining high TPR at low FPR remains a systemic challenge for current A-NIDS approaches.

Observation 2: TPR collapses under substantial attack–benign overlap and constrained FPR. Performance degradation becomes more pronounced when attack variants closely resemble benign traffic, particularly for attacks that do not cause significant changes in network throughput. The Nkiller2 DoS attack [67] illustrates this challenge. Nkiller2 manipulates the TCP window size while preserving packet lengths, inter-arrival times, and throughput, features that are central to many existing detection methods [30], [39], [40], [63]. By maintaining throughput patterns that are statistically similar to benign traffic, Nkiller2 can evade detectors that rely primarily on volume-based anomalies. This effect is visible when projecting Kitsune's feature space onto its most discriminative principal component (PC1), where Nkiller2 traffic coincides with benign traffic (Figure 1b). Under low FPR constraints, this overlap leads to detection failure. On CICIoT2023, Kitsune attains 99.88% F1 score on conventional TCP DoS attacks yet yields 0% F1 score on Nkiller2. This discrepancy highlights that current A-NIDS may struggle to generalize to operationally realistic variants whose behavior is not adequately captured by the monitored features.



(a) Partial overlap: DoS-Hulk vs. benign traffic



(b) Substantial overlap: Nkiller2 vs. benign traffic

Figure 1: **Attack-Benign Distributional Overlap in KIT feature space.** Probability density distributions projected onto the first principal component (PC1). (a) DoS-Hulk exhibits partial overlap with benign traffic, forcing a TPR-FPR trade-off. (b) Nkiller2 exhibit near-complete overlap, rendering detection infeasible at low FPR.

Observation 3: Inconsistent TPR across feature spaces for attacks that are statistically-distinct from benign. Under the same FPR constraints, baseline models can exhibit inconsistent performance even on attacks that are statistically distinct from benign traffic. We evaluate EoA with hyperparameter optimization on three feature spaces for CICIoT2023 DDoS attacks. As shown in Figure 2, these attacks display markedly elevated packet rates with

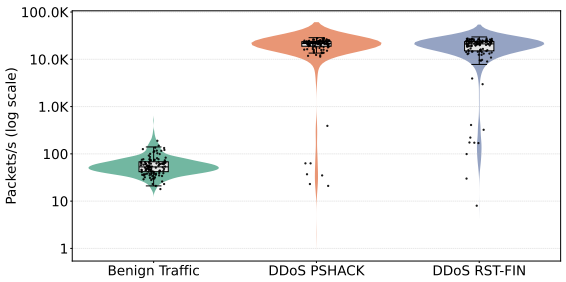


Figure 2: **Statistical Separation in Packet Rate Distributions.** DDoS attacks (DDoS-SYN/ACK Flood and DDoS-RST/FIN Flood) on CICIoT2023 exhibit substantially elevated and concentrated packet rates compared to benign traffic, representing statistically obvious attack signals that should enable straightforward detection.

TABLE 2: **Detection Inconsistency Across different Feature Spaces.** F1-scores (%) on CICIoT2023 DDoS attacks under FPR ($\leq 0.1\%$). Applying identical model architecture (EoA) with optimized hyperparameters. Despite all feature spaces containing packet rate features and attacks exhibiting obvious statistical separation, different feature spaces exhibit markedly different detection capabilities.

Attack	SFS	CIC	KIT
DDoS-PSHACK (CIoT-23)	0.00	0.78	98.31
DDoS-RSTFIN (CIoT-23)	0.00	0.00	99.50

clear distributional separation from benign traffic. Although all three feature spaces include packet/s features, detection performance differs substantially (Table 2). KIT achieves 98.31% and 99.50% F1-scores on DDoS PSHACK and DDoS RSTFIN, respectively, whereas the SFS [80] and CIC [82] obtain 0% F1-score on both attacks. Upon inspection of the evaluated feature spaces, it reveals that KIT aggregates packet size and rate at different endpoint levels, suggesting that the way traffic statistics are aggregated, rather than their mere presence, plays a critical role in effective detection for A-NIDS.

2.2. Identified Knowledge Gaps

The three observations indicate that current A-NIDS can be limited by *inadequate feature-space design*. In particular, available features may lack attack-relevant semantics or appropriate aggregation granularity to reliably separate benign from malicious traffic under practical FPR constraints. These limitations can be viewed as arising from the following two knowledge gaps.

Gap I: Out-of-Dimension Blindness. The first gap appears when attacks exploit dimensions that are not represented in the feature space. Human-designed feature spaces are constrained by researchers' knowledge of plausible attack strategies and specific detection tasks. When discriminative dimensions are absent, attack traffic can become statistically indistinguishable from benign traffic, which is reflected in Observations 1 and 2. The Nkiller2 attack exemplifies this behavior. Kitsune, for instance, monitors jitter and packet size but does not include TCP window-related features, leading to near-complete overlap between Nkiller2 and benign traffic in the monitored

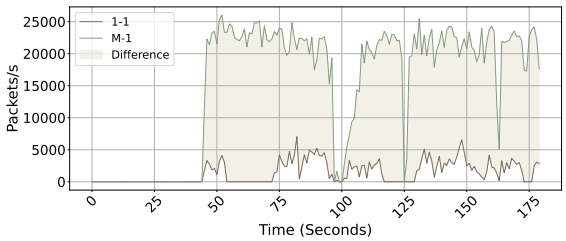


Figure 3: **Detection Failure Under Per-Flow Analysis.** RST-FIN packets/s during distributed DoS. Individual flow monitoring (1:1, brown) shows benign-appearing traffic below detection thresholds. Aggregating multiple sources to single victim (M:1, green) exposes attack bursts (shaded regions) invisible to per-flow feature space.

space. The feature space illustrated in Figure 1 similarly lacks HTTP protocol semantics, which makes HTTP-based attacks difficult to distinguish from benign traffic when throughput does not differ substantially (Observation 1). Under such conditions, achieving high TPR at low FPR becomes infeasible: the substantial overlap between attack and benign distributions allows for no decision boundary that improves TPR without incurring excessive false alarms.

Gap II. Attack Strategy Aggregation Failure. The second critical gap arises when attackers employ combined strategies where individual attack components remain invisible when monitored individually. A canonical example is a multi source distributed DoS attack. Many contemporary A-NIDS are trained on feature spaces that provide per-flow statistics and analyze each connection in isolation. When monitoring traffic to a single victim ($N = 1$), short-lived TCP connections from multiple sources often appear benign on a per-flow basis and remain below the anomaly threshold φ. In contrast, aggregating connections across sources (M attackers to $N = 1$ victim) reveals clear bursts, as illustrated in Figure 3, where $M : 1$ temporal aggregation exposes patterns that are invisible in individual $1:1$ flows. Observation 3 demonstrates this effect. Even when attacks exhibit clear statistical signals, such as elevated packet rates in Figure 2, detection success depends on how features aggregate traffic. For example, an $M-N$ feature that sums corresponding $1-N$ flows sharing the same destination can make a distributed attack salient, whereas purely per-flow features may not. In our experiments, the same packet rate signal yields a 99.50% F1 score under one aggregation scheme (KIT) but 0% under others (SFS, CIC), despite identical underlying traffic and model architecture (Table 2). Although distributed attacks are well studied [6], [75], [83], this knowledge is not always reflected in feature space design, which can hinder the detection of combined attack strategies. These limitations do not invalidate existing feature spaces but indicate that they are insufficient on their own under realistic operational constraints.

3. Our Approach: Leveraging Attack Knowledge for Systematic Generalization

Current A-NIDS fall short because human-designed feature spaces cannot capture the diverse attack sur-

faces adversaries exploit (Gap I) nor represent the combined strategies attackers employ (Gap II). We propose KNOWML, which addresses both limitations by mining known attack implementations to systematically enumerate attack dimensions and their combinations through a Knowledge Graph (KG), and translating these insights into Knowledge-Augmented Features (KAFS) that encode both atomic attack tactics and their composite strategies. Unlike human-designed features that rely on researcher intuition about plausible attacks, our approach grounds feature space construction in *actual attacker behavior* extracted from relevant open-source attack implementations, ensuring coverage of dimensions and aggregations that baselines miss.

We construct this KG from publicly available attack implementations. This approach is practical because adversaries demonstrably reuse attack mechanisms across time. Analysis of NIST vulnerability databases reveals recurring patterns; e.g., the previously mentioned Nkiller2 DoS appeared in 2008, 2009, and 2024 [25], [64], [67]; Slowloris connection throttling was reported in 2007 and resurfaced in 2025 [1], [5]; a 2025 HTTP denial-of-service attack combined a 2024 header exploitation with request smuggling from 2020 [3], [4], [12]. While this analysis does not claim comprehensive coverage of all attack variants, these examples illustrate that past attack knowledge provides actionable primitives for anticipating emergent threats through recombination.

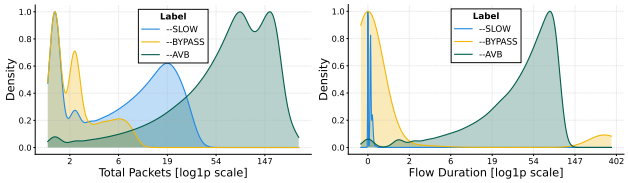
To extract these behavioral primitives, we analyze launch parameters from attack implementations' `README.md` documentation. Launch parameters are command-line arguments controlling attack execution, typically documented as:

```
$ python script.py --keep-alive
                   --syn_flag
  --keep-alive: Maintain persistent \
                connections
  --syn_flag: Send SYN packets with \
              specified flags
```

These parameters encode behavioral attack strategies specifying how attackers vary their techniques. For example, launching a TCP DoS attack with `--keep-alive` and `--syn_flag` prolongs resource exhaustion by preventing connection timeouts while flooding SYN queues (see §4.2 for how we remove unrelated parameters). Launch parameters effectively represent attack variations, as demonstrated by their use in prior work for generating attack variants in generalization testing [91]. As shown in Figure 4, different parameter combinations for HTTP DoS attacks (SLOW, BYPASS, AVB) produce measurably distinct traffic patterns in packet count and flow duration distributions. This variability confirms that launch parameters encode operationally relevant behaviors, making them effective primitives for enumerating attack strategies and analyzing their compositions.

4. KNOWML

Informed by the knowledge gaps and motivation in §2 and §3, KNOWML pursues two core objectives. First, we construct a Knowledge Graph of attack strategies organized by attack family, as different protocols expose



(a) Total Packet Count (b) Flow Duration

Figure 4: **Launch parameter variations in HTTP DoS attacks [51] produce distinct traffic distributions.** Different settings (SLOW, BYPASS, and AVB for details see [51]) yield measurably different behaviors in (a) Packet Count and (b) Flow Duration, demonstrating that parameters encode operationally significant attack strategies.

distinct attack surfaces through their states, header structures, and connection semantics. Second, we analyze how these strategies combine through parameter compositions to reveal attack variants emerging from combined strategies. Note that in this work we focus on automatically extracting attack strategies and analyzing their combinations. We intentionally retain a human in the loop when mapping identified strategies to measurable features, which enables the incorporation of domain-specific expertise that current LLMs do not capture in a sufficiently reliable or fully automatic manner. We discuss the implications of this design choice in §6.

Threat Model. We focus on active network attacks that generate observable traffic patterns at the flow level, consistent with the operational constraints of anomaly-based NIDS. Our KG is constructed from publicly available attack implementations, ensuring reproducibility while covering widely deployed threat classes (a more detailed discussion of adversaries in this space is provided in §6). We assume an attacker who uses automated scripts to deploy one or more known attack strategies. We exclude three categories from scope. First, we exclude passive reconnaissance attacks (e.g., off-path TCP inference [35], [36]), which produce no anomalous flow-level traffic. Second, we exclude content-dependent attacks requiring Deep Packet Inspection (DPI), such as SQL Injection or Command Injection, as these operate at the payload level beyond flow-based monitoring. Third, we exclude entirely novel strategies with no publicly-available implementation, as our approach is grounded in observable implementation characteristics.

Background: Knowledge Graphs and Symbolic Reasoning. In contrast to statistical and data-driven methods that learn from raw data, symbolic reasoning operates on structured, human-interpretable knowledge to deduce general rules [99]. It models relationships between abstract concepts and integrates structural and semantic knowledge, enabling relational dependence analysis of concepts.

