PIDSMaker: Building and Evaluating Provenance-based Intrusion Detection Systems

Tristan Bilot
University of British Columbia
Vancouver, British Columbia, Canada
tbilot@cs.ubc.ca

Baoxiang Jiang
Xi'an Jiaotong University
Xi'an, China
comlan@stu.xjtu.edu.cn

Thomas Pasquier
University of British Columbia
Vancouver, British Columbia, Canada
tfjmp@cs.ubc.ca

Abstract

Recent provenance-based intrusion detection systems (PIDSs) have demonstrated strong potential for detecting advanced persistent threats (APTs) by applying machine learning to system provenance graphs. However, evaluating and comparing PIDSs remains difficult: prior work uses inconsistent preprocessing pipelines, non-standard dataset splits, and incompatible ground-truth labeling and metrics. These discrepancies undermine reproducibility, impede fair comparison, and impose substantial re-implementation overhead on researchers. We present PIDSMaker, an open-source framework for developing and evaluating PIDSs under consistent protocols. PIDSMaker consolidates eight state-of-the-art systems into a modular, extensible architecture with standardized preprocessing and ground-truth labels, enabling consistent experiments and apples-to-apples comparisons. A YAML-based configuration interface supports rapid prototyping by composing components across systems without code changes. PIDSMaker also includes utilities for ablation studies, hyperparameter tuning and visualization, addressing methodological gaps identified in prior work. By open-sourcing the framework and releasing preprocessed versions of widely used datasets, PIDSMaker lowers the entry barrier for new researchers and facilitates reproducible research in the field.

Code: https://github.com/ubc-provenance/PIDSMaker
Docs: https://ubc-provenance.github.io/PIDSMaker

Note: This is a preprint version of the paper accepted at the 32nd SIGKDD Conference on Knowledge Discovery and Data Mining (KDD'26) [11].

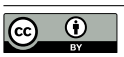
1 Introduction

Modern operating systems continuously generate detailed logs of system activity, recording how processes interact with files, network connections, and other system resources. *Provenance-based security* leverages these logs by organizing them into directed graphs—called *provenance graphs*—where nodes represent system entities (e.g., processes, files, and sockets) and edges capture the causal relationships between them (e.g., a process reading a file or spawning another process). This graph-based representation preserves the temporal ordering and dependencies of system events, providing a rich audit trail that enables security analysts to trace the origin and propagation of suspicious activity across an entire system [7, 9, 10, 13, 19, 21–23, 26, 28–30, 32–34, 36, 37, 40, 42, 50, 57–63, 67]. This makes provenance graphs particularly well-suited for detecting sophisticated, multi-stage attacks such as advanced persistent threats (APTs)—stealthy intrusions in which an adversary maintains prolonged, unauthorized access to a network while progressively exfiltrating data or escalating privileges [21, 44].

Building on this representation, recent provenance-based intrusion detection systems (PIDSs) employ machine learning to automatically distinguish benign from malicious behavior [10, 13, 19, 23, 29, 30, 37, 50, 59]. Most state-of-the-art PIDSs adopt a *self-supervised, anomaly-based* approach: they learn normal system behavior from benign data alone and flag deviations as potential intrusions, enabling the detection of previously unseen threats—including zero-day exploits and novel APT campaigns—without overfitting to known attack signatures. Graph neural networks (GNNs) have become the architecture of choice because they can natively operate on the graph structure of provenance data, capturing fine-grained structural patterns that simpler models miss. Despite reporting strong detection results, many of these systems face practical barriers that limit real-world adoption and hinder reproducibility [4, 10, 16].

A primary challenge is *inconsistent evaluation practices*. Even on the same datasets, evaluations differ in preprocessing, graph construction, and feature extraction. Dataset splits also lack a shared protocol: prior work varies temporal boundaries, hosts, and the mix of attack scenarios. Ground-truth labeling is similarly inconsistent. Prior work adopts heterogeneous labeling strategies: some mark entire neighborhoods [29, 50, 59] or temporal batches [13] as malicious, others label descendants of known malicious nodes [19], while some directly annotate individual nodes [10, 30, 37]. These choices can substantially inflate or depress reported metrics [30], resulting in unfair comparisons across studies. A recent reproducibility work [4] corroborates these concerns, reporting that many published results could not be reproduced due to missing code, data, or critical implementation details.

A second challenge is the *high cost of re-implementation*. New PIDSs are often built from scratch, requiring repeated effort to parse and preprocess provenance data, implement graph construction and feature-extraction pipelines, develop training and inference code, write evaluation scripts, and re-integrate baselines. This duplication is wasteful and error-prone: small implementation or tuning differences can skew results, potentially leading to unfair comparisons due to implementation issues or insufficient tuning [6, 10].



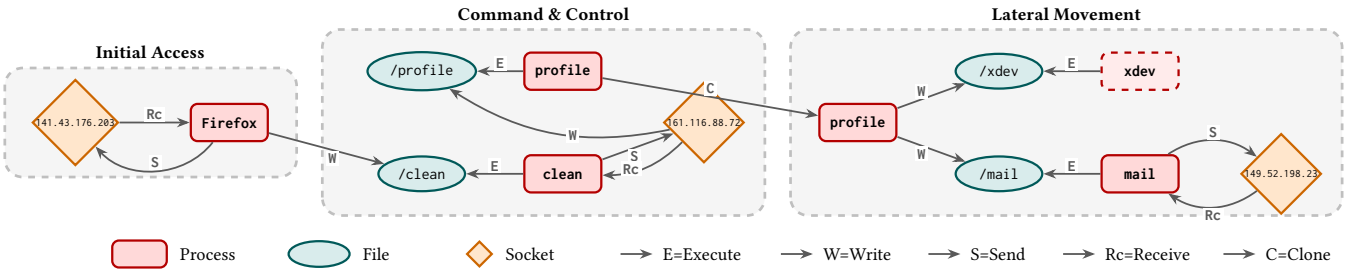


Figure 1: Provenance graph of a multi-stage APT attack from the E3-THEIA dataset [2]. The three stages illustrate how attackers progressively compromise a system. The dashed `xdev` process indicates a dormant payload awaiting activation.

Prior work has also raised methodological concerns beyond evaluation inconsistency and engineering overhead [10]. Neural network–based systems are often sensitive to hyperparameters, yet baseline methods are not always tuned with the same rigor as the proposed approach, which can bias reported gains. Moreover, the contribution of individual components within complex architectures often remains unclear due to limited ablation studies.

To address these issues, we introduce PIDSMaker, a unified framework for developing and evaluating PIDSs under identical protocols. PIDSMaker grew out of the experimental infrastructure we developed for Orthrus [30] and Velox [10]; this paper generalizes that infrastructure into a reusable, community-facing framework. It integrates eight state-of-the-art systems within a modular architecture that enables consistent evaluation and rapid prototyping through standardized preprocessing, ground-truth labels, and a YAML-based configuration interface for composing components across systems without code changes. Built-in support for ablation studies, hyperparameter tuning, and visualization further addresses methodological gaps identified in prior work.

Contributions

- **Consistent evaluation framework.** We identify evaluation inconsistencies across recent PIDSs [10, 13, 19, 29, 30, 37, 50, 59] and introduce a unified framework that consolidates these architectures into a single, modular codebase, enabling consistent evaluation and fair comparison.
- **Config-driven prototyping.** Our YAML-based configuration system enables researchers to assemble new PIDSs by mixing and matching existing components without writing code, reducing development time and supporting systematic design-space exploration.
- **Experimental utilities.** We include integrated support for ablation studies, hyperparameter tuning, and visualization, addressing key methodological gaps identified in prior work [6, 10].
- **Open-source release.** We release PIDSMaker as open-source software, together with preprocessed datasets and ground-truth labels, to serve as a common evaluation platform for the community and encourage contributions to the framework.

2 Background

We review provenance graphs and the common architectural patterns of PIDSs that inform the design of PIDSMaker.

2.1 Provenance Graphs

Modern operating systems continuously log low-level events—such as process creation, file access, and network connections—for auditing and compliance purposes. A *provenance graph* organizes these events into a directed graph $G = (V, E)$ that captures causal relationships among system entities. Nodes $v \in V$ represent entities such as processes, files, and network sockets, while edges $e \in E$ represent interactions between them, such as a process reading from a file or sending data to a remote host. Figure 1 illustrates a provenance graph capturing a multi-stage attack, where rectangular nodes denote processes, ellipses denote files, and diamonds denote network sockets.

Nodes and edges carry attributes that provide semantic context. For example, a process node may include the executable path and command-line arguments, while edges are annotated with the operation type (e.g., `read`, `write`, `execute`) and a timestamp. These attributes enable analysts to reconstruct the sequence of events leading to a given system state and are often used as node and edge features provided as input to neural networks for detection.

Provenance graphs are derived from system audit logs recorded by kernel-level auditing frameworks [8, 47, 48], dynamic instrumentation [53], or hardware-assisted tracing [54, 66]. In production environments, these graphs grow rapidly, often accumulating millions of nodes and edges over days or weeks of operation.

