A First-Principles Evaluation of Graph-Based Network Intrusion Detection Systems

Rui Zhao
University of Virginia
Charlottesville, USA
dkw7xn@virginia.edu

Wajih Ul Hassan
University of Virginia
Charlottesville, USA
hassan@virginia.edu

Abstract

Graph-based network intrusion detection systems (GIDS) report strong benchmark detection metrics, but those metrics establish little about deployability. We approach the problem from first principles: rather than inheriting the preprocessing, windowing, and thresholding conventions of each published system, we ask what a controlled comparison requires and impose it uniformly. The result is GIDS-Eval, an evaluation framework that decomposes a GIDS into six interchangeable stages and turns those conventions into explicit experimental variables, so reported performance can be attributed to individual stages instead of whole pipelines. We survey nine representative GIDS, reimplement five of them within GIDS-Eval, and evaluate them on four datasets under one matched protocol. We identify nine recurring evaluation gaps and quantify the impact of each: two crafted edges achieve full evasion against three of the eight detector–dataset pairs with anything to hide; the snapshot window alone accounts for a mean 38.3% relative swing in average precision (AP); aligning preprocessing across systems moves AP by up to 61.8 percentage points for a single detector; and none of the 18 detector–dataset pairs we replay can alert as events arrive. We introduce GIDS-Lite, an encoder-free control built in the same framework, which ranks first by AP on two of the four datasets at up to 575× lower runtime. Architectural complexity is therefore not a consistent driver of detection quality under our matched protocol on current benchmarks, but it does enlarge the runtime, calibration, and attack surfaces operators must defend.

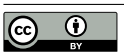
CCS Concepts

• **Security and privacy → Network security**.

Keywords

Network Intrusion Detection; Graph Neural Networks; Security Evaluation

ACM Reference Format:
Rui Zhao and Wajih Ul Hassan. 2026. A First-Principles Evaluation of Graph-Based Network Intrusion Detection Systems. In *Proceedings of the 2026 ACM SIGSAC Conference on Computer and Communications Security (CCS '26), November 15–19, 2026, The Hague, Netherlands.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3830454.3846576



CCS '26, The Hague, Netherlands

ACM ISBN 979-8-4007-2871-6/2026/11
https://doi.org/10.1145/3830454.3846576

1 Introduction

Network intrusion detection remains a core defense for enterprise security, particularly against attacks that unfold across many network events rather than a single malicious connection [23, 30, 33]. Graph-based Network Intrusion Detection Systems (GIDS) address this setting by applying graph neural networks and temporal models to network traffic, aiming to detect attacker behaviors such as lateral movement across hosts [14]. Recent GIDS [3, 9, 14, 15, 19, 21, 35] combine graph encoders such as GCNs [16] and GraphSAGE [10] with temporal modules such as GRUs [4] and recurrent graph models [9], and typically report very high headline detection metrics on public datasets, even as several acknowledge precision limitations. From a security standpoint, however, benchmark accuracy is not deployability. The architectural choices that produce a strong precision number on a fixed snapshot also define a graph-scoring path an attacker can target: in our experiments, two crafted edges inserted into the network graph hide every attack edge that state-of-the-art GIDS catch, while leaving every other detector input unchanged. The relevant security question is not whether a GIDS detector wins a clean benchmark, but whether its decisions survive an attacker who can shape the graph it reads, an operator who must wire it into a streaming sensor, and the pipeline choices that current GIDS publications leave undocumented.

This deployability gap has three practical consequences. First, complex graph and temporal designs enlarge the attack surface, as the covering-edge result above demonstrates. Second, they increase training time, memory demand, and tuning burden, raising operational cost. Third, existing GIDS results do not establish what actually drives the reported performance, because each paper bundles a different preprocessing pipeline, snapshot window, feature representation, thresholding procedure, and metric choice. Performance differences across papers therefore cannot be attributed to the graph or temporal architecture itself. An operator deploying a recent GIDS detector inherits all three problems and cannot tell whether the added complexity yields any defensive benefit.

Recent evidence suggests this concern is not theoretical. The R+R study by Wang *et al.* [32] re-ran five published GIDS end to end and showed that reported results vary substantially across preprocessing choices, dataset splits, and evaluation protocols, treating each system as a black box. In parallel, Bilot *et al.* [2] showed that, in provenance-based intrusion detection (PIDS), a simple neural network matches or outperforms much more complex graph-based systems under a unified evaluation, and identified nine PIDS evaluation shortcomings. Neither study establishes whether the architectural complexity of modern GIDS provides a real defensive advantage on the axes that matter to a security operator: adversarial robustness,

System	Graph Setting	Core Technique
Implemented in our modular framework		
● Argus [35]	Discrete temporal graph over enterprise logs	MPNN encoder with edge features, attack-aware decoder, and AP-oriented loss
● Euler [15]	Discrete temporal host graph	Graph encoder plus sequence encoder for temporal link prediction
● Pikachu [21]	Dynamic graph over time-ordered traffic events	Temporal walks, Skip-gram embeddings, and GRU autoencoder
● VGRNN [9]†	Dynamic graph snapshots	Variational graph recurrent model for latent node embeddings and link reconstruction
● Anomal-E [3]	Flow graph with edge attributes	E-GraphSAGE and modified DGI with a classical anomaly detector
Surveyed but not implemented		
● NetWalk [37][a]	Dynamic network stream	Clique-based network embedding with deep autoencoder regularization and clustering
● Jbeil [14][b]	Continuous-time authentication graph	TGN-style memory encoder and decoder for inductive lateral-movement link prediction
● E-GraphSAGE [19][c]	Flow graph with edge attributes	Edge-aware GraphSAGE for supervised intrusion detection on flow-based traffic graphs
● ARGANIDS [31][d]	Flow graph with node features	Adversarially regularized graph autoencoder for unsupervised intrusion detection

Table 1: Representative GIDS considered in this study. The upper block lists the five systems implemented in GIDS-Eval. The lower block lists additional surveyed systems that define the broader design space but are excluded from full modular evaluation for the reasons in the footnotes.