A primary tool for implementing symbolic reasoning is the Knowledge Graph (KG). A KG is a directed graph composed of subject-predicate-object (s, p, o) triples, where each triple defines a relationship between entities [52]. We formally define our KG as $G = (V, E)$, where nodes $V = S \cup R \cup F$ consist of strategy nodes $S = \{s_1, \ldots, s_n\}$ representing attack techniques, reposi-

tory nodes $R = \{r_1, \ldots, r_m\}$ representing repository implementations, and attack family nodes $F = \{f_1, \ldots, f_k\}$ representing attack families. The primary edge set $E_{SR} \subseteq S \times R$ links strategies to their implementing repositories. After clustering semantically similar strategies into a partition $\Pi = \{C_1, \ldots, C_p\}$ of S, where each cluster $C_i \subseteq S$ contains related strategies (§4.2.3), we select one representative strategy s_j per cluster to form the atomic strategy set $\mathcal{A} = \{s_1, \ldots, s_p\}$. Symbolic reasoning rules (§4.3) then derive transitive edges $E_{\text{trans}} \subseteq \mathcal{A} \times \mathcal{A}$ capturing strategy co-occurrence patterns.

4.1. Overview of KnowML

KnowML is a framework for constructing KG of attack strategies and inferring Knowledge-Augmented Features (KAFS) that directly address the generalization challenge of A-NIDS under operational constraints. The overall pipeline is shown in Figure 5 and consists of the following stages:

- **KG Construction ❶ – ❸** (§4.2): Open-source attack repositories are automatically parsed to build a KG given an attack family name. This step unifies identified attack strategies and selects representative strategies to build the final KG.
- **Symbolic Reasoning ❹** (§4.3): Inference rules are applied over the KG to enumerate possible attack strategies and analyze their combinations through transitive relations.
- **Knowledge-Augmented Feature Space (KAFS) ❺** (§4.4): Identified strategies and their combinations are translated into features through defined rules.

4.2. KG Construction

We construct a KG by systematically extracting attacker strategies from open-source implementations. Specifically, we extract parameter values and their description from `README.md` (see §3 for rationale). Note that here we intentionally avoid examining the specific values assigned to these parameters (e.g., `--keep-alive=0` vs. `--keep-alive=1`) and only capture the parameter name and its description. This abstraction avoids overfitting to narrow configurations and enables generalization to variants generated by different parameter value combinations (empirically validated in §5). The next section describes how we (1) acquire relevant repositories, (2) construct graphs from individual repositories, and (3) merge them into a unified KG.

4.2.1. Source Acquisition ❶. We design a pipeline to mine open-source repositories for diverse attack implementations (see ❶ in Figure 5 and details of the acquisition process are provided in §A). Given an attack family name (e.g., "HTTP DoS"), the pipeline queries the GitHub API to search [44] repository names and descriptions. To account for naming variations, we combine the family name with protocol-specific keywords such as "HTTP Flood," "HTTP GET," and "HTTP Overload". We retrieve `README.md` files from matched repositories, filtering out those with missing or empty documentation. These files serve as the primary source for constructing Repository Graphs in subsequent steps. GitHub-based acquisition was selected over alternative sources (e.g., NIST NVD, security advisories) because it provides structured, parsable documentation (`README.md` files) amenable to large-scale automated extraction. While vulnerability databases contain valuable information, they typically provide only brief descriptions of encountered vulnerabilities without the behavioral parameters necessary for our analysis. The implications of this design choice, including coverage limitations, are discussed in §6.

4.2.2. Repository Graph Construction ❷. In this step, we transform individual attack implementations retrieved in ❶ into Repository Graphs. To achieve this, we define an ontology (KG schema) following best practices in ontology engineering [52]. The schema specifies four key classes and properties: (i) *Attack Family Name* (e.g., TCP DoS), (ii) *Strategy* (a specific technique employed by attackers), (iii) *Description* (a textual explanation of how the strategy configures the attack), and (iv) *Repository Identifier* (a URI linking each strategy to its source repository). The URI is a functional property, ensuring a one-to-one mapping between strategies and repositories. The Repository graph is illustrated in Figure 6. We cast the extraction problem as Named Entity Recognition (NER) and employ Large Language Models (LLMs) to identify entities and relations from code and documentation at scale. Prior work has demonstrated that LLMs are effective for a range of security tasks [7], including security-oriented NER applications [32], [43], [61], [79]. We adopt a GraphRAG-style approach [32], enabling LLMs to extract the most relevant entities. Using the GPT-4o-mini model, our implementation achieves 90.91% Recall on our NER benchmark (see §B for details) comparable to state-of-the-art systems especially designed for NER tasks [79]. This benchmark is based on a human-annotated dataset we created for evaluation, and the high Recall demonstrates strong agreement with human annotations. This design enables large-scale, automated construction of Repository Graphs, previously infeasible with manual analysis. The subsequent paragraphs discuss how KnowML mitigates issues related to scalability, Recall degradation (due to increased context-window sizes), and LLM hallucinations.

Logical Consistency and Tractable Reasoning. To ensure that the constructed KG remains scalable and suitable for symbolic reasoning, we defined the ontology in accordance with OWL Lite standards [26], which guarantee polynomial-time reasoning under Description Logic (DL). We further validated the ontology using the HermiT reasoner [28], confirming that (i) no contradictions exist, (ii) all concepts are satisfiable, and (iii) all axioms are mutually compatible.

Scalability and Recall Preservation. A challenge in extracting attack strategies arises from the unpredictable length of `README.md` files and source code, which can degrade LLM Recall as context size increases [55], [60]. To address this, we implemented an iterative "gleaning" method. After the initial extraction, the LLM performs binary (`YES/NO`) checks for missed entities, guided by logit bias to enforce confirmation. If the model returns `YES`, an additional extraction round is triggered to extract missed entities [32].

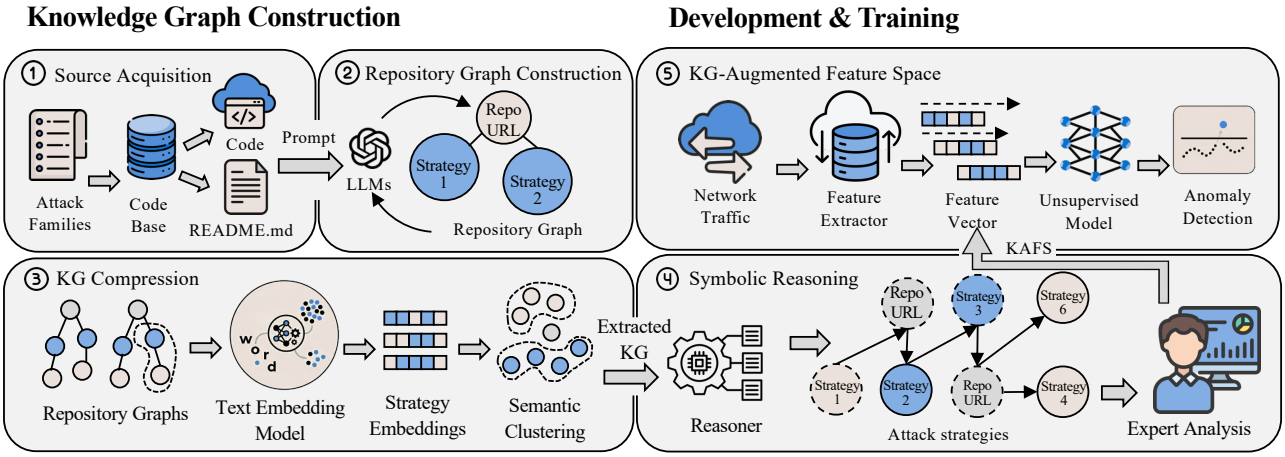


Figure 5: **Overview of the KnowML Pipeline from Attack Implementation to Feature Generation.** Given an attack name, the system retrieves its implementation and constructs repository graphs, which are then unified into a KG (Steps 1–3). Symbolic reasoning is applied over the KG to infer atomic, and composite strategies (Step 4). These results are translated into KAFS that serve as inputs to A-NIDS for training and evaluation (Step 5).

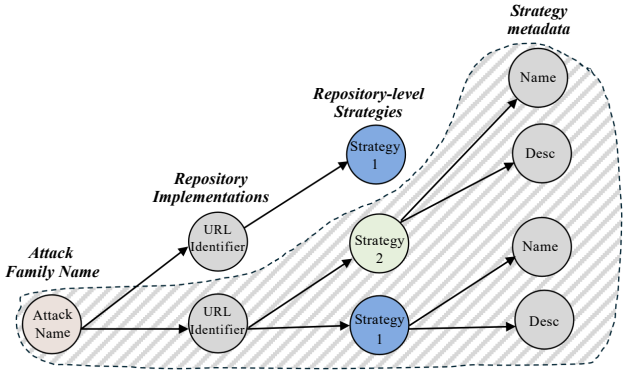


Figure 6: **Repository Graph Structure.** Each Repository Graph contains: 1) a shared Attack Family Name root node (e.g., "HTTP DoS"), 2) URL Identifier nodes representing individual repositories, and 3) Strategy nodes extracted from each repository, where each strategy has Name and Description ('Desc') properties. Dashed borders delineate individual repositories. The two blue-tinted Strategy 1 nodes represent semantically similar strategies extracted from different repositories, which will be merged during KG compression (§4.2.3).

Managing Hallucination and Noise. LLMs are known to hallucinate, i.e., generate inaccurate or fabricated entities [49]. To mitigate this, we employed a few-shot prompting strategy with explicit examples that cover the full spectrum of expected inputs: (i) *Structured inputs*, where extraction is straightforward (e.g., option lists in code or documentation: `python script.py [-p1] [-p2]` with `-p1` and `-p2` explained); (ii) *Unstructured inputs*, where strategies are described in free-form text; (iii) *Empty inputs*, where no parameters are explicitly mentioned (e.g., simply `$ python script.py`); and (iv) *Irrelevant inputs*, where search results contain unrelated repositories (e.g., an HTTP Redis pooler returned in response to an HTTP DoS query [47]). In cases where the LLM extracts unrelated parameters (for example, `threads=1`), we first cluster related inputs using the procedure in §4.2.3 and then manually discard the irrelevant clusters. This is efficient because removal is performed at the level of clusters that correspond to unique parameters. Moreover, we enforced structured output templates aligned with our ontology [72], placing each entity into predefined fields. By constraining the LLM to produce structured outputs and applying few-shot prompting covering diverse scenarios, KnowML achieves over 90% Recall in the NER task, matching human-annotated data and reducing hallucination (see §B).

4.2.3. KG Compression ❸. To eliminate redundant strategy nodes, we compress the extracted Repository Graphs by clustering similar strategies, e.g., "Randomize Ports" and "Random Port Targeting". Each strategy is embedded using its Name and Description via OpenAI's text-embedding model [73], and clustering is performed with Hierarchical Agglomerative Clustering (HAC) [66]. We adopt HAC with complete linkage, as it maximizes inter-cluster separation and yields more fine-grained clusters compared to alternatives [29], ensuring that semantically distinct strategies are not merged. Here, we prioritize inter-cluster distance over intra-cluster distance, ensuring that the worst case produces a few repetitive clusters rather than losing unique attack strategies. For each cluster C_j, we select a representative strategy S_j defined as the node with minimum cumulative embedding distance to all others in the same cluster:

$$S_j = \arg\min\{ \sum_{i:s_i \in C_j} \|e_i - e_\ell\| : s_\ell \in C_j \}, \quad (1)$$

where e_i is the embedding of strategy s_i. This representative preserves the core semantics of the cluster while eliminating redundant variants.

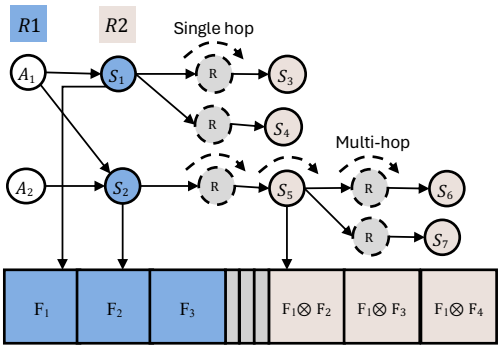


Figure 7: **Example of Symbolic Reasoning Rules (top) to derive Knowledge-Augmented Feature Space (bottom)**. Atomic Strategy (R1), and Transitive Rule (R2).

4.3. Symbolic Reasoning ❹

We define two rules that transform the KG into KAFS: (i) Atomic tactics (Atomic Rule §4.3.1, addressing Gap I), (ii) composite strategies (Transitive Rule §4.3.2, addressing Gap II). Together, these rules form the Knowledge-Augmented Features Space in Figure 7.

4.3.1. Atomic Strategy Rule (R1). The atomic rule enumerates a unique, representative strategy from each cluster of repetitive strategies. This rule ensures comprehensive coverage analysis of possible attack surface, including rare or legacy strategies and mechanisms that attackers may employ, which can potentially resurface in future attacks (as described in §3). Given clusters $\Pi = \{C_1, \ldots, C_p\}$ from KG compression (§4.2.3), we select one representative S_j for each cluster that minimizes cumulative embedding of its name and description to all other strategies within the same cluster (Equation 1):

$$\mathcal{A} = \{S_j \ : \ C_j \in \Pi\}. \tag{2}$$

4.3.2. Transitive Rule (R2). The Transitive Rule captures composite strategies by identifying tactics that co-occur either within a single repository or across multiple repositories. This reflects how attackers may chain strategies, configuring multiple parameters jointly or combining tactics learned from different implementations. For instance, many TCP DoS tools allow enabling both `--ACK` and `--fragment`, yielding a *Fragmented ACK* attack that merges flooding with fragmentation [90], or distributed DoS by combining `--IP-Spoofing` and `--Flood`.