2.2 Motivating Example

Advanced Persistent Threats (APTs) are sophisticated, long-running attack campaigns in which adversaries establish a foothold in a target network, move laterally to access sensitive resources, and maintain persistence while evading detection. Unlike opportunistic attacks, APTs unfold over extended periods and involve multiple coordinated stages. Figure 1 depicts a simplified APT scenario captured as a provenance graph, based on the DARPA TC E3-THEIA dataset [2].

Initial Access. The attacker exploits a vulnerability in Firefox, causing it to establish bidirectional communication with a command-and-control (C&C) server at `141.43.176.203`. Through this channel, the attacker delivers and writes the `clean` payload to the victim's disk.

Command & Control. The `clean` payload executes with elevated (root) privileges and contacts a second C&C server at `161.116.88.72`.

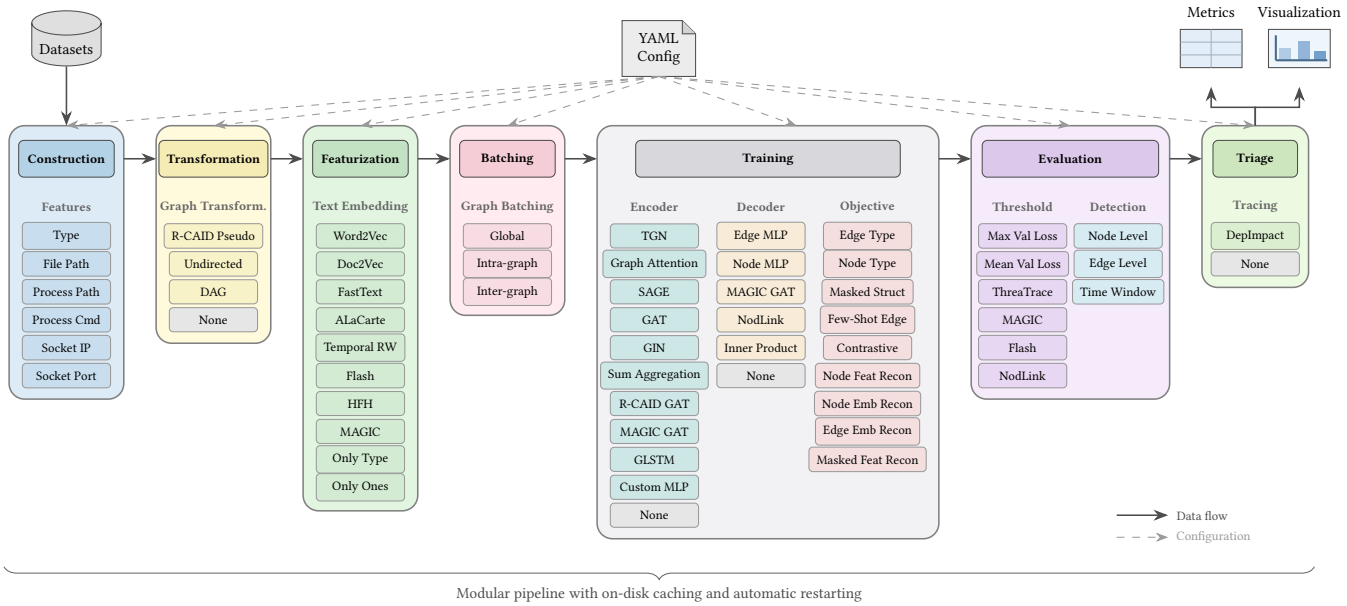


Figure 2: PIDSMaker architecture with seven pipeline stages. Each stage contains configurable components that can be freely combined via YAML configuration for customization and architecture search. Data flows between stages are cached to disk, enabling automatic pipeline restart from any stage.

This server delivers the `profile` payload, which subsequently clones itself to establish persistence on the compromised host.

Lateral Movement. The cloned `profile` process writes two additional payloads: `xdev`, which remains dormant awaiting further instructions, and `mail`, which performs network reconnaissance by probing remote hosts such as `149.52.198.23`. This reconnaissance enables the attacker to identify additional targets for lateral movement within the victim's network.

Provenance graphs are particularly effective for detecting such multi-stage attacks because they preserve *causal relationships* between system entities across time. Unlike traditional log analysis, which examines events in isolation, provenance-based approaches capture the full chain of dependencies from initial exploitation to final reconnaissance. This causal context enables models to capture anomalous behaviors spanning multiple processes, reconstruct complete attack narratives, and distinguish malicious activity from benign operations that may appear similar in isolation.

2.3 Self-Supervised Detection on Provenance Graphs

Modern PIDSs commonly formulate attack detection as an anomaly detection problem using self-supervised learning on provenance graphs. The central idea is to train a model to capture structural regularities of benign system behavior and flag deviations as potentially malicious. Self-supervised learning enables this by training models on unlabeled benign data using auxiliary tasks (e.g., predicting edge types or node attributes), eliminating the need for labeled attacks and framing detection as an anomaly detection problem.

Typical PIDS pipeline. Raw system logs are first parsed into provenance graphs, where nodes and edges represent system entities and interactions. Textual attributes such as file paths, process command lines, and IP addresses are mapped to dense representations using embedding techniques like Word2Vec [43] or FastText [12]. These embeddings are typically concatenated with one-hot encodings of node and edge types to form input features. The resulting graphs are processed by a neural architecture composed of a graph encoder and a task-specific decoder, trained exclusively on benign data. Encoders range from simple linear projections to GNNs that aggregate neighborhood information via message passing. Common architectures include GraphSAGE [20], Graph Attention Networks [56], and Temporal Graph Networks [52], the latter additionally modeling event ordering. Training relies on self-supervised objectives that do not require attack labels. Typical tasks include edge-type prediction, where the model predicts the system call type given endpoint node embeddings, and node-type prediction, which classifies entities as processes, files, or sockets. By minimizing prediction error on benign activity, the model learns representations that encode normal system behavior. At inference time, nodes or edges are assigned anomaly scores based on prediction error, with the assumption that malicious activity deviates from learned benign patterns. These scores are then converted into binary decisions using thresholding strategies, ranging from fixed thresholds [13, 50, 59] to adaptive threshold tuned on validation data [10, 30, 37]. We present the state-of-the-art PIDSs integrated in PIDSMaker in §3.3.

2.4 Practical Challenges

While the self-supervised formulation elegantly removes the need for labeled attacks, it introduces several practical challenges.

Benign Anomalies. Unlike classical anomaly detection, treating attacks as highly anomalous events means that rare but benign

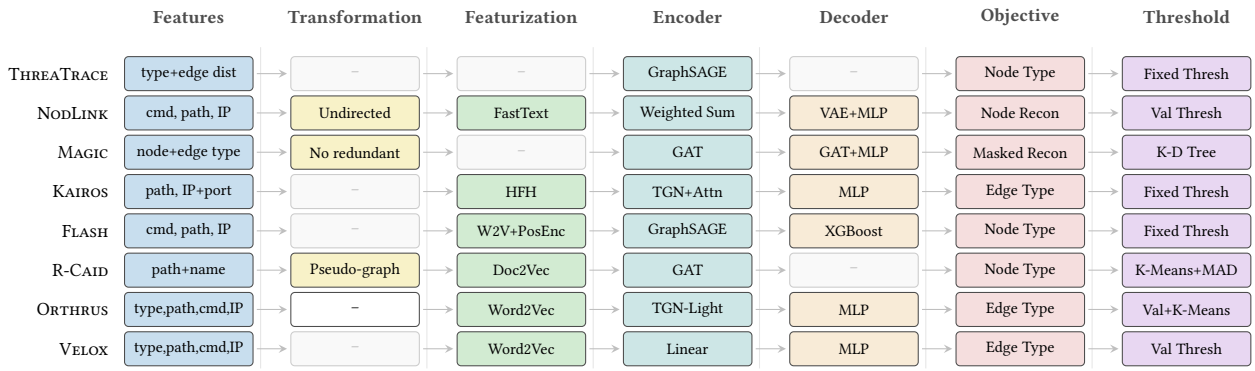


Figure 3: Component modularity in PIDSMaker. Rows represent the eight supported systems; columns represent a subset of components. Each component and associated hyperparameters can be set via YAML configuration.

behaviors may also be flagged as malicious. In the absence of attack supervision, the model cannot distinguish benign deviations, such as a user performing a previously unseen activity, from genuine malicious events, leading to elevated false positive rates compared to standard anomaly detection settings.

Extreme Class Imbalance. The extreme class imbalance in real deployments makes standard evaluation metrics misleading, rendering false positive rates particularly critical. In existing datasets, malicious nodes often account for fewer than one in 10,000, and this ratio is expected to be even lower in real-world environments where attacks are genuinely rare.

Incomplete Ground Truth. Most widely used datasets [2, 3, 55] provide only textual attack descriptions and lack fine-grained labels. Consequently, researchers and security analysts must reconstruct manually labels from ground-truth reports, often leaving many attack-related nodes unlabeled due to missing or ambiguous semantic information. To simplify evaluation, prior work frequently adopts coarse labeling strategies, such as marking entire graphs or subgraphs as malicious rather than individual nodes [30].