We construct transitive edge $E_{\text{trans}} \subseteq \mathcal{A} \times \mathcal{A}$ as follows. First, *Single-hop co-occurrence:* Two representative strategies $S_i, S_j \in \mathcal{A}$ are directly connected if $\exists r \in R, \exists s_1 \in C_i, \exists s_2 \in C_j$ such that $(s_1, r) \in E_{SR} \wedge (s_2, r) \in E_{SR}$. Second, *Multi-hop chaining:* We compute the *transitive closure* of these direct edges: if there exists a path $S_x \to S_i \to \cdots \to S_y$ through direct edges, then $(S_x, S_y) \in E_{\text{trans}}$.

For a representative strategy $S_x \in \mathcal{A}$, the set of transitively connected strategies is (illustrated in Figure 7):

$$TC(S_x) = \{S_y \in \mathcal{A} : (S_x, S_y) \in E_{\text{trans}}\}. \tag{3}$$

This construction exposes both direct (single-hop) and indirect (multi-hop) composites, expanding the feature space to cover variants that recombine known tactics in novel ways.

4.4. Knowledge-Augmented Feature Space ❺

We translate the symbolic reasoning outputs (R1 and R2) into measurable network features that directly address the identified knowledge gaps. R1 produces atomic features addressing Gap I (Out-of-Dimension Blindness), while R2 produces composite features with adaptive aggregation addressing Gap II (Attack Strategy Aggregation Failure).

R1 → Atomic Strategy Features (Addressing Gap I). The Atomic Rule enumerates representative strategies $\mathcal{A} = \{S_1, \ldots, S_p\}$ from KG compression. Each atomic strategy $S_i \in \mathcal{A}$ maps to a measurable network feature f_i via function ϕ_1:

$$\phi_1 : \mathcal{A} \to \mathcal{F}_{\phi_1}, \quad \phi_1(S_i) = f_i. \tag{4}$$

For example, strategy `--window-size` maps to feature *TCP window size*, while `--packet-size` maps to *packet size*. These atomic features are extracted at the per-flow level (5-tuple: Source IP, Source Port, Destination IP, Destination Port, Protocol). By systematically enumerating dimensions from attack implementations, $\mathcal{F}_{\phi_1}$ incorporates behavioral dimensions absent from human-designed feature spaces.

R2 → Composite Features (Addressing Gap II). The Transitive Rule identifies strategy pairs (S_i, S_j) connected via E_{trans}, indicating co-occurring tactics in composite attacks. For each such pair where $S_j \in TC(S_i)$, we construct a composite feature via function ϕ_2:

$$\phi_2 : \{(S_i, S_j) : S_j \in TC(S_i)\} \ \to \ \mathcal{F}_{\phi_2}, \tag{5}$$

$$\phi_2(S_i, S_j) = f_i \otimes f_j \tag{6}$$

where $\otimes$ denotes the aggregation operator (e.g., element-wise multiplication, joint counting). For instance, detecting `--ACK` and `--fragment` co-occurrence yields composite feature `tcp_ack_fragment_count`.

Analysis of our KG reveals that 70% of composite strategies involve IP spoofing (source IP manipulation), indicating distributed attack patterns where multiple sources target a single destination. This finding directly informs our aggregation design. To expose such M:1 attack patterns invisible in per-flow analysis (Observation 3), we compute composite features at *destination-level aggregation*: statistics are aggregated across all flows sharing the same (Destination IP, Destination Port), regardless of source. Formally, for composite feature $f_i \otimes f_j$, we aggregate over all conversations $(H_{\text{src}}, H_{\text{dst}})$ where H_{dst} is constant.

Feature Value Extraction. For each feature $f_i \in \mathcal{F}_{\phi_1} \cup \mathcal{F}_{\phi_2}$, we compute four incremental statistics: mean (μ), standard deviation (σ), cumulative sum (CS), and sum of squared residuals (SSR). We employ Welford's online algorithm [94] and the damped windowing approach from [63], enabling constant-time updates with $O(1)$ memory per feature (details in §C). To summarize our two-level monitoring: 1) Channel-level: Per-flow

statistics identified by 5-tuple, capturing individual connections, and 2) Destination-level: Aggregated statistics for all traffic to a destination, identified by (Destination IP, Destination Port), capturing distributed attack patterns.

At time t, the complete KAFS feature vector is:

$$\boldsymbol{v}(t) = \mathcal{F}_{\phi_1} \cup \mathcal{F}_{\phi_2}, \tag{7}$$

where $\mathcal{F}_{\phi_1}$ contains atomic features addressing missing dimensions (Gap I), and $\mathcal{F}_{\phi_2}$ contains destination-aggregated composite features addressing strategy combinations (Gap II).

5. Evaluation

We evaluate KNOWML by addressing the following research questions:

RQ1: Does KNOWML's KAFS achieve effective attack detection while meeting the practical FPR requirements of realistic operational environments? (§5.1)

RQ2: Does KNOWML's KAFS improve generalization across different attack variants compared to other feature spaces? (§5.2)

RQ3: Does KAFS achieve computational efficiency suitable for online deployment rather than being limited to offline analysis? (§5.3)

RQ4: Can current machine learning techniques replace the need for KAFS? (§5.4)

RQ5: How does each reasoning rule contribute to overall performance? (§5.5)

Experimental Methodology. Our experimental evaluation follows best practices for A-NIDS assessment [14], [15]. We utilize 70% of benign traffic for training, 10% for validation and threshold tuning, and 20% strictly held out for testing, using time-aware splitting. As established in §2, operationally practical FPR is $\leq$0.1%.

Prototype. We implemented a prototype of KNOWML, covering three key components: KG construction, Symbolic Reasoning, and the extraction of KAFS from network traffic. Packet capture and preprocessing rely on `tshark`. The extraction module is designed as a plug-and-play component, supporting both offline analysis (`pcap` files) and online analysis, ensuring compatibility with a variety of A-NIDSs. For our prototype investigating TCP DoS, HTTP DoS, and Brute Force attacks, the KG is created from 7,853 repositories (details of the collection process are provided in §A), and stored in `.graphml` format to support symbolic inference (for a summary of the extracted KNOWML features, see §D). We extract 214 features in total, including flow-level metrics, temporal dynamics, and protocol-specific behaviors (comprehensive list in §D).

Feature-Space Baselines. We use the three (CIC, SFS, and KIT) feature space baselines defined in §2.

Datasets. We evaluate KNOWML on four datasets, each serving a distinct purpose in our evaluation methodology. We use two established benchmarks to assess performance across diverse deployment scenarios, as prior work shows that detection accuracy varies significantly by environment [71]. The first benchmark is CICIoT2023 Dataset (CIoT-23) [70], which contains traffic from 105 IoT devices. The second is CIC-IDS2017 Dataset (CIDS-17) [82], which emulates an enterprise network with heterogeneous protocols. For CIDS-17, we adopt the corrected labels from [34]. To evaluate generalization on attack variants absent from existing benchmarks, we collected CAP, which contains 9 variants targeting our identified knowledge gaps: out-of-dimension attacks exploiting vulnerabilities disclosed in 2025 (e.g. Nkiller2 [67], HTTP Mal [5]), non-throughput mimicry attacks that overlap distributionally with benign traffic (e.g. Low-rate TCP [56]), and parameter-variant attacks using identical tools with different configurations (e.g. SSH-Patator [57] with P=0 vs P=1 from Concap [91]). These parameter variants demonstrate that KNOWML generalizes across configuration variations rather than overfitting to specific parameter settings. Complete details for dataset collection and processing are given in §F.

A-NIDS Models. We evaluate each feature space, including that of KNOWML, with three representative A-NIDS of varying complexity: (i) Gaussian Mixture Models (GMM) [8], [11], [98]—simple probabilistic baseline that models benign traffic as a mixture of Gaussians; (ii) Denoising Autoencoder (DA) [87]—a neural model that reconstructs benign traffic patterns, with anomalies identified via high reconstruction error; and (iii) Ensemble of Autoencoders (EoA)[2] [63]—an ensemble trained on correlated feature subsets to exploit feature dependencies and learn richer representations from Kitsune [63].

5.1. Attack Detection Performance at Low FPR

The evaluation covers 3,384 scenarios per dataset (6,768 in total), combining 3 models, 94 hyperparameter configurations, 3 thresholds, and 4 feature spaces (i.e., $3 \times 94 \times 3 \times 4$, as detailed in §E). For each model, we perform hyperparameter tuning to select the configuration that maximizes Recall while keeping the FPR at or below 0.1%. Thus, every reported result corresponds to the best operating point of that model within this constraint, rather than to a single arbitrary threshold. This procedure reduces "benchmark lottery" effects [27], where apparent performance differences arise from accidental or suboptimal hyperparameter choices.

Table 3 presents the results. Benign FPR_{te} denotes the FPR on the holdout test set for the corresponding training environment. For instance, Benign FPR_{te} (CIoT-23) refers to the value obtained when trained and tested on CIoT-23 benign data. Our features achieve higher F1-scores in most cases while consistently yielding the lowest false positive rate. KIT occasionally exceeds our F1-score but only at the cost of a higher FPR. The results also show the effect of background traffic complexity. KIT performs well in simpler environments such as CIoT-23, achieving an F1-score of 99.91% for DA and 97.48% for EoA on HTTP DoS attacks. Its performance collapses in complex environments, for example, falling to nearly 0% F1-score on the GoldenEye HTTP DoS variant in the enterprise-like CIDS-17 dataset. These findings support our claim

2. Using the code from [63], we observed an exponential increase in feature-extraction runtime, as the number of packets grew. We identified the cause, implemented a patch, and verified that the original results were reproducible (see §G). All experiments are performed using the patched code for a fairer comparison.

TABLE 3: **Attack Detection at Low FPR.** F1-scores of KNOWML and baselines. **Bold** marks the best feature space per detection model, ↑ indicates KNOWML improvements, and FPR values are presented in percentage (%). Benign FPR_{te} denotes the false positive rate on the hold-out benign-only test set.

Attack	GMM				DA				EoA			
	SFS	CIC	KIT	KNOWML	SFS	CIC	KIT	KNOWML	SFS	CIC	KIT	KNOWML
DoS-SYN (CIoT-23)	0.0439	0.0146	0.9173	↑ **0.9834**	0.0897	0.0593	0.9156	↑ **0.9835**	0.0930	0.0000	**0.9990**	0.9835
DoS-TCP (CIoT-23)	0.1734	0.7509	0.8800	↑ **1.0000**	0.3273	0.3329	0.9786	↑ **0.9999**	0.3357	0.0000	0.9988	↑ **0.9999**
DoS-HTTP (CIoT-23)	0.6676	0.1453	0.0010	↑ **0.9003**	0.7277	0.0742	**0.9991**	0.9000	0.6905	0.0256	**0.9748**	0.9001
Brute Force (CIoT-23)	0.0000	0.0000	0.0000	↑ **0.9727**	0.0000	0.0000	0.0000	↑ **0.9750**	0.0000	0.0000	0.0000	↑ **0.9750**
DoS Hulk (CIDS-17)	0.0228	0.1843	0.7637	↑ **0.9178**	0.0406	0.2616	0.6317	↑ **0.9677**	0.0270	0.5604	0.8641	↑ **0.9679**
DoS GoldenEye (CIDS-17)	0.3609	0.2990	0.0150	↑ **0.9934**	0.3122	0.4376	0.0005	↑ **0.9981**	0.3052	0.7792	0.0000	↑ **0.9953**
Brute Force (CIDS-17)	0.0000	0.0000	0.0000	↑ **0.9607**	0.0002	0.0000	0.0028	↑ **0.4476**	0.0000	0.0000	0.0000	↑ **0.2243**
Benign FPR_{te} (CIoT-23)	0.0980%	0.0662%	0.0296%	**0.0017%**	0.0914%	0.1200%	0.0074%	**0.0000%**	0.0147%	0.1323%	0.0030%	**0.0000%**
Benign FPR_{te} (CIDS-17)	0.1203%	0.1021%	0.1186%	**0.0137%**	0.0971%	0.0994%	0.1465%	**0.0000%**	0.0957%	0.1047%	0.1967%	**0.0076%**

in §2, confirming that Recall degradation under partial attack-benign overlap represents a systematic limitation across contemporary A-NIDS approaches. KAFS's attack-relevant features enable discriminative separation of malicious traffic without false positive inflation, achieving 22.43-99.99% F1-scores at lower FPR compared to baselines. This demonstrates that building feature space on attack knowledge resolves the fundamental FPR-Recall tradeoff under operational constraints.

> *Takeaway 1:* KAFS achieves attack detection F1-scores above 90% in most cases while simultaneously operating at lower, operationally practical FPR levels than all baselines.

5.2. Generalization on Attack Variants

To evaluate generalization of KNOWML to variants and to highlight the limitations of existing approaches (§2), we conduct experiments on diverse attack categories. The experiment is conducted using the same model parameters as in Table 3 (hence FPR omitted as it remains unchanged). Results in Table 4 show that KNOWML outperforms baselines on most variants, irrespective of the underlying anomaly model; in the few cases where KNOWML does not outperform, the baselines result in a higher FPR (check with Table 3). Even a single two-layer DA trained with KAFS achieves performance comparable to the ensemble of autoencoders. This suggests that well-chosen features can enable models to remain effective against new attack variants without increasing model complexity, an important property for deployment in resource-constrained environments.

Table 4 also highlights how incorporating only narrow or incomplete attack knowledge into A-NIDS feature design severely limits their capacity to detect unseen variants. First, Table 4 shows that SFS and CIC consistently perform poorly on M-N endpoint aggregation. In the rare cases where their performance appears higher, we found that the attack traffic involved little variation in source IPs, reducing the scenario to what is effectively a 1-1 attack rather than a true M–N setting e.g., DDoS-ACK Fragmentation attack. Moreover, changing the model architecture does not remedy this limitation. In fact, SFS achieves higher scores on HTTP DoS across models and SYN DoS performance remains fixed at 0% F1-score, indicating that feature space, not model type, is the primary determinant of detection capability. In contrast, KIT aggregates features at the source level, giving the model a global view that yields stronger results on M–N attacks.

Second, Table 4 confirms that existing feature spaces are highly susceptible to Out-of-Dimension blindness, with contemporary approaches often collapsing to 0% F1-score across all models. This further underscores that increasing model type alone does not improve detection when the feature space fails to capture attack-relevant dimensions. An exception arises with CIC on the Nkiller2 variant. CIC's partial detection of Nkiller2 (77.92-86.96%) does not contradict Observation 2 but rather validates our thesis. CIC succeeds only because its feature space includes TCP window statistics, the exact dimension Nkiller2 exploits. This finding further confirms that encoding attack dynamics into the feature space is critical for effective detection. In contrast, SFS and KIT completely fail (0% F1-score) despite identical model architectures, confirming that out-of-dimension blindness is a feature space property, not a model limitation.

Similarly, models lacking attack-relevant semantics fail to generalize to non-throughput mimicry attacks. CIC attains an F1-score of 86.96% on Low-rate TCP only because its features include several TCP-level indicators (e.g., SYN flag counts) directly tied to the attack's mechanics. This attack exploits carefully timed bursts of SYN packets synchronized with the TCP retransmission-timeout mechanism [56], and CIC's feature space captures these semantics. These findings emphasize that generalization hinges less on architecture than on whether the feature space encodes attack-relevant semantics. By embedding this knowledge, KNOWML overcomes these limitations and achieves strong results at low FPR.

> *Takeaway 2:* Baselines achieved 0% on out-of-dimension attacks and M-N distributed attacks, confirming Observations 2-3 and both knowledge gap categories. KAFS achieved 76.99-100% on these attacks, validating that mining attack implementations addresses out-of-dimension blindness (Gap I) and aggregation failure (Gap II).

5.3. Computational Efficiency

High detection performance is insufficient if feature space extraction creates computational bottlenecks in operational settings. We evaluate KAFS's computational efficiency to verify it provides a practical solution rather than a theoretically effective but operationally infeasible approach. We evaluated computational efficiency along two dimensions: 1) feature extraction runtime, and 2) model inference time. We compare against Kitsune [63],

TABLE 4: **Generalization on Attack Variants.** F1-scores of KNOWML and baselines. **Bold** marks the best feature space per detection model, ↑ indicates KNOWML improvements, and FPR values are presented in percentage (%).

Category	Attack	GMM				DA				EoA			
		SFS	CIC	KIT	KNOWML	SFS	CIC	KIT	KNOWML	SFS	CIC	KIT	KNOWML
Out-of-Dimension	HTTP Mal (CAP)	0.0000	0.0000	0.0000	↑ **0.9844**	0.0000	0.0000	0.0000	↑ **0.9985**	0.0000	0.0000	0.0000	↑ **0.9985**
	HTTP Overflow (CAP)	0.0000	0.0000	0.0000	↑ **0.9817**	0.0000	0.0000	0.0000	↑ **0.9770**	0.0000	0.0000	0.0000	↑ **0.9800**
	Nkiller2 (CAP)	0.0000	0.8696	0.0000	↑ **0.8841**	0.0000	0.7792	0.0000	↑ **0.9190**	0.0000	0.8451	0.0000	↑ **0.9276**
	Brute Force P=0 (Con)	0.0000	0.0000	0.0002	↑ **0.9998**	0.0000	0.0000	0.0000	↑ **0.9998**	0.0000	0.0000	0.0000	↑ **0.9998**
Non-Throughput	Fiberfox AVB (CAP)	**0.9622**	0.0000	0.0000	0.8523	0.0042	0.0000	0.0000	↑ **0.7649**	0.0042	0.0000	0.0000	↑ **0.8638**
	Fiberfox BYPASS (CAP)	0.3205	0.0000	0.2786	↑ **0.9185**	0.0000	0.5509	0.0045	↑ **0.8576**	0.0000	0.0000	0.0030	↑ **0.8431**
	Fiberfox GET (CAP)	0.0321	0.1623	0.0064	↑ **0.9766**	0.2770	0.3205	**0.9957**	0.9742	0.0298	0.1619	**0.9997**	0.9742
	Brute Force P=1 (Con)	0.0000	0.0000	0.0000	↑ **0.9994**	0.0000	0.0000	0.0000	↑ **0.9994**	0.0000	0.0000	0.0000	↑ **0.9994**
	DDoS-Slowloris (CIoT-23)	0.0287	0.0027	0.0001	↑ **0.8114**	0.0000	0.0000	0.0015	↑ **0.7548**	0.0302	0.0001	0.5198	↑ **0.7560**
	DoS-Slowhttptest (CIDS-17)	0.0031	0.0000	0.0000	↑ **0.9214**	0.0002	0.3117	0.0018	↑ **0.9652**	0.0000	0.0281	0.0000	↑ **0.9647**
	DoS-SlowLoris (CIDS-17)	0.0568	0.0000	0.0031	↑ **0.9803**	0.0000	0.0131	0.0028	↑ **0.9714**	0.0000	0.0000	0.0000	↑ **0.9818**
	Low rate TCP (CAP)	0.0000	**0.8696**	0.0000	0.7699	0.0000	0.7792	0.0000	↑ **0.9194**	0.0000	0.8451	0.0000	↑ **0.9277**
Composite	DDoS-HTTP (CIoT-23)	0.8360	0.0022	0.0002	↑ **0.9696**	0.8502	0.0101	0.2909	↑ **0.9476**	0.8418	0.0033	0.8506	↑ **0.9476**
	DDoS-SYN (CIoT-23)	0.0000	0.9099	0.1292	↑ **0.9979**	0.0000	0.0004	0.9620	↑ **0.9979**	0.0000	0.0000	0.9873	↑ **0.9979**
	DDoS-TCP (CIoT-23)	0.0000	0.0000	0.4294	↑ **0.9997**	0.0000	0.0000	0.9992	↑ **0.9997**	0.0000	0.0000	0.9986	↑ **0.9999**
	DDoS-PSHACK (CIoT-23)	0.0000	0.2405	0.0728	↑ **0.9778**	0.0000	0.0008	0.9656	↑ **0.9753**	0.0000	0.0078	0.9831	↑ **0.9772**
	DDoS-RSTFIN (CIoT-23)	0.0000	0.7057	0.5166	↑ **1.0000**	0.0000	0.0000	0.9983	↑ **1.0000**	0.0000	0.0000	0.9950	↑ **1.0000**
	DDoS-SynIP (CIoT-23)	0.0000	0.0000	0.9992	↑ **1.0000**	0.0000	0.8000	0.9999	↑ **1.0000**	0.0000	0.8001	0.9999	**1.0000**
	DDoS-ACK Frag (CIoT-23)	**0.9768**	0.8534	0.0006	0.9438	**0.9803**	0.8442	0.5375	0.9438	**0.9802**	0.8237	0.9476	0.9438
	Fiberfox SLOW (CAP)	0.0000	0.1828	0.2637	↑ **0.9825**	0.0000	0.2514	0.9959	↑ **0.9796**	0.0000	0.1610	0.9997	↑ **0.9796**
	Fiberfox STRESS (CAP)	0.9312	0.2492	0.0000	↑ **0.9605**	0.9157	0.9914	0.9869	↑ **0.9291**	0.8814	0.9930	**0.9992**	0.9295

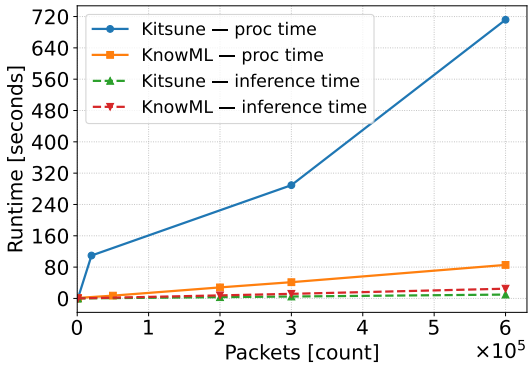


Figure 8: **Computational Efficiency.** KNOWML vs. Kitsune [63]: (a) feature extraction and (b) inference time. X-axis: input packets; Y-axis: runtime (s) with DA model.

which explicitly claims deployability as an online NIDS. Since the C++ implementation of Kitsune is not publicly available, we use the released Python version [62]. In addition, our evaluation is constrained to a single model rather than the full ensemble. In Kitsune, the Feature Mapper (FM) allocates the number of denoising autoencoders dynamically based on feature correlations (see [63] for details). To ensure a fair and unified comparison, we evaluate only the single underlying model for both evaluated feature spaces. KNOWML is also implemented in Python, which avoids bias from implementations in different programming language. All experiments were conducted on an Ubuntu 20.04 VM (2.6 GHz CPU, single thread, 4 GB RAM).

Figure 8 presents the results. Kitsune scales poorly with traffic volume, processing 1k packets required 0.42s, while 600k packets required 711.85s, showing a non-linear runtime increase. By contrast, KNOWML processed 600k packets in 85.59s (≈7,010 pps), despite having more than twice the feature dimensionality (214 vs. 100[3]). This throughput is comparable to the 7,500 pps reported for Kitsune's C++ implementation on a faster 3.6 GHz CPU [63]. Given that C++ implementations are typically 8–29× faster than Python under identical conditions [59], a C++ version of KNOWML would likely surpass Kitsune in feature extraction speed. In terms of inference, KNOWML is slower due to the higher feature dimensionality, achieving ≈24,836 pps compared to Kitsune's ≈61,166 pps. However, this overhead is offset by its more efficient extraction pipeline, supporting the suitability of KNOWML's feature space for practical deployment.

Takeaway 3: The KAFS feature space is practically obtainable in operational settings and does not introduce prohibitive computational overhead, achieving feature extraction efficiency comparable to SotA A-NIDS online systems.

5.4. Comparing with Data-Driven Approaches

We evaluate whether purely data-driven frameworks for generalization can substitute for the KAFS. This experiment tests if advances in representation learning alone are sufficient, or whether semantic features remain necessary. We focus on two recent methods that represent strong SotA approaches. AOC-IDS [101] employs a Cluster Repelling Contrastive loss within an autoencoder and reports over 91% zero-day detection. Trident [102] is an incremental learning framework that operates on existing feature spaces to improve generalization, using tScissors for outlier thresholding and tMagnifier for clustering-based class discovery. Trident and AOC-IDS have demonstrated improved generalization on existing models [18], [39], [40], making them suitable as competitive baselines for evaluating whether data-driven generalization alone can replace semantic knowledge.

3. The original paper [63] claims 115 features, whereas its artifact 100 [89].

TABLE 5: **Comparing with Data-Driven Approaches.** Per-attack F1-scores of AOC-IDS, Trident, Kitsune, and KNOWML, along with each approach's FPR on benign traffic on the same sub-sampled dataset.