Contrast with Related Domains. These challenges distinguish provenance-based intrusion detection from other anomaly detection domains. In financial fraud detection, for instance, transactions follow well-defined schemas with limited variability, and the field benefits from fully-labeled benchmark datasets containing hundreds of thousands of annotated transactions [14, 27]. This enables direct supervised learning, with standard classifiers achieving near-perfect accuracy [45]. In contrast, PIDSs operate over heterogeneous, unbounded graph structures where ground-truth reports identify only a subset of affected nodes, leaving many attack-related behaviors unlabeled and rendering supervised techniques largely inapplicable.

PIDSMaker's Objective. While prior work has surveyed PIDS techniques [9, 28] and highlighted reproducibility issues such as missing artifacts and methodological shortcomings [4, 10], no unified benchmark exists for consistent evaluation before this effort. PIDSMaker fills this gap by providing standardized preprocessing, rigorous evaluation protocols, and a modular environment for experimenting with components from the literature.

3 PIDSMaker

We describe the design and architecture of PIDSMaker (Figure 2) by outlining the core design principles, presenting the modular pipeline, and presenting the supported datasets and systems. The framework is implemented in Python using PyTorch and PyTorch Geometric [1].

3.1 Design Principles

PIDSMaker's architecture follows four core principles to reduce implementation overhead, ensure reproducibility, and enable systematic experimentation.

Modularity. Each pipeline stage is decomposed into independent, interchangeable components with well-defined interfaces. For example, researchers can combine the lightweight encoder from a given system with the decoder of another system without editing any component. As shown in Figure 3, components can be freely mixed across systems, enabling systematic ablation studies and rapid prototyping of new variants.

Configurability. Complete PIDS pipelines are specified declaratively in YAML files rather than code (see §A.1). This ensures all experimental details are explicitly documented, improving reproducibility, and enables researchers to prototype new systems through configuration composition rather than implementation from scratch.

Efficiency. PIDSMaker implements on-disk caching at each pipeline stage to avoid redundant computation. Each stage hashes its configuration and stores outputs in a uniquely named directory. When only downstream parameters change (e.g., tuning learning rate), upstream stages (construction, transformation, featurization) are skipped and cached outputs are reused, significantly reducing experimental turnaround time.

Extensibility. New components integrate by implementing minimal Python interfaces—for instance, adding a text embedding method requires only defining how to encode attributes into vectors. Once registered, components become immediately available to all systems through YAML configuration, allowing PIDSMaker to evolve with emerging techniques and enabling community contributions. New datasets can be similarly added by following the

standardized graph format and naming conventions used for existing DARPA benchmarks.

3.2 Datasets and Ground Truth

PIDSMaker (v2.0.0) comes with preprocessed versions of standard DARPA datasets used in PIDS research: **DARPA TC E3** [2], **DARPA TC E5** [3], **DARPA OpTC** [55]. Starting with v2.1.0, two additional datasets built on top of Carbon Black EDR are also supported: **Atlas v2** [51] and **CARBANAK v2** [38]. Additional datasets will be continuously integrated to the framework.

Node-level Evaluation. Consistent evaluation requires unified ground-truth labels. We adopt the node-level labels previously released in [30], which identify between 41 and 123 individual malicious nodes per dataset through manual analysis of attack descriptions and system logs. Node-level granularity is the natural choice for a unified benchmark for two reasons. First, it reflects operational needs: analysts must identify *which* entities are compromised to initiate incident response. Second, node-level labels subsume coarser ones—predictions can always be aggregated to neighborhood [29, 50, 59] or batch level [13], whereas the reverse is not possible. We acknowledge that systems designed for coarser granularities are disadvantaged under this protocol; we analyze this effect in §4. The labels are publicly available and can serve as a basis for evaluation at alternative granularities.

3.3 Supported Systems

PIDSMaker implements the major PIDSs published in top security venues in recent years, all of which share a similar architecture: ThreaTrace [59], NodLink [37], Magic [29], Kairos [13], Flash [50], R-Caid [19], Orthrus [30], and Velox [10]. As shown in Figure 3, each system is decomposed into interchangeable components that can be freely combined to design new variants. While these systems operate at varying granularities (neighborhood, batch, or node level), PIDSMaker unifies all evaluations at the node level for consistent comparison.

At the neighborhood level, Threatrace employs a GraphSAGE encoder to predict entity types before fixed thresholding, while Magic learns node embeddings via a hybrid loss and detects anomalies by measuring embedding similarity within neighborhoods, and Flash computes node-level anomaly scores from text embeddings using a lightweight encoder-decoder for node type prediction. To capture temporal dynamics, Kairos applies Temporal Graph Networks on dynamic provenance graphs, detecting anomalies at the batch level through edge type prediction. At the node level, NodLink formulates APT detection as a Steiner Tree Problem to improve detection granularity, R-Caid constructs pseudo-graphs connecting nodes to their root causes and identifies anomalies through cluster deviation, and Orthrus combines a lightweight TGN variant with K-Means clustering to reduce false positives. More recently, Velox demonstrates that lightweight detection without GNNs can achieve state-of-the-art results while detecting the majority of attacks.

Reproduction Details. Each system was integrated into PIDSMaker based on its original public source code. For R-Caid, which is not open-source, we re-implemented the approach based on the published paper. A recent reproducibility study [4] found that many PIDS results cannot be reproduced due to incomplete artifacts, and prior work [5, 13, 15, 29, 37, 50, 59, 67] reports inconsistent results for identical systems using different evaluation protocols. Rather than attempting to reproduce individual results under these inconsistent settings, PIDSMaker standardizes the evaluation protocol itself—unifying preprocessing, ground-truth labels, and dataset splits. We acknowledge that systems may achieve different absolute performance than originally published; these discrepancies arise from methodological differences (labeling granularity, preprocessing choices, dataset-specific configurations) rather than implementation errors. Our goal is not to reproduce individual results but to establish a consistent baseline enabling fair comparison going forward. Implementation correctness was verified by confirming that each system approaches its originally reported performance trends when using the labeling strategy from the original work, with author consultation when needed. For fair comparison under the standardized protocol, we recommend using PIDSMaker's built-in hyperparameter tuning to optimize each system for the new setting, as done in our experiments (§4.1).

3.4 Pipeline Stages

Through systematic analysis of the eight systems described in §3.3, we identified seven common stages that collectively constitute PIDSMaker's detection pipeline (Figure 2). While individual systems may skip certain stages or implement them differently, this decomposition captures the end-to-end workflow shared across all studied approaches: from raw provenance logs to final attack detection. Figure 2 summarizes the component implementations currently available in PIDSMaker for each stage, including components reimplemented from public PIDS repositories and additional variants that enable exploration beyond the original systems.

(1) **Construction.** Parse raw provenance to construct a graph and extract attributes (e.g., entity types, file paths, process command lines, and network addresses).
(2) **Transformation.** Apply graph transformations to improve learning, such as converting to undirected graphs, removing redundant edges, converting to directed acyclic graphs (DAGs), or constructing pseudo-graphs that connect nodes to root causes.
(3) **Featurization.** Convert attributes into numerical representations using text embeddings (e.g., Word2Vec [43], Doc2Vec [35], FastText [12]) or domain-specific encodings such as Hierarchical Feature Hashing (HFH) [68].
(4) **Batching.** Partition provenance into temporal subgraphs that fit in memory, typically using fixed-duration windows or a fixed number of events.
(5) **Training.** Train an encoder (typically a GNN) to produce node/edge embeddings and a decoder to map embeddings to predictions, using a self-supervised objective over benign activity.
(6) **Evaluation.** At inference, anomaly scores from prediction errors or reconstruction losses are thresholded (fixed, validation-based, or clustering) to produce detections. Validation-based thresholds usually use the maximum loss on a benign validation set. Metrics can be reported at the node, edge, or graph level, substantially affecting results.
(7) **Triage.** Optionally post-process detections to prioritize alerts or identify attack entry points using backward tracking techniques