Attack	F1-score			
	AOC-IDS [101]	Trident [102]	Kitsune	KNOWML
Low rate TCP (SIM)	0.0000	0.0000	0.0000	**0.9247**
Fiberfox AVB (CAP)	0.1659	0.7952	0.0000	**0.8702**
Fiberfox BYPASS (CAP)	0.0536	0.9302	0.0000	**0.9565**
HTTP Overflow (SIM)	0.0000	0.0000	0.0000	**0.9703**
Nkiller2 (SIM)	0.0000	0.0000	0.0000	**0.9348**
Brute Force P=1 (CAP)	0.0180	0.9744	0.0000	**1.0000**
Benign FPR_{te}	**0.0043**	**0.2067**	**0.0011**	**0.0000**

We applied both methods to the KIT, chosen because it performs well on M–N endpoint attacks but shows poor results on Out-of-Dimension and Non-Throughput variants; specifically, we focus on scenarios in which KIT performed 0% across all three detection models, to test whether advanced data-driven learning can compensate for KIT's limitations. Due to computational costs (even A100 GPUs with 80GB memory were insufficient), we trained on a sub-sampled dataset that remained 125 times larger than the original AOC-IDS dataset and comparable in size to Trident, with hold-out test data to avoid data snooping. Results in Table 5 show that both methods fail to generalize to attack variants. In the few cases where performance improves, it comes at the cost of exceptionally high false positive rates, for example, 20.67% with Trident. These findings reveal that machine learning techniques such as contrastive and incremental learning cannot substitute for knowledge-based feature spaces, failing to generalize under operationally required FPR constraints.

Takeaway 4: Purely data-driven generalization frameworks, including contrastive and incremental learning, provide limited improvements under operational FPR constraints and do not replace the knowledge-augmented feature space.

5.5. Ablation Study

We perform an ablation study on rule-derived features from §4.3. Using the DA model, we evaluate each rule set independently and in combination (Table 6). Atomic Features (R1) achieve strong performance across categories (average 79.19% F1), excelling on out-of-dimension attacks (93.73%) by directly encoding strategy-specific dimensions extracted from attack parameters. Composite Features (R2) perform best on M-N endpoint attacks (96.43%) where multi-source behavior requires aggregation across temporal and spatial dimensions, and attacks that employ composite strategies e.g., R2 features achieve 94.38% on DDoS ACK-Fragmentation attack compared to 31.36% F1-score for R1. Overall, each rule contributes complementary strengths. R1 ensures broad strategy coverage, and R2 captures multi-step combinations. Both rule sets are essential for generalization, as their integration outperforms either rule alone, demonstrating that effective threat knowledge embedding requires both atomic primitives and their compositions.

TABLE 6: **Ablation Study**. Average F1-scores on the DA model when using features derived from individual rules. A tick (✓) denotes inclusion of features from the corresponding rule, while a cross (✗) denotes their removal.

Category	R1 (✓, ✗)	R2 (✗, ✓)	R1,R2 (✓, ✓)
Out-of-Dimension	0.9373	0.7834	0.9761
Non-Throughput-based	0.7706	0.4542	0.9219
Composite	0.6678	0.9643	0.9815
Benign FPR_{te}	0.0006	0.0000	0.0000

Takeaway 5: Features derived from Atomic Rules (R1) and Transitive Rules (R2) provide complementary benefits, with each covering different attack variants. Their combination yields the best overall performance, indicating that effective threat knowledge embedding requires both fine grained strategy primitives and their aggregated forms.

6. Discussion

Manual Validation for Feature Mapping. KNOWML provides a general framework for structural strategy analysis; in this work, we instantiate it with a concrete feature space designed to capture the identified strategies and their corresponding attack behaviors. Our experimental results indicate that this instantiation is effective, yet the framework is not tied to a single mapping, and we encourage future work to explore alternative feature designs within KNOWML. Importantly, the aim of KNOWML is not to replace human experts. An experienced analyst may be equally capable of identifying relevant attack strategies. Rather, KNOWML automates the labor-intensive and fatigue-prone tasks (collecting, annotating, organizing, and analyzing attack implementations) that are difficult to perform manually at scale (7,853 repositories in our evaluation).

We intentionally retain a human in the loop to incorporate domain-specific expertise in feature mapping, particularly when translating abstract strategies into concrete network-level measurements and aggregation schemes as described in §4.4. This is a one-time effort per attack family that subsequently enables detection of numerous variants. In contrast, traditional approaches require expert analysis for each individual variant. Our evaluation demonstrates that a single knowledge-guided feature space, once validated, generalizes across multiple variants (Table 4), amortizing the upfront expert effort. As LLMs continue to advance, they may assist in automating this step. We leave a systematic exploration of this direction to future work.

Choice of Attack Families. In this paper, we focus on TCP DoS, HTTP DoS, and SSH Brute Force. These families are common in real-world deployments [24] and widely studied [45], [50], [86], allowing us to demonstrate that existing solutions fail to generalize even within well-studied families. While it is easier to show gaps in A-NIDS using obscure or uncommon attacks, doing so with well-known and heavily researched variants demonstrates the practical utility of our approach. KNOWML enables the systematic enumeration of attack strategies

and the examination of relationships between them. This design makes it extensible to the broader threat landscape. While our evaluation uses GitHub-sourced strategies, the KG can be enriched with proprietary threat intelligence, incident response data, or frameworks such as MITRE ATT&CK. Organizations can augment KNOWML with internal knowledge sources to address environment-specific threats not present in public repositories.

Adversarial ML Attacks. We evaluate KNOWML in the context of generalization of detection tasks with attack variants, and do not directly assess its resilience to adversarial ML. Since KNOWML distills behavioral features, we expect knowledge-augmented features to be more robust to adversarial manipulation, as attackers must satisfy problem-space constraints [74]. We plan to explore how knowledge-augmented systems may mitigate adversarial threats in future work.

Other A-NIDS Feature-Space Baselines. Other feature-space baselines include FlowLens [18], Whisper [39], and HyperVision [40]. These methods extract few packet-level features such as size and inter-packet timing (throughput-based), then transform them through quantization, Fourier analysis, or flow-level statistics. They function as data-driven feature enhancement techniques. Our results show that Kitsune, which monitors 100 throughput-based statistics, cannot detect Out-of-Dimension or Non-Throughput attacks even when combined with SotA enhancement or incremental learning frameworks (§5.4). This suggests other approaches face the same limits. That being said, HyperVision, which uses connection degree for detection, might detect M-N endpoint attacks but would suffer from the same constraint as Kitsune in detecting the other two categories of attacks. It is important to note that these systems were designed for efficiency and Tbps-scale deployment, sometimes on programmable switches, and intentionally use very few features, whereas KNOWML focuses on semantic generalization across attack variants. In future work, we plan to systematically study feature spaces from different paradigms, particularly with focus on high-speed networks.

Detecting Zero-Days. KNOWML's KG models diverse attack instances in order to distill generalizable methodology patterns rather than overfitting to specific cases. As a result, a model using the resulting KAFS detects variants that extend or combine known strategies but cannot anticipate entirely new vulnerabilities outside this space. When such a vulnerability emerges, however, integrating it requires only a constant-time update to the KG (e.g., adding a new edge), enabling the framework to evolve quickly as new threat intelligence becomes available.

Cost of Knowledge Graph Construction. Constructing the KG remains scalable and cost-efficient even for large codebases. For instance, extracting data and text embeddings for TCP DoS across 2,333 repositories using GPT-4o-mini cost approximately US$13. Beyond monetary cost, KNOWML substantially reduces human effort. We estimate that manual expert analysis of TCP DoS, HTTP DoS, and Brute-Force families would require approximately 164 workdays (7,853 repositories at 10 minutes each over 8-hour workdays). KNOWML automates the bulk of this effort through large-scale analysis of attack implementations.

7. Related Work

KG Applications in Security. KGs have gained traction in cybersecurity, with a range of applications from detection to knowledge management [106]. The most relevant to our work are those that directly integrate KGs into detection pipelines. For example, Yang et al. [96] construct a KG from NSL-KDD feature co-occurrence and embed it into a CNN-BiLSTM. Their approach enhances raw features through data-driven correlations, whereas KNOWML introduces an attack-implementation-based feature space grounded in domain knowledge, rather than manipulations of existing features. Another approach is KnowGraph [104], which models buyer–seller interactions as a KG for anomaly detection, learning embeddings via GNNs and refining predictions with expert-provided probabilistic rules. However, KnowGraph depends on external corrections (expert-defined rules) and does not expand the knowledge base. By contrast, KNOWML embeds foundational attack knowledge directly into the feature space, allowing models to capture the broader attack landscape to improve generalization.

Automated Feature Engineering in Security. Automated feature engineering in security follows two directions. Representation learning derives latent features from data to improve detection [88], [93], [100]. Feature mining uses external knowledge to refine an existing feature space, as in FeatureSmith [105], which enhances Drebin [16] by selecting useful features. KNOWML instead expands the feature space with attack-relevant features grounded in implementation details, creating new semantics rather than filtering existing ones. This approach outperforms prior methods focused only on feature enhancement (§5.4).

Exploring Limitations of ML-NIDS. Recent critiques emphasize two issues: 1) dataset inadequacies [14], [34], [38], [71] and 2) flawed ML design practices [15], [31]. Flood et al. [38] describe "bad smells" in benchmark datasets, such as *poor diversity*, which produces highly dependent features. Such flaws prevent models from learning meaningful representations, limit generalization, and inflate reported performance. In contrast, we analyze limitations from a semantic perspective. We reason about how well features capture attack semantics, i.e., their ability to generalize, how this affects their ability to separate benign from malicious traffic, and how it influences the number of false positive alarms.

8. Conclusion

This paper identifies fundamental gaps in SotA feature spaces used by Anomaly-based ML-NIDS (A-NIDS), showing that they are insufficient for capturing attack variants. To address this gap, we proposed KNOWML, a framework that, given attack families of interest, builds a KG of attack strategies and applies symbolic reasoning to derive a corresponding Knowledge-Augmented Feature Space (KAFS) grounded in attack semantics.

Our evaluation on established benchmarks and new attack variants revealed two categories of knowledge gaps

in existing approaches. We demonstrated that KNOWML closes these gaps, achieving effective generalization while maintaining low false positive rates. These results highlight the importance of incorporating attack-relevant semantics into the feature space of A-NIDS. The findings also have broader implications. They encourage the exploration of semantic reasoning in other security applications and motivate the development of adaptive defenses that evolve with attacker strategies.

Ethical Considerations

We conducted our research ethically, with the principle of beneficence in mind. To the best of our knowledge, neither the work nor its experiments raise ethical concerns, no traffic was collected without user consent, and the CAP dataset was generated on controlled lab machines where the attacks caused no harm.

Acknowledgments

This work was supported by: the UKRI Centre for Doctoral Training in Safe and Trusted Artificial Intelligence (EP/S023356/1), the UK EPSRC Grant no. EP/X015971/2, a Google Academic Research Award (GARA), and Google ASPIRE awards. We thank the anonymous reviewers for their valuable feedback, and Giovanni Apruzzese for providing several attack variants used in this study.

References

[1] CVE-2007-6750: Apache http server partial http requests dos (slowloris). https://nvd.nist.gov/vuln/detail/CVE-2007-6750, 2007. National Vulnerability Database; published December 27, 2011; last updated April 11, 2025.

[2] CVE-2009-1926 detail, 2009.

[3] CVE-2020-35863 detail. https://nvd.nist.gov/vuln/detail/CVE-2020-35863, December 2020. Accessed: 2025-02-26.

[4] Cve-2024-35296 detail. https://nvd.nist.gov/vuln/detail/CVE-2024-35296, July 2024. Accessed: 2025-02-26.

[5] CVE-2025-32472: Denial-of-service via slowloris-type attack on multiscan and picoscan. https://nvd.nist.gov/vuln/detail/CVE-2025-32472, April 2025. National Vulnerability Database; published 2025-04-28; last modified 2025-04-29.

[6] Devrim Akgun, Selman Hizal, and Unal Cavusoglu. A new ddos attacks intrusion detection model based on deep learning for cybersecurity. *Computers & Security*, 118:102748, 2022.

[7] Abdullah Aldaihan, Fahad Alotaibi, and Sergio Maffeis. Clouseau: A hierarchical multi-agent approach for autonomous attack investigation. In *2025 IEEE Annual Computer Security Applications Conference (ACSAC)*, pages 516–532. IEEE, 2025.

[8] Suresh Kumar Amalapuram, Akash Tadwai, Reethu Vinta, Sumohana S Channappayya, and Bheemarjuna Reddy Tamma. Continual learning for anomaly based network intrusion detection. In *2022 14th International Conference on COMmunication Systems & NETworkS (COMSNETS)*, pages 497–505. IEEE, 2022.

[9] Mohammed A. Ambusaidi, Xiangjian He, Priyadarsi Nanda, and Zhiyuan Tan. Building an intrusion detection system using a filter-based feature selection algorithm. *IEEE Transactions on Computers*, 65(10):2986–2998, 2016.