System	E3-CADETS								E3-THEIA								E3-CLEARSCOPE								E5-THEIA							
	TP	FP	TN	FN	Prec	Rec	MCC	ADP	TP	FP	TN	FN	Prec	Rec	MCC	ADP	TP	FP	TN	FN	Prec	Rec	MCC	ADP	TP	FP	TN	FN	Prec	Rec	MCC	ADP
Threatrace	26	17k	265k	42	0.00	0.38	0.02	0.01	118	701k	0	0	0.00	1.00	0.00	0.10	41	35k	76k	0	0.00	1.00	0.03	0.03	64	722k	28k	5	0.00	0.93	-0.01	0.05
NodLink	53	58k	223k	15	0.00	0.78	0.02	0.96	94	445k	256k	24	0.00	0.80	-0.00	0.50	40	7k	104k	1	0.01	0.98	0.07	0.03	57	273k	477k	12	0.00	0.83	0.01	0.00
Magic	68	144k	138k	0	0.00	1.00	0.02	0.06	74	196k	505k	44	0.00	0.63	-0.00	0.00	0	0	111k	41	0.00	0.00	0.00	0.03	68	445k	305k	1	0.00	0.99	0.00	1.00
Kairos	0	2	282k	68	0.00	0.00	-0.00	0.01	2	337	700k	116	0.01	0.02	0.00	0.50	1	18k	94k	40	0.00	0.02	-0.03	1.00	2	15	750k	67	0.12	0.03	0.05	0.50
Flash	1	3	282k	67	0.25	0.01	0.11	0.34	3	8k	693k	115	0.00	0.03	0.00	0.05	35	23k	88k	6	0.00	0.85	0.03	0.07	31	311k	439k	38	0.00	0.45	-0.00	0.02
R-Caid	2	0	282k	66	1.00	0.03	0.17	0.44	–	–	–	–	–	–	–	–	15	169	111k	26	0.08	0.37	0.46	0.50	–	–	–	–	–	–	–	–
Orthrus	10	7	282k	58	0.59	0.15	0.36	1.00	4	27	701k	114	0.13	0.03	0.07	1.00	1	2k	110k	40	0.00	0.02	-0.01	1.00	2	3	750k	67	0.40	0.03	0.13	1.00
Velox	13	3	282k	55	0.81	0.19	0.43	1.00	10	1	701k	108	0.91	0.08	0.28	0.97	1	531	111k	40	0.00	0.02	0.01	1.00	2	0	750k	67	1.00	0.03	0.17	1.00

System	E5-CLEARSCOPE								OpTC-H201								OpTC-H501								OpTC-H051							
	TP	FP	TN	FN	Prec	Rec	MCC	ADP	TP	FP	TN	FN	Prec	Rec	MCC	ADP	TP	FP	TN	FN	Prec	Rec	MCC	ADP	TP	FP	TN	FN	Prec	Rec	MCC	ADP
Threatrace	26	64k	87k	8	0.00	0.76	0.01	0.01	3k	1.4M	35k	5	0.00	1.00	0.02	0.17	2	42	1.5M	747	0.05	0.00	0.01	1.00	114	1.5M	0	0	0.00	1.00	0.00	1.00
NodLink	0	24k	127k	34	0.00	0.00	-0.01	0.17	1k	82k	1.4M	2k	0.02	0.44	0.09	1.00	376	106k	1.4M	373	0.00	0.50	0.05	0.08	30	100k	1.4M	84	0.00	0.26	0.00	1.00
Magic	34	105k	46k	0	0.00	1.00	0.01	0.01	1k	1.2M	235k	2k	0.00	0.38	-0.03	1.00	185	260k	1.2M	564	0.00	0.25	0.00	1.00	102	259k	1.2M	12	0.00	0.89	0.02	0.50
Kairos	0	6	151k	34	0.00	0.00	-0.00	0.33	2	11	1.4M	3k	0.15	0.00	0.01	1.00	1	42	1.5M	748	0.02	0.00	0.00	1.00	1	2	1.5M	113	0.33	0.01	0.06	1.00
Flash	8	24k	127k	26	0.00	0.24	0.01	0.04	2k	239k	1.2M	1k	0.01	0.52	0.04	1.00	222	215k	1.3M	527	0.00	0.30	0.01	1.00	2	9k	1.5M	112	0.00	0.02	0.01	0.50
R-Caid	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
Orthrus	8	13	151k	26	0.38	0.24	0.33	0.12	2	22	1.4M	3k	0.08	0.00	0.01	1.00	0	0	1.5M	749	0.00	0.00	0.00	0.33	0	0	1.5M	114	0.00	0.00	0.00	1.00
Velox	8	8	151k	26	0.50	0.24	0.38	0.61	2	7	1.4M	3k	0.22	0.00	0.01	1.00	1	7	1.5M	748	0.13	0.00	0.01	0.50	2	11	1.5M	112	0.15	0.02	0.07	1.00

Table 1: Node-level detection results under consistent evaluation setting (best run among five iterations). Prec=Precision, Rec=Recall, MCC=Matthews Correlation Coefficient, ADP=Attack Detection Precision (best of 5 runs). For R-Caid, "–" denotes runs aborted after exceeding 24 hours; as the system is not open-source, our re-implementation may not scale optimally.

such as DepImpact [17]. The potential of this stage to improve actionability is discussed in §5.

4 Benchmark Results

In this section, we benchmark all eight systems on all datasets using PIDSMaker 's unified node-level protocol, evaluating detection performance followed by computational scalability.

4.1 Overall Detection Performance

Table 1 presents node-level detection results across all datasets using consistent ground-truth labels. We tuned all systems on each dataset (§A.4) using PIDSMaker's built-in hyperparameter tuning. Each system was evaluated over five runs; the best-seed results appear in Table 1. In intrusion detection's highly imbalanced setting, where low false positives matter more than many true positives [16], we report TP, FP, TN, FN, and MCC [41], a robust metric that is high only when all confusion matrix entries are predicted well [49]. We also measure attack detection precision (ADP) [10], a standard metric in PIDS that measures AUPRC where precision is at the node level and recall is at the attack level.

Interpreting Performance Under Unified Evaluation. The near-zero MCC observed for most systems is not an artifact of the benchmark — it is the benchmark's central finding. Node-level evaluation reflects operational reality: incident responders must identify which entities are compromised, not merely that an attack occurred somewhere in a neighborhood or time window. The performance gap relative to originally published results quantifies, for the first time, how much coarse labeling strategies inflate reported metrics. Systems designed for neighborhood- or batch-level evaluation [13, 29, 50, 59] naturally struggle at finer granularity—this does not invalidate them, but reveals that their reported results reflect a less demanding task. PIDSMaker's node-level protocol establishes the harder but operationally meaningful baseline the field currently lacks.

Three Detection Regimes. The results reveal three distinct regimes traceable to architectural choices (Figure 4). Threatrace, NodLink, and Magic exhibit *indiscriminate scoring*: their message-passing mechanisms propagate anomaly signals across neighborhoods, producing overlapping benign/attack distributions with high recall but near-zero precision at the node level. Kairos and Flash fall into an *over-conservative* regime, with scores near zero for both classes, suppressing most detections. Velox, Orthrus, and R-Caid achieve *sparse but precise* detection, with well-separated score distributions, concentrating high anomaly scores on individual nodes rather than spreading them across neighborhoods.

Practical Significance. As confirmed in prior work [10, 16], a low recall does not imply operational uselessness. In practice, detecting at least one node per attack campaign with manageable false positives suffices to reconstruct attack paths [16, 17]. From this perspective, Velox detects at least one attack node across all eight datasets with single-digit false positives on most E3 benchmarks, while Orthrus achieves the same on six datasets.

Implications for the Field. These results suggest that strong performance reported in prior PIDS literature may partly reflect evaluation choices—particularly coarse labeling—rather than fundamental detection capability, underscoring the value of standardized evaluation. The consistent superiority of node-level systems (Velox, Orthrus) further suggests that detection granularity is a first-order architectural decision for future PIDSs.

4.2 Impact of Components on Detection

The unified architecture of PIDSMaker makes it possible to estimate which component contributes most to detection. We focus on three components shared across all PIDS architectures: the text encoder (featurization), the graph encoder, and the training objective. We ran 108 ablations spanning four featurizations, three encoders, and three training objectives on three E3 datasets, reporting ADP averaged across datasets (Table 2).

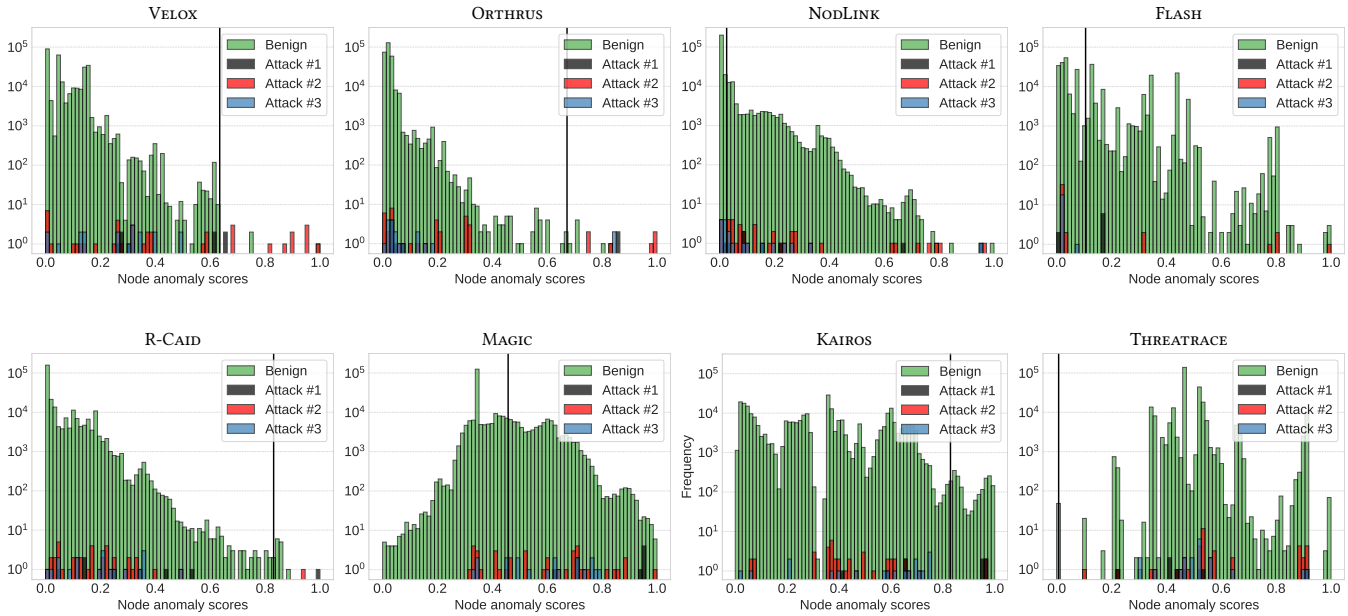


Figure 4: Predicted node anomaly score distributions of systems on the `E3-CADETS` dataset, containing three APT attacks. The vertical line represents the threshold of each system.

Objective	Featurization				Encoder		
	FastText	HFH	W2V	None	Linear	NodLink	Orthrus
Edge Type Pred.	0.29	0.30	**0.42**	0.22	0.30	0.27	**0.35**
Node Type Pred.	0.06	0.05	0.08	0.08	0.07	**0.13**	0.00
Node Feat. Recon.	0.07	0.03	0.14	0.08	0.05	**0.15**	0.04

Table 2: Component ablation: ADP averaged across three E3 datasets. Left block varies featurization (averaging across encoders); right block varies encoder (averaging across featurizations). None uses one-hot node types only (no text encoding).

Training objective dominates. Edge type prediction outperforms alternatives across every featurization and encoder. The gap between objectives exceeds any encoder- or featurization-driven variation in our sweep, identifying objective design as the dominant performance lever.

Encoder choice interacts with the objective. A linear encoder paired with edge prediction is competitive with Orthrus's encoder, while NodLink's lags both. Under suboptimal objectives, encoder complexity actively hurts: Orthrus collapses under node type prediction whereas NodLink retains some signal.

Text attributes bring value. Removing text encoding noticeably degrades performance in the best-performing setting. Word2vec leads under edge prediction, confirming that text embeddings carry meaningful semantic signal, though the gap between objectives is substantially larger than that between featurization methods.

Design implications. Three lessons emerge for future PIDS design: (1) prioritize objective design before scaling encoder capacity; (2) avoid neighborhood aggregation when fine-grained precision matters, as it diffuses anomaly signal to benign neighbors (Figure 4); (3) consider simpler architectures before adding structural complexity. Velox's strong performance is consistent with these lessons: it pairs edge prediction with a linear encoder and competitive featurization.

4.3 False Positive Analysis

We further analyze where false positives are located in the graph relative to true attack nodes. Figure 5 reports the distribution of each system's false positives by hop distance to the nearest attack node. For most systems, the majority of false positives sit within one or two hops of an attack node, indicating that misclassifications cluster tightly around genuine malicious activity rather than scattering across the graph. This locality has direct implications for the coarser-grained evaluation protocols used in prior work [29, 50, 59], where every node within a 2-hop neighborhood of an attack is treated as malicious: under such protocols, the blue portions of Figure 5 would collapse into true positives, mechanically inflating precision without any change in detection capability. From the analysts' perspective, these nearby nodes are often shared libraries and common entities with limited operational value [16]. We adopt node-level as the default because it reflects the operationally meaningful task, but PIDSMaker computes hop-distance breakdowns from the same ground truth, so both perspectives can be reported without relabeling. Note that these measurements were obtained from an independent run; absolute counts may differ slightly from Table 1 due to seed-induced variability, though the relative trends are preserved.

[1]Some datasets contain disjoint graphs; whether they should is beyond the scope of this paper.

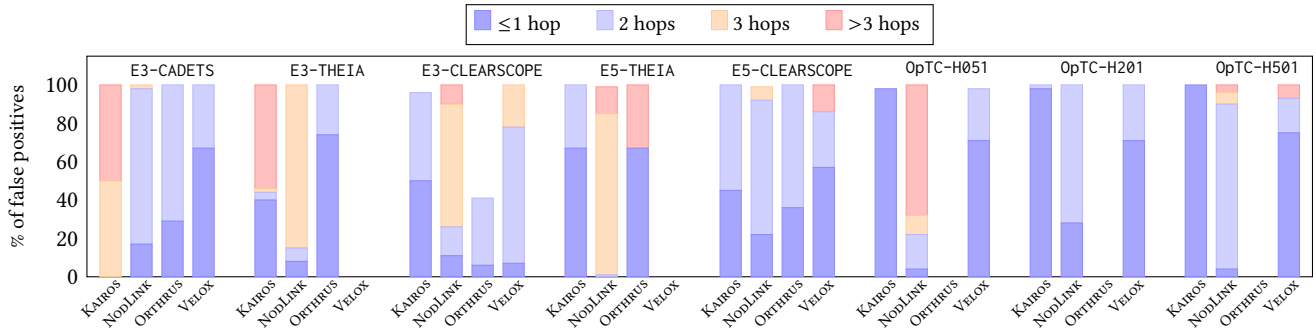


Figure 5: Hop-distance distribution of false positives. Empty bars denote zero false positives. On `E3-CLEARSCOPE`, some bars do not reach 100% because false positives occur in disjoint graphs unrelated to attacks.[1]

4.4 Computational Cost

Figure 6 compares the memory footprint, execution time, and detection accuracy (MCC) of PIDS systems on E3-CADETS, a small-scale dataset, and E5-THEIA, a larger and more representative benchmark. For each dataset, we present three complementary views: MCC vs. memory (Figure 6a, Figure 6d), MCC vs. time (Figure 6b, Figure 6e), and throughput vs. memory (Figure 6c, Figure 6f). These visualizations reveal critical trade-offs between detection accuracy, resource consumption, and processing speed.

Memory Usage. Memory consumption spans three orders of magnitude, from 27MB (Velox on E3-CADETS) to 29GB (Kairos on E5-THEIA). Kairos and R-Caid incur the largest memory footprints (20–30GB) due to temporal node memories and large pseudo-graph constructions, respectively. In contrast, Velox, Flash, and Threatrace remain below 1GB, making them suitable for resource-constrained deployments; Velox achieves the lowest usage by avoiding GNNs and learning pairwise interactions over fixed-size edge batches. Figure 6a and Figure 6d further show that higher memory usage does not translate to better detection accuracy: despite their large footprints, Kairos and R-Caid attain near-zero MCC, while lightweight systems such as Velox and Orthrus achieve the best performance.

Execution Time and Throughput. Execution time scales with dataset size: across all detection systems, the larger E5-THEIA requires 5–10× longer processing than E3-CADETS. Kairos has the longest runtimes (52 min on E3-CADETS, 422 min on E5-THEIA) due to the overhead of temporal modeling, resulting in the lowest throughput (Figure 6c, Figure 6f). Node-level systems such as Flash, NodLink, and Threatrace are among the fastest, as provenance graphs contain far fewer nodes than edges. However, Figure 6b and Figure 6e show that speed does not imply better detection: Velox attains the highest MCC on both datasets despite moderate runtimes, while faster systems like Flash and NodLink perform poorly. We note that the framework contributes 2.0%–2.8% of the total execution time.

The Accuracy-Efficiency Trade-off. The throughput vs. memory plots (Figure 6c, Figure 6f) reveal that Velox and Orthrus occupy the Pareto frontier: they combine low memory footprint with reasonable throughput while delivering the best detection accuracy (MCC > 0.13 on E5-THEIA, MCC > 0.36 on E3-CADETS). In contrast, systems like Kairos and R-Caid consume massive resources but fail to detect threats effectively (MCC ≈ 0), while Flash and Threatrace, though fast and lightweight, also struggle with accuracy. This shows that detection performance and computational efficiency are orthogonal, requiring careful architectural choices rather than increased model complexity.

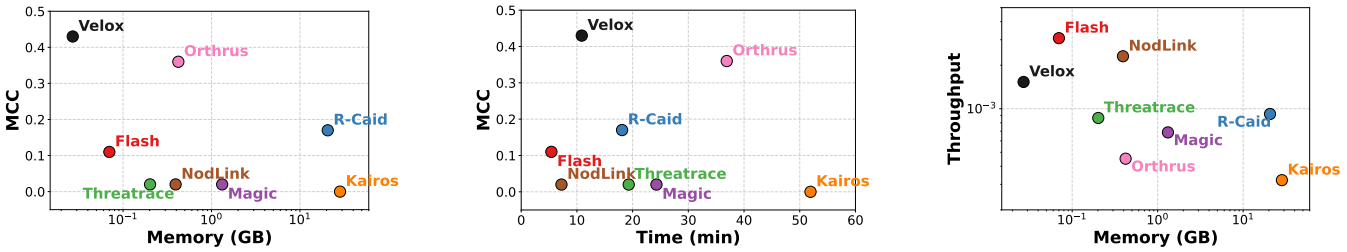
Deployment Implications. These results highlight a fundamental trade-off between model complexity and deployability. Systems with high memory requirements (Kairos, R-Caid) are impractical for deployment on endpoints or in environments with limited resources, restricting their use to centralized analysis servers—yet even in such settings, their poor detection accuracy makes them unsuitable. Lightweight alternatives such as Velox and Orthrus offer a more practical path toward real-time, distributed deployment—an important consideration for enterprise-scale monitoring where provenance data is generated continuously across thousands of hosts. The superior accuracy of these efficient systems further strengthens their case for practical deployment.