[10] Fatemeh Amiri, MohammadMahdi Rezaei Yousefi, Caro Lucas, Azadeh Shakery, and Nasser Yazdani. Mutual information-based feature selection for intrusion detection systems. *Journal of network and computer applications*, 34(4):1184–1199, 2011.

[11] Peng An, Zhiyuan Wang, and Chunjiong Zhang. Ensemble unsupervised autoencoders and gaussian mixture model for cyberattack detection. *Information Processing & Management*, 59(2):102844, 2022.

[12] Apache Software Foundation. CVE-2025-31650: Improper Input Validation in Apache Tomcat. https://nvd.nist.gov/vuln/detail/CVE-2025-31650, 2025. Published: 2025-04-28, Last Modified: 2025-05-06.

[13] Giovanni Apruzzese, Pavel Laskov, Edgardo Montes de Oca, Wissam Mallouli, Luis Brdalo Rapa, Athanasios Vasileios Grammatopoulos, and Fabio Di Franco. The role of machine learning in cybersecurity. *Digital Threats: Research and Practice*, 4(1):1–38, 2023.

[14] Giovanni Apruzzese, Pavel Laskov, and Johannes Schneider. Sok: Pragmatic assessment of machine learning for network intrusion detection, 2023.

[15] Daniel Arp, Erwin Quiring, Feargus Pendlebury, Alexander Warnecke, Fabio Pierazzi, Christian Wressnegger, Lorenzo Cavallaro, and Konrad Rieck. Dos and don'ts of machine learning in computer security. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 3971–3988, 2022.

[16] Daniel Arp, Michael Spreitzenbarth, Malte Hubner, Hugo Gascon, Konrad Rieck, and CERT Siemens. Drebin: Effective and explainable detection of android malware in your pocket. In *Ndss*, volume 14, pages 23–26, 2014.

[17] Stefan Axelsson. The base-rate fallacy and the difficulty of intrusion detection. *ACM Transactions on Information and System Security (TISSEC)*, 3(3):186–205, 2000.

[18] Diogo Barradas, Nuno Santos, Luís Rodrigues, Salvatore Signorello, Fernando MV Ramos, and André Madeira. Flowlens: Enabling efficient flow classification for ml-based network security applications. In *NDSS*, 2021.

[19] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1798–1828, 2013.

[20] Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy layer-wise training of deep networks. *Advances in neural information processing systems*, 19, 2006.

[21] Sergei Bogdanov, Alexandre Constantin, Timothée Bernard, Benoit Crabbé, and Etienne Bernard. Nuner: entity recognition encoder pre-training via llm-annotated data. *arXiv preprint arXiv:2402.15343*, 2024.

[22] Donald A Burgio. *Reduction of false positives in intrusion detection based on extreme learning machine with situation awareness*. PhD thesis, Nova Southeastern University, 2020.

[23] Cisco Systems, Inc. Cisco ios netflow, 2025. Accessed: 2025-05-25.

[24] CrowdStrike, Inc. Common cyberattacks: Definitions, examples and prevention tips. https://www.crowdstrike.com/en-us/cybersecurity-101/cyberattacks/common-cyberattacks/, 2025. Accessed: 2025-06-04.

[25] National Vulnerability Database. CVE-2009-1926: Tcp/ip null pointer dereference vulnerability. https://nvd.nist.gov/vuln/detail/CVE-2009-1926, 2009. Accessed: 2024-10-20.

[26] Michael Dean and Guus Schreiber. Owl web ontology language guide: Feature synopsis. https://www.w3.org/TR/2004/REC-owl-features-20040210/#s4, 2004. W3C Recommendation, Accessed: 2025-06-01.

[27] Mostafa Dehghani, Yi Tay, Alexey A Gritsenko, Zhe Zhao, Neil Houlsby, Fernando Diaz, Donald Metzler, and Oriol Vinyals. The benchmark lottery. *arXiv preprint arXiv:2107.07002*, 2021.

[28] Kathrin Dentler, Ronald Cornet, Annette Ten Teije, and Nicolette De Keizer. Comparison of reasoners for large ontologies in the owl 2 el profile. *Semantic Web*, 2(2):71–87, 2011.

[29] Jairo Diaz-Rodriguez. k-llmmeans: Summaries as centroids for interpretable and scalable llm-based text clustering. *arXiv preprint arXiv:2502.09667*, 2025.

[30] R. Doriguzzi-Corin, S. Millar, S. Scott-Hayward, J. Martínez-del Rincón, and D. Siracusa. Lucid: A practical, lightweight deep learning solution for ddos attack detection. *IEEE Transactions on Network and Service Management*, 17(2):876–889, 2020.

[31] Laurens D'hooge, Miel Verkerken, Bruno Volckaert, Tim Wauters, and Filip De Turck. Establishing the contaminating effect of metadata feature inclusion in machine-learned network intrusion detection models. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 23–41. Springer, 2022.

[32] Darren Edge, Ha Trinh, Newman Cheng, Joshua Bradley, Alex Chao, Apurva Mody, Steven Truitt, and Jonathan Larson. From local to global: A graph rag approach to query-focused summarization, 2024.

[33] Devon Edwards. The laymans guide to network performance: What are packets per second?, 2025. Secure Compliance Solutions Blog.

[34] Gints Engelen, Vera Rimmer, and Wouter Joosen. Troubleshooting an intrusion detection dataset: the cicids2017 case study. In *2021 IEEE Security and Privacy Workshops (SPW)*, pages 7–12, 2021.

[35] Xuewei Feng, Chuanpu Fu, Qi Li, Kun Sun, and Ke Xu. Off-path tcp exploits of the mixed ipid assignment. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pages 1323–1335, 2020.

[36] Xuewei Feng, Qi Li, Kun Sun, Zhiyun Qian, Gang Zhao, Xiaohui Kuang, Chuanpu Fu, and Ke Xu. {off-path} network traffic manipulation via revitalized {ICMP} redirect attacks. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 2619–2636, 2022.

[37] Mohamed Amine Ferrag, Leandros Maglaras, Sotiris Moschoyiannis, and Helge Janicke. Deep learning for cyber security intrusion detection: Approaches, datasets, and comparative study. *Journal of Information Security and Applications*, 50:102419, 2020.

[38] Robert Flood, Gints Engelen, David Aspinall, and Lieven Desmet. Bad design smells in benchmark nids datasets. In *2024 IEEE 9th European Symposium on Security and Privacy (EuroS&P)*, pages 658–675. IEEE, 2024.

[39] Chuanpu Fu, Qi Li, Meng Shen, and Ke Xu. Realtime robust malicious traffic detection via frequency domain analysis. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 3431–3446, 2021.

[40] Chuanpu Fu, Qi Li, and Ke Xu. Detecting unknown encrypted malicious traffic in real time via flow interaction graph analysis. *arXiv preprint arXiv:2301.13686*, 2023.

[41] Álvaro García-Barragán, Alberto González Calatayud, Oswaldo Solarte-Pabón, Mariano Provencio, Ernestina Menasalvas, and Víctor Robles. Gpt for medical entity recognition in spanish. *Multimedia Tools and Applications*, pages 1–20, 2024.

[42] Hossein Gharaee and Hamid Hosseinvand. A new feature selection ids based on genetic algorithm and svm. In *2016 8th International Symposium on Telecommunications (IST)*, pages 139–144. IEEE, 2016.

[43] Hamed Babaei Giglou, Jennifer D'Souza, Felix Engel, and Sören Auer. Llms4om: Matching ontologies with large language models. *arXiv preprint arXiv:2404.10317*, 2024.

[44] GitHub. GitHub REST API - Search, 2022. Accessed: 2025-06-02.

[45] Brendon Harris and Ray Hunt. Tcp/ip security threats and attack methods. *Computer communications*, 22(10):885–897, 1999.

[46] Mohammad J Hashemi and Eric Keller. Enhancing robustness against adversarial examples in network intrusion detection systems. In *2020 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, pages 37–43. IEEE, 2020.

[47] Hiett. serverless-redis-http. https://github.com/hiett/serverless-redis-http, 2020. Accessed: 2025-06-02.

[48] Yuelin Hu, Futai Zou, Jiajia Han, Xin Sun, and Yilei Wang. Llm-tikg: Threat intelligence knowledge graph construction utilizing large language model. *Computers & Security*, 145:103999, 2024.

[49] Lei Huang, Weijiang Yu, Weitao Ma, Weihong Zhong, Zhangyin Feng, Haotian Wang, Qianglong Chen, Weihua Peng, Xiaocheng Feng, Bing Qin, et al. A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions. *ACM Transactions on Information Systems*, 43(2):1–55, 2025.

[50] Mobin Javed and Vern Paxson. Detecting stealthy, distributed ssh brute-forcing. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 85–96, 2013.

[51] Andrey Kachayev. Fiberfox. https://github.com/kachayev/fiberfox. Accessed: 2025-04-18.

[52] Elisa F Kendall and Deborah L McGuinness. *Ontology engineering*. Morgan & Claypool Publishers, 2019.

[53] Taehoon Kim and Wooguil Pak. Robust network intrusion detection system based on machine-learning with early classification. *IEEE Access*, 10:10754–10767, 2022.

[54] Adrian Komadina, Mislav Martinić, Stjepan Groš, and Željka Mihajlović. Comparing threshold selection methods for network anomaly detection. *IEEE access*, 2024.

[55] Yuri Kuratov, Aydar Bulatov, Petr Anokhin, Dmitry Sorokin, Artyom Sorokin, and Mikhail Burtsev. In search of needles in a 10m haystack: Recurrent memory finds what llms miss. *arXiv preprint arXiv:2402.10790*, 2024.

[56] Aleksandar Kuzmanovic and Edward W Knightly. Low-rate tcp-targeted denial of service attacks: the shrew vs. the mice and elephants. In *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 75–86, 2003.

[57] Lanjelot. Patator: A multi-purpose brute-forcer. https://github.com/lanjelot/patator/blob/master/src/patator/patator.py, 2025. Accessed: 2025-04-20.

[58] XuKui Li, Wei Chen, Qianru Zhang, and Lifa Wu. Building auto-encoder intrusion detection system based on random forest feature selection. *Computers & Security*, 95:101851, 2020.

[59] David Lion, Adrian Chiu, Michael Stumm, and Ding Yuan. Investigating managed language runtime performance: Why {JavaScript} and python are 8x and 29x slower than c++, yet java and go can be faster? In *2022 USENIX Annual Technical Conference (USENIX ATC 22)*, pages 835–852, 2022.

[60] Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173, 2024.

[61] Bonan Min, Hayley Ross, Elior Sulem, Amir Pouran Ben Veyseh, Thien Huu Nguyen, Oscar Sainz, Eneko Agirre, Ilana Heintz, and Dan Roth. Recent advances in natural language processing via large pre-trained language models: A survey. *ACM Comput. Surv.*, 56(2), September 2023.

[62] Yisroel Mirsky. Kitsune-py: A network intrusion detection system based on incremental statistics (afterimage) and an ensemble of autoencoders (kitnet). https://github.com/ymirsky/Kitsune-py, 2018. Accessed: 2025-04-15.

[63] Yisroel Mirsky, Tomer Doitshman, Yuval Elovici, and Asaf Shabtai. Kitsune: an ensemble of autoencoders for online network intrusion detection. *arXiv preprint arXiv:1802.09089*, 2018.

[64] MITRE Corporation. CVE-2008-4609: Tcp implementation vulnerability. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-4609.

[65] Nour Moustafa and Jill Slay. Unsw-nb15: a comprehensive data set for network intrusion detection systems (unsw-nb15 network data set). In *2015 military communications and information systems conference (MilCIS)*, pages 1–6. IEEE, 2015.

[66] Daniel Müllner. Modern hierarchical, agglomerative clustering algorithms. *arXiv preprint arXiv:1109.2378*, 2011.

[67] National Institute of Standards and Technology. CVE-2024-32984: Yamux stream multiplexer unbounded pending frames vulnerability. https://nvd.nist.gov/vuln/detail/CVE-2024-32984.

[68] National Vulnerability Database. CVE-2021-31166: HTTP Protocol Stack Remote Code Execution Vulnerability. https://nvd.nist.gov/vuln/detail/CVE-2021-31166, 2021. Accessed: 2025-04-18.

[69] National Vulnerability Database. CVE-2024-39316: Regular Expression Denial of Service (ReDoS) in Rack::Request::Helpers. https://nvd.nist.gov/vuln/detail/CVE-2024-39316, 2024. Accessed: 2025-04-10.

[70] Euclides Carlos Pinto Neto, Sajjad Dadkhah, Raphael Ferreira, Alireza Zohourian, Rongxing Lu, and Ali A Ghorbani. Ciciot2023: A real-time dataset and benchmark for large-scale attacks in iot environment. *Sensors*, 23(13):5941, 2023.