5 Discussion

Evaluating multiple PIDSs within a unified framework exposes several open challenges.

Lack of benign anomaly benchmarks. Existing benchmark datasets typically treat the world as binary (benign vs. attack) and rarely label *benign anomalies* (legitimate but uncommon behavior) as a separate class. As a result, self-supervised PIDSs trained on "normal" traces often assign high anomaly scores to rare yet acceptable events (e.g., software updates, administrative maintenance, or bursty but authorized activity), which can inflate false positives. Further, real deployments face shifting definitions of normal across hosts, workloads, and time, and some datasets already contain unlabeled benign irregularities that confound evaluation. Benchmarks that explicitly annotate benign anomalies (or provide workload/-maintenance labels) would enable more realistic measurement of detection specificity and help disentangle "rare but benign" from truly malicious behavior.

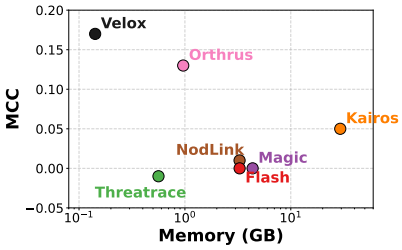
Future extensions of PIDSMaker. PIDSMaker currently targets self-supervised PIDSs. Extending the framework to supervised and



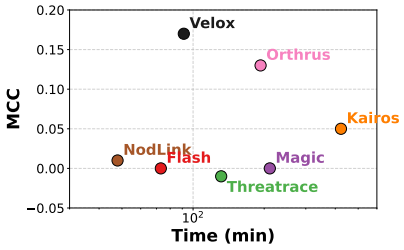
(a) MCC vs. memory (E3-CADETS).

(b) MCC vs. time (E3-CADETS).

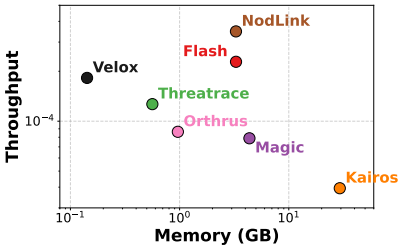
(c) Time vs. memory (E3-CADETS).



(d) MCC vs. memory (E5-THEIA).



(e) MCC vs. time (E5-THEIA).



(f) Time vs. memory (E5-THEIA).

Figure 6: Efficiency–accuracy trade-offs on E3-CADETS (top row) and E5-THEIA (bottom row). Results at the top-left corner are best while bottom-right are worst.

rule-based approaches [44, 57] would broaden coverage and enable more comprehensive comparisons.

Although we do not evaluate it in this paper, PIDSMaker also includes an optional tracing stage that reconstructs attack paths from anomalous nodes and edges. This component represents a promising research direction that has been explored in prior work [17, 24, 25, 39, 65]. Integrating such techniques with existing systems could enforce connectivity constraints and incorporate domain knowledge about attack behavior, translating anomaly scores into coherent incident narratives, surfacing additional compromised entities along dependency paths, and suppressing false positives that lack plausible causal context.

Beyond detection performance, deployment raises challenges such as concept drift and adversarial manipulation. As benign behavior shifts over time, models trained on historical traces can degrade and require continual adaptation. Attackers may also shape provenance patterns to evade detectors. We plan to extend PIDSMaker with drift detection and adaptation mechanisms [31, 64] and with adversarial robustness evaluation [18, 46]. Supporting end-to-end benchmarks of attacks and defenses from the literature would make it easier to measure resilience systematically.

6 Conclusion

We presented PIDSMaker, a unified framework for developing and evaluating provenance-based intrusion detection systems (PIDSs). PIDS research faces significant evaluation challenges: prior work uses inconsistent preprocessing pipelines, non-standard dataset splits, and incompatible ground-truth labeling strategies, undermining reproducibility and impeding fair comparison. PIDSMaker addresses these by consolidating the main recent systems into a modular architecture with standardized preprocessing, consistent ground-truth labels, and integrated experimental utilities. Researchers can rapidly prototype via component reuse, run systematic ablations and hyperparameter tuning, and perform fair comparisons under consistent protocols. We view common evaluation standards as essential for sustained progress and welcome adoption and contributions that extend the framework.

Acknowledgments

We acknowledge the support of the Natural Sciences and Engineering Research Council of Canada (NSERC). Nous remercions le Conseil de recherches en sciences naturelles et en génie du Canada (CRSNG) de son soutien. We acknowledge the support of the Canadian Foundation for Innovation (CFI). Nous remercions la Fondation canadienne pour l'innovation (FCI) de son soutien. We acknowledge the support of the British Columbia Knowledge Development Fund (BCKDF), UBC Advanced Research Computing (ARC), and Dell Computer. Research reported in this publication was supported by an Amazon Research Award. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not reflect those of the sponsors.

References

[1] 2025. PyTorch Geometric. https://pytorch-geometric.readthedocs.io/en/latest/.
[2] Accessed May 26, 2026. Transparent Computing Engagement 3 Data Release. https://github.com/darpa-i2o/Transparent-Computing/blob/master/README-E3.md.
[3] Accessed May 26, 2026. Transparent Computing Engagement 5 Data Release. https://github.com/darpa-i2o/Transparent-Computing.

[4] Talha Abrar, Ahmad Shamail, Mohammad Jaffer Iqbal, Amaan Ahmed, Muhammad Abdullah, Muhammad Shayan, Fareed Zaffar, Thomas Pasquier, David Eyers, and Ashish Gehani. 2025. On the Reproducibility of Provenance-based Intrusion Detection that uses Deep Learning. In *Conference on Reproducibility and Replicability (REP'25)*. ACM.

[5] Abdulellah Alsaheel, Yuhong Nan, Shiqing Ma, Le Yu, Gregory Walkup, Z. Berkay Celik, X. Zhang, and Dongyan Xu. 2021. ATLAS: A Sequence-based Learning Approach for Attack Investigation. In *Security Symposium (USENIX Sec'21)*. USENIX.

[6] Daniel Arp, Erwin Quiring, Feargus Pendlebury, Alexander Warnecke, Fabio Pierazzi, Christian Wressnegger, Lorenzo Cavallaro, and Konrad Rieck. 2022. Dos and Don'ts of Machine Learning in Computer Security. In *Security Symposium (USENIX Sec'22)*. USENIX.

[7] Gbadebo Ayoade, Khandakar Ashrafi Akbar, Pracheta Sahoo, Y. Gao, Anmol Agarwal, Kangkook Jee, L. Khan, and Anoop Singhal. 2020. Evolving Advanced Persistent Threat Detection using Provenance Graph and Metric Learning. In *Conference on Communications and Network Security (CNS'20)*. IEEE.

[8] Adam Bates, Dave Jing Tian, Kevin R. B. Butler, and Thomas Moyer. 2015. Trustworthy Whole-System Provenance for the Linux Kernel. In *Security Symposium (USENIX Sec'15)*. USENIX.

[9] Tristan Bilot, Nour El Madhoun, Khaldoun Al Agha, and Anis Zouaoui. 2023. Graph neural networks for intrusion detection: A survey. *IEEE Access* 11 (2023), 49114–49139.

[10] Tristan Bilot, Baoxiang Jiang, Zefeng Li, Nour El Madhoun, Khaldoun Al Agha, Anis Zouaoui, and Thomas Pasquier. 2025. Sometimes Simpler is Better: A Comprehensive Analysis of State-of-the-Art Provenance-Based Intrusion Detection Systems. In *Security Symposium (USENIX Sec'25)*. USENIX.

[11] Tristan Bilot, Baoxiang Jiang, and Thomas Pasquier. 2026. PIDSMaker: Building and Evaluating Provenance-based Intrusion Detection Systems. In *Conference on Knowledge Discovery and Data Mining (KDD'26)*. ACM.

[12] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics* 5 (2017), 135–146.

[13] Zijun Cheng, Qiujian Lv, Jinyuan Liang, Yan Wang, Degang Sun, Thomas Pasquier, and Xueyuan Han. 2023. Kairos: Practical Intrusion Detection and Investigation using Whole-system Provenance. In *Symposium on Security and Privacy (S&P'24)*. IEEE.

[14] Andrea Dal Pozzolo, Olivier Caelen, Reid A. Johnson, and Gianluca Bontempi. 2015. Calibrating Probability with Undersampling for Unbalanced Classification. In *IEEE Symposium on Computational Intelligence and Data Mining (CIDM)*. 159–166.

[15] Hailun Ding, Juan Zhai, Yuhong Nan, and Shiqing Ma. 2023. AIRTAG: Towards Automated Attack Investigation by Unsupervised Learning with Log Texts. In *Security Symposium (USENIX Sec'23)*. USENIX.

[16] Feng Dong, Shaofei Li, Peng Jiang, Ding Li, Haoyu Wang, Liangyi Huang, Xusheng Xiao, Jiedong Chen, Xiapu Luo, Yao Guo, et al. 2023. Are we there yet? An Industrial Viewpoint on Provenance-based Endpoint Detection and Response Tools. In *Conference on Computer and Communications Security (CCS'23)*. ACM.

[17] Pengcheng Fang, Peng Gao, Changlin Liu, Erman Ayday, Kangkook Jee, Ting Wang, Yanfang (Fanny) Ye, Zhuotao Liu, and Xusheng Xiao. 2022. Back-Propagating System Dependency Impact for Attack Investigation. In *Security Symposium (USENIX Sec'22)*. USENIX.

[18] Akul Goyal, Xueyuan Han, G. Wang, and Adam Bates. 2023. Sometimes, You Aren't What You Do: Mimicry Attacks against Provenance Graph Host Intrusion Detection Systems. In *Network and Distributed System Security Symposium (NDSS'23)*. The Internet Society.

[19] Akul Goyal, Gang Wang, and Adam Bates. 2024. R-CAID: Embedding Root Cause Analysis within Provenance-based Intrusion Detection. In *Symposium on Security and Privacy (S&P'24)*. IEEE.

[20] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. *Advances in neural information processing systems* 30 (2017).

[21] Xueyuan Han, Thomas Pasquier, Adam Bates, James Mickens, and Margo I. Seltzer. 2020. Unicorn: Runtime Provenance-Based Detector for Advanced Persistent Threats. In *Network and Distributed System Security Symposium (NDSS'20)*. The Internet Society.

[22] Xueyuan Han, Thomas Pasquier, Tanvi Ranjan, Mark Goldstein, and Margo I. Seltzer. 2017. FRAPpuccino: Fault-detection through Runtime Analysis of Provenance. In *Workshop on Hot Topics in Cloud Computing (HotCloud'17)*. USENIX.

[23] Xueyuan Han, Xiao Yu, Thomas Pasquier, Ding Li, Junghwan John Rhee, James W. Mickens, Margo I. Seltzer, and Haifeng Chen. 2021. SIGL: Securing Software Installations Through Deep Graph Learning. In *Security Symposium (USENIX Sec'21)*. USENIX.

[24] Wajih Ul Hassan, Shengjian Guo, Ding Li, Zhengzhang Chen, Kangkook Jee, Zhichun Li, and Adam Bates. 2019. NoDoze: Combatting Threat Alert Fatigue with Automated Provenance Triage. In *Network and Distributed System Security Symposium (NDSS'19)*. The Internet Society.

[25] Wajih Ul Hassan, Ding Li, Kangkook Jee, Xiao Yu, Kexuan Zou, Dawei Wang, Zhengzhang Chen, Zhichun Li, Junghwan John Rhee, Jiaping Gui, and Adam Bates. 2020. This is Why We Can't Cache Nice Things: Lightning-Fast Threat Hunting using Suspicion-Based Hierarchical Storage. In *Annual Computer Security Applications Conference (ACSAC'20)*. IEEE.

[26] Md Nahid Hossain, Sadegh M. Milajerdi, Junao Wang, Birhanu Eshete, Rigel Gjomemo, R. C. Sekar, Scott D. Stoller, and Venkat Venkatakrishnan. 2017. SLEUTH: Real-time Attack Scenario Reconstruction from COTS Audit Data. In *Security Symposium (USENIX Sec'17)*. USENIX.

[27] IEEE Computational Intelligence Society and Vesta Corporation. 2019. IEEE-CIS Fraud Detection. Kaggle Competition. https://www.kaggle.com/c/ieee-fraud-detection

[28] Muhammad Adil Inam, Yinfang Chen, Akul Goyal, Jason Liu, Jaron Mink, Noor Michael, Sneha Gaur, Adam Bates, and Wajih Ul Hassan. 2023. SoK: History is a Vast Early Warning System: Auditing the Provenance of System Intrusions. In *Symposium on Security and Privacy (S&P'23)*. IEEE.

[29] Zian Jia, Yun Xiong, Yuhong Nan, Yao Zhang, Jinjing Zhao, and Mi Wen. 2024. MAGIC: Detecting Advanced Persistent Threats via Masked Graph Representation Learning. In *Security Symposium (USENIX Sec'24)*. USENIX.

[30] Baoxiang Jiang, Tristan Bilot, Nour El Madhoun, Khaldoun Al Agha, Anis Zouaoui, Shahrear Iqbal, Xueyuan Han, and Thomas Pasquier. 2025. ORTHRUS: Achieving High Quality of Attribution in Provenance-based Intrusion Detection Systems. In *Security Symposium (USENIX Sec'25)*. USENIX.

[31] Roberto Jordaney, Kumar Sharad, Santanu K Dash, Zhi Wang, Davide Papini, Ilia Nouretdinov, and Lorenzo Cavallaro. 2017. Transcend: Detecting Concept Drift in Malware Classification Models. In *Security Symposium (USENIX Sec'17)*. USENIX.

[32] Maya Kapoor, Joshua Melton, Michael Ridenhour, Siddhartha Krishnan, and Thomas Moyer. 2021. PROV-GEM: Automated Provenance Analysis Framework using Graph Embeddings. In *International Conference on Machine Learning and Applications (ICMLA'21)*. IEEE.

[33] Isaiah J. King and Huimin Huang. 2022. Euler: Detecting Network Lateral Movement via Scalable Temporal Graph Link Prediction. In *Network and Distributed System Security Symposium (NDSS'22)*. The Internet Society.

[34] Isaiah J. King, Xiaokui Shu, Jiyong Jang, Kevin Eykholt, Taesung Lee, and H. Howie Huang. 2023. EdgeTorrent: Real-time Temporal Graph Representations for Intrusion Detection. In *International Symposium on Research in Attacks, Intrusions and Defenses (RAID'23)*.

[35] Quoc Le and Tomas Mikolov. 2014. Distributed Representations of Sentences and Documents. In *International Conference on Machine Learning (IMCL'14)*.

[36] Mark Lemay, Wajih Ul Hassan, Thomas Moyer, Nabil Schear, and Warren Smith. 2017. Automated Provenance Analytics: A Regular Grammar Based Approach with Applications in Security. In *Workshop on the Theory and Practice of Provenance (TaPP'17)*. USENIX.

[37] Shaofei Li, Feng Dong, Xusheng Xiao, Haoyu Wang, Fei Shao, Jiedong Chen, Yao Guo, Xiangqun Chen, and Ding Li. 2024. NODLINK: An Online System for Fine-Grained APT Attack Detection and Investigation. In *Network and Distributed System Security Symposium (NDSS'24)*. The Internet Society.

[38] Jason Liu, Muhammad Adil Inam, Akul Goyal, Dylen Greenenwald, Adam Bates, and Saurav Chittal. 2026. How to Effectively Trace Provenance on Windows Endpoint Detection & Response Telemetry. In *Workshop on Attack Provenance, Reasoning, and Investigation for Security in the Monitored Environment (PRISM'26)*. Internet Society.

[39] Yushan Liu, Mu Zhang, Ding Li, Kangkook Jee, Zhichun Li, Zhenyu Wu, Junghwan John Rhee, and Prateek Mittal. 2018. Towards a Timely Causality Analysis for Enterprise Security. In *Network and Distributed System Security Symposium (NDSS'18)*. The Internet Society.

[40] Emaad Manzoor, Sadegh Momeni, Venkat Venkatakrishnan, and Leman Akoglu. 2016. Fast Memory-efficient Anomaly Detection in Streaming Heterogeneous Graphs. In *International Conference on Knowledge Discovery and Data Mining (KDD'16)*. ACM.

[41] Brian W Matthews. 1975. Comparison of the predicted and observed secondary structure of T4 phage lysozyme. *Biochimica et Biophysica Acta (BBA)-Protein Structure* (1975).

[42] Lingxiang Meng, Rongrong Xi, Ziang Li, and Hongsong Zhu. 2024. PG-AID: An Anomaly-based Intrusion Detection Method Using Provenance Graph. In *International Conference on Computer Supported Cooperative Work in Design (CSCWD'24)*. IEEE.

[43] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient Estimation of Word Representations in Vector Space. *arXiv preprint arXiv:1301.3781* (2013).

[44] Sadegh M Milajerdi, Rigel Gjomemo, Birhanu Eshete, Ramachandran Sekar, and VN Venkatakrishnan. 2019. HOLMES: Real-time APT Detection through Correlation of Suspicious Information Flows. In *Symposium on Security and Privacy (S&P'19)*. IEEE.

[45] Soroor Motie and Bijan Raahemi. 2024. Financial Fraud Detection Using Graph Neural Networks: A Systematic Review. *Expert Systems with Applications* (2024).