[71] Benoit Nougnanke, Gregory Blanc, and Thomas Robert. How dataset diversity affects generalization in ml-based nids. In *ESORICS 2025-30th European Symposium on Research in Computer Security*, 2025.

[72] OpenAI. Structured outputs with language models. https://cookbook.openai.com/examples/structured_outputs_intro, 2023. Accessed: 2025-06-02.

[73] OpenAI. Embedding models, 2024. Accessed: 2025-06-06.

[74] Fabio Pierazzi, Feargus Pendlebury, Jacopo Cortellazzi, and Lorenzo Cavallaro. Intriguing properties of adversarial ml attacks in the problem space. In *2020 IEEE symposium on security and privacy (SP)*, pages 1332–1349. IEEE, 2020.

[75] Naveen Mohan Prajapati, Atish Mishra, and Praveen Bhanodia. Literature survey-ids for ddos attacks. In *2014 Conference on IT in Business, Industry and Government (CSIBIG)*, pages 1–3. IEEE, 2014.

[76] LLC QoSient. Argus + ml: Argus and machine learning. https://openargus.org/argus-ml, 2025. Accessed: 2025-11-14.

[77] Sandeepkumar Racherla, Prathyusha Sripathi, Nuruzzaman Faruqui, Md Alamgir Kabir, Md Whaiduzzaman, and Syed Aziz Shah. Deep-ids: A real-time intrusion detector for iot nodes using deep learning. *IEEE Access*, 12:63584–63597, 2024.

[78] Salah Rifai, Pascal Vincent, Xavier Muller, Xavier Glorot, and Yoshua Bengio. Contractive auto-encoders: Explicit invariance during feature extraction. In *Proceedings of the 28th international conference on international conference on machine learning*, pages 833–840, 2011.

[79] Oscar Sainz, Iker García-Ferrero, Rodrigo Agerri, Oier Lopez de Lacalle, German Rigau, and Eneko Agirre. Gollie: Annotation guidelines improve zero-shot information-extraction. *arXiv preprint arXiv:2310.03668*, 2023.

[80] Mohanad Sarhan, Siamak Layeghy, and Marius Portmann. Towards a standard feature set for network intrusion detection system datasets. *Mobile networks and applications*, pages 1–14, 2022.

[81] Saeed Shafieian and Mohammad Zulkernine. Multi-layer stacking ensemble learners for low footprint network intrusion detection. *Complex & Intelligent Systems*, 9(4):3787–3799, 2023.

[82] Iman Sharafaldin, Arash Habibi Lashkari, Ali A Ghorbani, et al. Toward generating a new intrusion detection dataset and intrusion traffic characterization. *ICISSp*, 1:108–116, 2018.

[83] Mohammad Shurman, Rami Khrais, Abdulrahman Yateem, et al. Dos and ddos attack detection using deep learning and ids. *Int. Arab J. Inf. Technol*, 17(4A):655–661, 2020.

[84] Mahdi Soltani, Behzad Ousat, Mahdi Jafari Siavoshani, and Amir Hossein Jahangir. An adaptable deep learning-based intrusion detection system to zero-day attacks. *Journal of Information Security and Applications*, 76:103516, 2023.

[85] Robin Sommer and Vern Paxson. Outside the closed world: On using machine learning for network intrusion detection. In *2010 IEEE symposium on security and privacy*, pages 305–316. IEEE, 2010.

[86] Cristian-Alexandru Staicu and Michael Pradel. Freezing the web: a study of {ReDoS} vulnerabilities in {JavaScript-based} web servers. In *27th USENIX security symposium (USENIX Security 18)*, pages 361–376, 2018.

[87] Ankit Thakkar and Ritika Lohiya. A review on machine learning and deep learning perspectives of ids for iot: recent updates, security issues, and challenges. *Archives of Computational Methods in Engineering*, 28(4):3211–3243, 2021.

[88] Mati Ur Rehman, Hadi Ahmadi, and Wajih Ul Hassan. Flash: A comprehensive approach to intrusion detection via provenance graph representation learning. In *2024 IEEE Symposium on Security and Privacy (SP)*, pages 3552–3570, 2024.

[89] GitHub user ella collins. Issue #12: The number of features in the code is not the same as in the paper. https://github.com/ymirsky/Kitsune-py/issues/12, 2021. Accessed: 2025-02-25.

[90] Mathy Vanhoef. Fragment and forge: breaking {Wi-Fi} through frame aggregation and fragmentation. In *30th USENIX security symposium (USENIX Security 21)*, pages 161–178, 2021.

[91] Miel Verkerken, Laurens D'hooge, Bruno Volckaert, Filip De Turck, and Giovanni Apruzzese. Concap: Practical network traffic generation for flow-based intrusion detection systems. *arXiv preprint arXiv:2509.16038*, 2025.

[92] David Wagner and Paolo Soto. Mimicry attacks on host-based intrusion detection systems. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, pages 255–264, 2002.

[93] Wei Wang, Songlei Jian, Yusong Tan, Qingbo Wu, and Chenlin Huang. Representation learning-based network intrusion detection system by capturing explicit and implicit feature interactions. *Computers & Security*, 112:102537, 2022.

[94] Barry Payne Welford. Note on a method for calculating corrected sums of squares and products. *Technometrics*, 4(3):419–420, 1962.

[95] Daan Willems, Katharina Kohls, Bob van der Kamp, and Harald Vranken. Data exfiltration detection on network metadata with autoencoders. *Electronics*, 12(12):2584, 2023.

[96] Xiuzhang Yang, Guojun Peng, Dongni Zhang, and Yangqi Lv. An enhanced intrusion detection system for iot networks based on deep learning and knowledge graph. *Security and Communication Networks*, 2022(1):4748528, 2022.

[97] Yubo Zhai and Xianghan Zheng. Random forest based traffic classification method in sdn. In *2018 international conference on cloud computing, big data and blockchain (ICCBB)*, pages 1–5. IEEE, 2018.

[98] Hongpo Zhang, Lulu Huang, Chase Q Wu, and Zhanbo Li. An effective convolutional neural network based on smote and gaussian mixture model for intrusion detection in imbalanced dataset. *Computer Networks*, 177:107315, 2020.

[99] Jing Zhang, Bo Chen, Lingxi Zhang, Xirui Ke, and Haipeng Ding. Neural, symbolic and neural-symbolic reasoning on knowledge graphs. *AI Open*, 2:14–35, 2021.

[100] Xinchen Zhang, Running Zhao, Zhihan Jiang, Zhicong Sun, Yulong Ding, Edith C. H. Ngai, and Shuang-Hua Yang. AOC-IDS: Code for "AOC-IDS: Autonomous Online Framework with Contrastive Learning for Intrusion Detection". https://github.com/xinchen930/AOC-IDS, 2024. Accessed: 2025-08-20.

[101] Xinchen Zhang, Running Zhao, Zhihan Jiang, Zhicong Sun, Yulong Ding, Edith C.H. Ngai, and Shuang-Hua Yang. Aoc-ids: Autonomous online framework with contrastive learning for intrusion detection. In *IEEE INFOCOM 2024 - IEEE Conference on Computer Communications*, pages 581–590, 2024.

[102] Ziming Zhao, Zhaoxuan Li, Zhuoxue Song, Wenhao Li, and Fan Zhang. Trident: A universal framework for fine-grained and class-incremental unknown traffic detection. In *Proceedings of the ACM Web Conference 2024*, pages 1608–1619, 2024.

[103] Ziming Zhao, Zhaoxuan Li, Zhuoxue Song, Wenhao Li, and Fan Terry Zhang. Trident: Code for "Trident: A Universal Framework for Fine-Grained and Class-Incremental Unknown Traffic Detection". https://github.com/Secbrain/Trident/tree/main/code, 2024. Accessed: 2025-08-20.

[104] Andy Zhou, Xiaojun Xu, Ramesh Raghunathan, Alok Lal, Xinze Guan, Bin Yu, and Bo Li. Knowgraph: Knowledge-enabled anomaly detection via logical reasoning on graph data. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, pages 168–182, 2024.

[105] Ziyun Zhu and Tudor Dumitraş. Featuresmith: Automatically engineering features for malware detection by mining the security literature. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pages 767–778, 2016.

[106] Xiaohan Zou. A survey on application of knowledge graph. In *Journal of Physics: Conference Series*, volume 1487, page 012016. IOP Publishing, 2020.

Appendix A. Repositories Extracted for KG

To systematically investigate the attack mutation space, we conducted a comprehensive search of GitHub repositories containing relevant implementations. We began by constructing a base set of keywords for each attack, then expanded these keywords with variations to capture different naming conventions and implementation styles. Each combination was used as a query to search GitHub, and the resulting repositories were collected into a candidate set. This structured process ensured broad coverage of potential attack implementations and minimized the risk of overlooking relevant variants. A list of searched keywords are given in Table 7.

Appendix B. Evaluating LLM Effectiveness in Strategy Extraction

We evaluate KNOWML 's ability to enumerate and extract attack strategies through a Named Entity Recognition (NER) task, as introduced in §4.2, since accurate identification of strategy entities from heterogeneous repository documentation is essential for constructing the attack strategy KG. From a corpus of 7,853 open source implementations across three attack families, we sample 150 repositories for detailed evaluation, balancing annotation cost and diversity. The sample covers four representative scenarios observed in practice: structured documentation, unstructured documentation, empty inputs to test robustness against hallucinations [49], and irrelevant content from mistakenly retrieved auxiliary repositories (for example, an HTTP Redis pooler returned for an HTTP DoS query [47]).

We compare three models, GPT 3.5 and GPT 4o, which are general-purpose LLMs shown to be effective in security-related extraction tasks [48], and Gollie [79], a task-specific NER model. Following standard practice [21], [41], [79], we report precision, recall and F1 score (Table 8), where Recall measures coverage of human-annotated strategy entities and Precision measures agreement with those annotations. GPT 4o attains the highest Recall at 90.91%, indicating strong coverage of relevant strategies across diverse documentation formats. Precision may appear lower, at 53.16%; however, this estimate should be interpreted as a conservative lower bound rather than as evidence of a fundamental limitation. We adopt "exact string matching" for evaluation, meaning semantically equivalent extractions are counted as false positives (e.g., "Port Randomisation" vs. "Random Port Selection"). We deliberately report this worst-case metric, as (to the best of our knowledge) no established measure of semantic closeness exists that avoids introducing subjective bias. In practice, semantically-similar strategies are merged via embedding-based clustering (§4.2.3), substantially reducing redundancy with minimal manual effort. Hallucinations are further mitigated through fixed seeds and structured output schemas (§4.2.2). In our setting, high Recall is crucial, because missed strategies create permanent gaps in the KG and limit downstream generalization, whereas superfluous-but-related entities can be pruned at low manual cost, yielding higher effective Precision after clustering.

Algorithm 1 Register Network Statistics at Time t

function UPDATE($CS_{t-x}, W_{t-x}, SSR_{t-x}, \sigma_{t-x}, x_i$)
  $CS_t \leftarrow CS_{t-x} + x_i$
  $W_t \leftarrow W_{t-x} + 1$
  $r_t \leftarrow \frac{CS_t}{W_t}$
  $SSR_t \leftarrow SSR_{t-x} + (x_i - r_t)^2$
  $\sigma_t \leftarrow \sqrt{\frac{SSR_t}{W_t}}$
  return $CS_t, W_t, r_t, SSR_t, \sigma_t$
end function

Appendix C. Feature Extraction Algorithm

For completeness, we provide the two core algorithms used in KNOWML's feature extraction procedure mentioned in §4.4. Algorithm 1 specifies the update rule for a single feature statistic. Given a new observation x_i, it recursively updates the cumulative sum (CS), sample count (W), running mean (r), sum of squared residuals (SSR), and standard deviation (σ), without storing the entire history of observations. This formulation combines Welford's algorithm [94] with the approach in Kitsune [63], enabling constant-time updates with minimal memory requirements. The complete update algorithm is given in 2.

Building on this primitive, 2 describes the end-to-end packet-level workflow for online feature extraction. For each packet, it retrieves the relevant feature spaces associated with the source–destination channel and the destination host, updates their statistics by repeatedly invoking 1, and computes incremental values to capture short-term dynamics. The updated feature vectors are then returned for subsequent analysis. In summary, 1 provides the atomic update mechanism for individual features, while 2 integrates this mechanism into the complete packet-processing pipeline, enabling efficient and scalable extraction of descriptive statistics from network traffic in real time.