[46] Kunal Mukherjee, Joshua Wiedemeier, Tianhao Wang, James Wei, Feng Chen, Muhyun Kim, Murat Kantarcioglu, and Kangkook Jee. 2023. Evading Provenance-Based ML Detectors with Adversarial System Actions. In *Security Symposium*

(USENIX Sec'23). USENIX.
[47] Thomas Pasquier, Xueyuan Han, Mark Goldstein, Thomas Moyer, D. Eyers, Margo I. Seltzer, and Jean Bacon. 2017. Practical whole-system provenance capture. In *Symposium on Cloud Computing (SoCC'17)*. ACM.
[48] Devin J Pohly, Stephen McLaughlin, Patrick McDaniel, and Kevin Butler. 2012. Hi-Fi: Collecting High-Fidelity Whole-System Provenance. In *Annual Computer Security Applications Conference (ACSAC'12)*. ACM.
[49] DM Powers. 2020. *Evaluation: from Precision, Recall and F-Measure to ROC, Informedness, Markedness and Correlation*. Technical Report. Flinders University.
[50] Mati Ur Rehman, Hadi Ahmadi, and Wajih Ul Hassan. 2024. Flash: A Comprehensive Approach to Intrusion Detection via Provenance Graph Representation Learning. In *Symposium on Security and Privacy (S&P'24)*. IEEE.
[51] Andy Riddle, Kim Westfall, and Adam Bates. 2023. ATLASv2: ATLAS Attack Engagements, Version 2. *arXiv preprint arXiv:2401.01341* (2023).
[52] Emanuele Rossi, Ben Chamberlain, Fabrizio Frasca, Davide Eynard, Federico Monti, and Michael Bronstein. 2020. Temporal Graph Networks for Deep Learning on Dynamic Graphs. In *International Conference on Machine Learning (ICML'20)*.
[53] Manolis Stamatogiannakis, Paul T. Groth, and Herbert Bos. 2014. Looking Inside the Black-Box: Capturing Data Provenance Using Dynamic Instrumentation. In *International Provenance and Annotation Workshop (IPAW'14)*. Springer.
[54] Jörg Thalheim, Pramod Bhatotia, and Christof Fetzer. 2016. INSPECTOR: Data Provenance Using Intel Processor Trace (PT). In *International Conference on Distributed Computing Systems (ICDCS'16)*. IEEE.
[55] Mike van Opstal and William Arbaugh. 2019. Operationally Transparent Cyber (OpTC) Data Release. https://github.com/FiveDirections/OpTC-data.
[56] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2017. Graph Attention Networks. In *International Conference on Learning Representations (ICLR'17)*.
[57] Lingzhi Wang, Xiangmin Shen, Weijian Li, Zhenyuan Li, R Sekar, Han Liu, and Yan Chen. 2025. Incorporating Gradients to Rules: Towards Lightweight, Adaptive Provenance-based Intrusion Detection. In *Network and Distributed System Security Symposium (NDSS'25)*. Internet Society.
[58] Qi Wang, Wajih Ul Hassan, Ding Li, Kangkook Jee, Xiao Yu, Kexuan Zou, Junghwan John Rhee, Zhengzhang Chen, Wei Cheng, Carl A. Gunter, and Haifeng Chen. 2020. You Are What You Do: Hunting Stealthy Malware via Data Provenance Analysis. In *Network and Distributed System Security Symposium (NDSS'20)*. The Internet Society.
[59] Su Wang, Zhiliang Wang, Tao Zhou, Hongbin Sun, Xia Yin, Dongqi Han, Han Zhang, Xingang Shi, and Jiahai Yang. 2021. THREATRACE: Detecting and Tracing Host-Based Threats in Node Level Through Provenance Graph Learning. *IEEE Transactions on Information Forensics and Security* (2021).
[60] Yulai Xie, Dan Feng, Yuchong Hu, Yan Li, Staunton Sample, and Darrell D. E. Long. 2020. Pagoda: A Hybrid Approach to Enable Efficient Real-Time Provenance Based Intrusion Detection in Big Data Environments. *IEEE Transactions on Dependable and Secure Computing* (2020).
[61] Yulai Xie, Dan Feng, Zhipeng Tan, and Junzhe Zhou. 2016. Unifying intrusion detection and forensic analysis via provenance awareness. *Future Generation of Computer Systems* (2016).
[62] Boyuan Xu, Yiru Gong, Xiaoyu Geng, Yun Li, Cong Dong, Song Liu, Yuling Liu, Bo-Sian Jiang, and Zhigang Lu. 2024. ProcSAGE: an efficient host threat detection method based on graph representation learning. *Springer Cybersecurity* (2024).
[63] F. Yang, Jiacen Xu, Chun lin Xiong, Zhou Li, and Kehuan Zhang. 2023. PROGRAPHER: An Anomaly Detection System based on Provenance Graph Embedding. In *Security Symposium (USENIX Sec'23)*. USENIX.
[64] Limin Yang, Wenbo Guo, Qingying Hao, Arridhana Ciptadi, Ali Ahmadzadeh, Xinyu Xing, and Gang Wang. 2021. CADE: Detecting and explaining concept drift samples for security applications. In *Security Symposium (USENIX Sec'21)*. USENIX.
[65] Runqing Yang, Shiqing Ma, Haitao Xu, X. Zhang, and Yan Chen. 2020. UIScope: Accurate, Instrumentation-free, and Visible Attack Investigation for GUI Applications. In *Network and Distributed System Security Symposium (NDSS'20)*. The Internet Society.
[66] Jun Zeng, Chuqi Zhang, and Zhenkai Liang. 2022. PalanTír: Optimizing Attack Provenance with Hardware-enhanced System Observability. In *Conference on Computer and Communications Security (CCS'22)*. ACM.
[67] Jun Zengy, Xiang Wang, Jiahao Liu, Yinfang Chen, Zhenkai Liang, Tat-Seng Chua, and Zheng Leong Chua. 2022. SHADEWATCHER: Recommendation-guided Cyber Threat Analysis using System Audit Records. In *Symposium on Security and Privacy (S&P'22)*. IEEE.
[68] Zhaoqi Zhang, Panpan Qi, and Wei Wang. 2020. Dynamic Malware Analysis with Feature Engineering and Feature Learning. In *AAAI Conference on Artificial Intelligence (AAAI'20)*.

A Using PIDSMaker

A.1 Configuration Files

Each system is defined by a YAML configuration file that specifies all pipeline components and their parameters. The following is an excerpt from the Orthrus configuration:

orthrus.yml

```yaml
construction:
  node_features:
    subject: type, path, cmd_line
    file: type, path
    netflow: type, remote_ip, remote_port
  [...]
transformation:
  used_methods: none
featurization:
  used_method: word2vec
  emb_dim: 128
  epochs: 50
  word2vec:
    alpha: 0.025
    [...]
batching:
  intra_graph_batching:
    used_methods: edges, tgn_last_neighbor
    [...]
training:
  lr: 0.00001
  node_hid_dim: 128
  encoder:
    used_methods: tgn, graph_attention
    graph_attention:
      num_heads: 8
      [...]
  decoder:
    [...]
evaluation:
  used_method: node_evaluation
  node_evaluation:
    threshold_method: max_val_loss
    [...]
triage:
  used_method: depimpact
  use_kmeans: True
  [...]
```

CLI arguments can override any YAML parameter using dot notation (e.g., `--training.lr=0.0001`) and take precedence over YAML values. The `--force_restart` flag re-executes stages from a specified point even when arguments are unchanged, while —-restart_from_scratch writes outputs to a new directory to isolate runs.

A.2 Automatic On-Disk Caching

PIDSMaker is designed to execute large batches of compute-intensive experiments, often involving heavyweight architectures. To avoid

redundant computation and automatically reuse intermediate results, PIDSMaker implements an on-disk caching mechanism at each stage of the pipeline. Each stage computes a unique hash derived from its configuration arguments and the hash of its predecessor, and stores its outputs in a directory indexed by this hash. Before execution, the pipeline checks for the existence of the corresponding directory; if it is found, the stage is skipped and cached outputs are loaded instead.

This design enables efficient reuse of intermediate results across experiments. For instance, modifying only the learning rate reuses cached outputs from upstream stages such as graph construction, transformation, featurization, and batching, and re-executes only the affected training and evaluation stages. Overall, this caching strategy significantly reduces runtime and supports rapid, iterative experimentation.

A.3 Minimum System Requirements

PIDSMaker is designed to train deep neural networks on large volumes of system data and is therefore primarily intended to run on servers equipped with substantial memory capacity and CPU resources. We recommend a minimum of 100 GB of system memory for processing the smallest datasets, and at least 500 GB for the largest datasets in order to avoid resource constraints. To improve throughput, PIDSMaker keeps most data resident in memory, thereby reducing time-consuming disk I/O operations, which in turn necessitates significant memory availability. PIDSMaker supports execution on both CPUs and GPUs. However, we strongly recommend the use of a GPU with at least 20 GB of memory for small to medium-scale datasets, and at least 40 GB for larger datasets.

A.4 Hyperparameter Search

Table 3 shows the hyperparameters searched for each system on each dataset in our experiments. For consistent model selection, we select the best-performing epoch for each system.

Stage	Parameter	Search Space
Featurization	Embedding dimension	{64, 128, 256}
Training	Hidden dimension	{128, 256, 320}
Training	Learning rate	{0.001, 0.0001}

Table 3: Hyperparameter search space for consistent evaluation across all systems.