Appendix D. Overview of Extracted Features

Table 9 summarizes the extracted feature space used in our experiments. The features were designed according to the rules specified in KNOWML, covering a broad range of traffic characteristics including flow-level metrics, packet size statistics, temporal dynamics, and protocol-specific behaviors. In addition to low-level features (e.g., TCP flags, inter-arrival times, retransmissions),

TABLE 7: Attack keywords used in search and number of repositories found

Attack Name	Base Keywords	Variations	Repos
TCP DoS	TCP	Flood, DoS, Denial of Service, Attack	2333
HTTP DoS	HTTP, HTTP GET, HTTP POST, HTTP Request	Flood, DoS, Denial of Service, Attack, Bombardment, Overload	2935
Brute Force	SSH Dictionary, SSH Brute Force, SSH Brute-Force, SSH Password Guessing, SSH Password Cracking, SSH, SSH Login Guessing, SSH Password Spraying	Attack, Technique, Method, Tool, Hack, Crack, Attempt, Threat, Vulnerability, Exploit, Breach, Brute Force, Dictionary Attack	2585

TABLE 8: Performance results for the Strategy-NER task.

Model	Precision	Recall	F1-Score
GPT-4o mini	0.5316	**0.9091**	0.6709
GPT-3.5 Turbo	0.5981	0.5541	0.5753
Gollie 13B	0.5417	0.0563	0.1020

Algorithm 2 Online Algorithm to Extract Relevant Feature Values for Packet at Time t

function EXTRACTFEATURES(packet p)
  $H_1, H_2 \leftarrow p.identifiers$ ▷ H_1: source identifier, H_2: destination identifier
  $V_{H_1,H_2} \leftarrow V_{channel}(H_1, H_2)$
  $V_{H_2} \leftarrow V_{destination}(H_2)$
  $\Delta V \leftarrow [\,]$
  for all v in V_{H_1,H_2} **do**
    $x_i \leftarrow p.f_i$ ▷ Get corresponding value to update
    $W_{t-x}, CS_{t-x}, r_{t-x}, SSR_{t-x}, \sigma_{t-x} \leftarrow v$
    $\Delta CS \leftarrow CS_{t-x}$ ▷ Store value at previous timestep
    $W_t, CS_t, r_t, SSR_t, \sigma_t \leftarrow$ UPDATE$(CS_{t-x}, W_{t-x}, SSR_{t-x}, \sigma_{t-x}, x_i)$
    $v \leftarrow W_t, CS_t, r_t, SSR_t, \sigma_t$
    $\Delta CS \leftarrow |CS_t - \Delta CS|$
    $\Delta V \leftarrow \Delta V \cup \{\Delta CS\}$
  end for
  for all (i, v) in ENUMERATE(V_{H_2}) **do**
    $x_i \leftarrow \Delta V[i]$ ▷ Get change in value
    $W_{t-x}, CS_{t-x}, r_{t-x}, SSR_{t-x}, \sigma_{t-x} \leftarrow v$
    $W_t, CS_t, r_t, SSR_t, \sigma_t \leftarrow$ UPDATE$(CS_{t-x}, W_{t-x}, SSR_{t-x}, \sigma_{t-x}, x_i)$
    $v \leftarrow W_t, CS_t, r_t, SSR_t, \sigma_t$ ▷ Store updated statistics
  end for
  return $V_{H_1,H_2} \| V_{H_2}$ ▷ Return updated feature values
end function

we also incorporate higher-level protocol semantics (e.g., HTTP request methods, SSH authentication counts) to capture application-layer behaviors.

Appendix E. Hyperparameter Selection

We performed grid search over the parameters listed in Table 10. Each model was tuned to maximize Recall under the constraint FPR $\leq 0.1\%$, selecting the lowest FPR that achieved at least 90% Recall. This avoids the "benchmark lottery" [27], ensuring that performance reflects the method rather than arbitrary hyperparameters. For GMM, we varied $n\_components \in 3, 5, 7, 10, 20, 30, 40$ and covariance type $\in$ `diag`, `spherical` (14 combinations). For DA, we searched over hidden ratios $0.1, 0.3, 0.5$, learning rates $0.0001, 0.001$, corruption levels $0.05, 0.1, 0.2$, activations `relu`, `tanh`, and l_2 regularization $0.0001, 0.001$ (72 combinations). For EoA, we tuned maxAE $1, 10$, learning rate $0.001, 0.01$, and hidden ratio $0.25, 0.5$ (8 combinations). This was repeated $2\times$ for different datasets.

Appendix F. Additional Dataset Details

In this paper, we evaluate attack variants organized by the categories defined in §2. The breakdown of evaluated and collected variants is presented in Table 11. Note that some variants overlap and do not fall into a single category. In the remainder of this section, we describe the collection and simulation of the variants used in this study.

F.1. Simulated Attacks

We simulated 4 attack scenarios, applied to benign traffic from both the CIoT-23 and CIDS-17 datasets, including the aforementioned Nkiller2 attack. Each attack represents a real-world *known* vulnerability. The simulations were conducted by taking an hour of benign traffic. For each simulation we forced at least a 2x increase in rate compared to the original benign traffic (if not stated otherwise). The goal is to show that even with an increase in the average byte rate, those attacks still cannot be captured due to the insufficient discriminative power of the feature space. The configuration and setup details are given as follows:

1) **Low-rate TCP**: Throttling the TCP SYN attack by sending packets in bursts of 1 second and an idle time of 1 second. Simulating a well-established Low-Rate TCP Targeted attack [56] targeting the TCP Retransmission Timeout (RTO) mechanism.
2) **Nkiller2**: Setting TCP window size of TCP packets to 0 and sending packets at 2.2x rate as the benign traffic. Simulating the vulnerability [2], [64], [67].
3) **HTTP DoS (HTTP Overflow)**: Setting HTTP Accept-Language to overflow a long valid string, causing the server to spend excessive time processing (as described in [69]). The packets are sent at 2.2x rate as benign traffic.

TABLE 9: **KAFS Feature Space of KNOWML**. Features for HTTP, TCP DoS and Brute Force. ICMP features are included due to reconnaissance strategies found in examined repositories. The feature space is organized according to their underlying characteristics.

Category	Statistics (Examples)	# Features	Description
Connection Metrics	Duration, keep-alive count, idle connections	8	Connection lifespan and persistence patterns
Packet Volume	Total count, size, inbound/outbound rates	18	Directional packet counts and transmission rates
Inter-Arrival Time	Total time, minimum, maximum, variation	8	Temporal spacing between packets
Throughput	Transmission rate, throughput	4	Data transfer rates (bytes/sec)
TCP Flags	SYN, ACK, RST, FIN, PSH, URG, XMAS, NULL	16	Control flags indicating connection state and anomalies
TCP State	State distribution (0–6), current state	8	TCP state machine progression
TCP Flow Values	Window size, header values	28	Receiver buffer management and congestion signals
TCP Quality	Retransmissions, packet loss, checksums, fragments	12	Transmission reliability and error indicators
Time-to-Live (TTL)	TTL (average, spread, min, max), packet count	14	IP hop count distribution (routing path)
HTTP Methods	GET, POST, PUT, DELETE, HEAD, PATCH, OPTIONS, TRACE	16	Request type distribution
HTTP Status	Success (2xx), client error (4xx), server error (5xx)	6	Response code categories
HTTP Statistics	Payload size, header size (average, spread, count)	18	HTTP message size statistics
HTTP Protocol Anomalies	Malformed headers, HTTP/2 frames (RST_STREAM, GOAWAY)	22	Protocol violations and HTTP/2-specific events
HTTP Timing	Connection rate, authentication attempts	4	Application-layer session dynamics
SSH Protocol	Packet size/count (average, spread), key exchange attempts	20	SSH session establishment and authentication patterns
ICMP	Packet count, error messages, ping requests	6	Network diagnostics and reconnaissance indicators
Request/Response Timing	Request interval, response interval	4	Application-layer interaction timing
Connection Dynamics	Establishment/termination attempts per second	4	Connection lifecycle event rates

Total: 214 features (108 channel-level + 106 destination-level).

TABLE 10: Grid search parameters and number of combinations for each model, tested at FPR_{te}=0, 0.01, 0.1%.

Model	Parameters	Combinations
GMM	`n_components`: [3, 5, 7, 10, 20, 30, 40] `covariance_type`: [diag, spherical]	14
DA	`hidden_ratio`: [0.1, 0.3, 0.5] `learning_rate`: [0.0001, 0.001] `corruption_level`: [0.05, 0.1, 0.2] `activation`: [relu, tanh] `l2_reg`: [0.0001, 0.001]	72
EoA	`maxAE`: [1, 10] `learning_rate`: [0.001, 0.01] `hidden_ratio`: [0.25, 0.5]	8

4) **HTTP DoS (Mal Header)**: Setting HTTP Accept-Encoding to manifold of invalid values which can cause the kernel to crash, simulating vulnerability [12], [68]. The packets are sent at 2.2x rate as benign traffic.

F.2. Captured Attacks

The CAP dataset was collected in a controlled but realistic setting using two heterogeneous hosts: a MacBook running Sonoma 14.1 and a Dell PC running Ubuntu 14. The Dell PC was configured with an `nginx` HTTP service when HTTP DoS attacks were executed. The Dell machine acted as the victim, and all traffic was captured directly on the victim using Wireshark to minimize measurement artifacts and ensure that the traces reflect the actual network load experienced by the service. The MacBook acted as the attacker and generated malicious traffic by replaying five different HTTP DoS attack strategies using Fiberfox [51].

To avoid dataset bias and ensure realism, we (i) used commodity hardware and standard operating systems instead of emulators, (ii) deployed widely used services (nginx for HTTP) with default configurations, and (iii)

TABLE 11: Summary of Attack Variants used for evaluation.

Category	Variant Name	Dataset
Out-of-Dimension	HTTP Overflow	CAP
	HTTP Mal	CAP
	Nkiller2	CAP
	Brute Force	IoT-23
	Brute Force	CIDS-17
	Brute Force (P=0)	Con
	DoS Golden Eye	CIDS-17
	DoS Hulk	CIDS-17
Non-Throughput	DoS-Slowloris	CIDS-17
	DDoS-Slowloris	CIoT-23
	DoS-Slowhttptest	CIDS-17
	Fiberfox AVB	CAP
	Fiberfox BYPASS	CAP
	Low-rate TCP	CAP
	Brute Force (P=1)	Con
	Fiberfox GET	CAP
Composite	DoS-HTTP	CIoT-23
	DDoS-HTTP	CIoT-23
	DoS-SYN	CIoT-23
	DDoS-SYN	CIoT-23
	DoS-TCP	CIoT-23
	DDoS-TCP	CIoT-23
	DDoS-PSHACK	CIoT-23
	DDoS-RSTFIN	CIoT-23
	DDoS-SynIP	CIoT-23
	DDoS-ACK Frag	CIoT-23
	Fiberfox SLOW	CAP
	Fiberfox STRESS	CAP

allowed background system processes and benign traffic to coexist with attacks during collection. This setup prevents artificial isolation of attack traffic and preserves natural protocol dynamics such as connection establishment, response codes, and timing behavior. In addition, we avoided synthetic traffic generation tools beyond the attack scripts themselves, thereby reducing the risk of artifacts in packet structure or timing distributions. Each attack strategy was executed for 20 minutes (as specified in the `README.md` of Fiberfox [51]), resulting in five distinct attack scenarios:

1) `--AVB`: Issues HTTP GET packets into an open connection with long delays between send operations.
2) `--GET`: Sends randomly generated HTTP GET requests over an open TCP connection.
3) `--STRESS`: Sends a sequence of HTTP requests with a large body over a single open TCP connection.
4) `--BYPASS`: Sends HTTP GET requests over an open TCP connection and reads responses back.
5) `--SLOW`: Similar to STRESS, but issues HTTP requests while keeping the connection active by reading back a single byte and sending additional payload with timed delays.

Con. This dataset was provided by external collaborators (not produced as part of this work) and was collected using SSH Patator [57] to evaluate generalization against SSH variants. `--SSH_PERSISTENT`: If set to 0, it creates a new connection for each attempt; otherwise, it sends multiple brute force attempts through the same connection until termination.

TABLE 12: Comparison of Kitsune Before and After Fix at FPR=0 on the Kitsune Mirai dataset

Metric	Kitsune Before Fix	Kitsune After Fix
True Positives (TP)	560,735	565,390
True Negatives (TN)	66,620	66,620
False Positives (FP)	0	0
False Negatives (FN)	81,781	77,126
Precision	1.0000	1.0000
Recall	0.8727	0.8791 ↑ **(+0.0064)**
F1-score	0.9320	0.9361 ↑ **(+0.0041)**

Appendix G. Patching Kitsune

When implementing baselines, we referred to the Kitsune open-source implementation at [62]. We observed an exponential increase in runtime during the feature extraction process in Kitsune. Our analysis revealed that the covariance update underwent an unintended double decay, which deviates from the algorithm described in the original paper [63], leading to the runtime growth. To ensure fair comparison, we corrected this issue, and all experiments in this work were conducted using the patched version of Kitsune. Table 12 presents performance results on the original Mirai dataset. Our results are comparable to those reported by Arp et al. [15], but differ from those in the original Kitsune paper, as a number of features had been removed by the authors [89].