† A general dynamic-graph model rather than a GIDS, studied as the temporal graph encoder backbone later GIDS adopted for intrusion detection. [a] Runtime is impractical for full system-dataset ablations. [b] The released implementation leaves the paper's preprocessing and evaluation protocol underspecified and omits details needed for faithful modularization. [c] The supervised design requires extensive labeled training data, outside our evaluation protocol. [d] No public implementation is available, and the paper lacks the detail needed for faithful modularization.

calibration stability under streaming traffic, and detection latency. We answer that question directly.

To do so, we build GIDS-Eval, a modular evaluation framework that decomposes each GIDS into six interchangeable stages: feature extraction, graph transformation, featurization, encoding, decoding, and optimization. We survey nine representative GIDS and implement five within GIDS-Eval (Table 1), running 1,370 experimental jobs in total: 960 of them are the 240-configuration component grid of EG7 run on each of the four datasets, and the rest are the per-gap sweeps of §4. The decomposition turns non-architectural choices such as snapshot windowing and threshold calibration into explicit experimental variables, so performance differences can be attributed to specific stages rather than whole pipelines. Within GIDS-Eval, we introduce GIDS-Lite, a deliberately simple control that scores each network event using only featurization and a shallow predictor, without a graph or temporal encoder. GIDS-Lite is not another specialized architecture. It is a control that tests under a matched protocol whether the complexity of recent GIDS yields a detection or robustness advantage that justifies its cost.

Through GIDS-Eval, we identify nine recurring evaluation gaps that limit the practical security value of current GIDS. The most security-critical is the adversarial fragility above, which a gradient-guided covering-edge attack [36] exposes (EG6). The remaining gaps show that even before an adversary intervenes, reported performance is dominated by non-architectural pipeline choices a real deployment cannot avoid: snapshot-window choice alone swings average precision (AP) by a mean of 38.3% across systems and datasets (EG1), aligning the preprocessing pipeline across systems shifts AP by up to 61.8 percentage points for a single detector–dataset pair (EG8), and none of the 18 detector–dataset pairs we replay can alert as events arrive (EG9). The others, from seed- and test-period-dependent thresholding to edge attributes that are near-collinear across attack and benign traffic, are quantified in §4. Each gap affects the security guarantees a GIDS can offer in deployment, and most are not examined in the original publications.

These findings extend prior work in two directions. R+R established *that* GIDS pipeline choices matter while treating each system as a black box. GIDS-Eval attributes the variance to specific stages, quantifies each as a single axis with the numbers above, and uses GIDS-Lite as a control R+R does not propose. The PIDS results of Bilot *et al.* do not transfer cleanly because the data models differ: provenance graphs are given by kernel audit causality [12], labeled under converged DARPA Transparent Computing (TC) conventions [5], and described by symbolic features such as file paths and command lines [24], whereas GIDS must build the graph from a continuous flow stream, have no canonical preprocessing contract, and rely on statistical edge attributes that look almost identical for attack and benign traffic. Of the nine gaps we identify, two are measurements neither prior study provides (EG1, EG8). Three sit on axes Bilot *et al.* also examine, with the answer inverted on network flows (EG4, EG6, EG7). Three sharpen shared evaluation-hygiene concerns into operator-facing measurements (EG2, EG3, EG5). One, streaming readiness (EG9), carries the deployment axis Bilot *et al.* address for PIDS to GIDS with a replay measurement. This paper thus extends R+R from black-box reproduction to stage-level attribution, and extends Bilot *et al.* from PIDS to a network setting whose data model creates gaps PIDS evaluation cannot encounter.

Under the matched protocol of GIDS-Eval, one fixed GIDS-Lite configuration ranks first by AP on two of the four datasets while reaching up to ~575× lower end-to-end runtime than the slowest system and avoiding the graph-scoring path the covering-edge attack exploits. On the other two datasets the winners are different architectures, Pikachu and VGRNN. Taken together with the gap results, this shows that the architectural complexity of recent GIDS is not a consistent driver of detection quality on current benchmarks, while it does enlarge the runtime, calibration, and adversarial attack surfaces an operator must defend.

We make the following contributions:

- We present GIDS-Eval, a modular evaluation framework that maps representative systems into six shared, interchangeable stages, and run 1,370 jobs, among them a 240-configuration component grid, to attribute reported GIDS performance to specific stages rather than whole pipelines.

System	Feature Extraction	Graph Transform.	Featurization	Encoding	Decoding	Optimization
● Argus [35]	src/dst IP, port, flow stats	Temporal snapshots	Flow features	MPNN + GRU	Edge scoring	Avg. precision loss
● Euler [15]	src/dst host identifiers	Temporal snapshots	Identity emb.	GCN + GRU	Adj. reconstruction	Reconstruction loss
● Pikachu [21]	src/dst IP, port, protocol	Temporal snapshots	Random walk emb.	Skip-gram + GRU	Link prediction	Binary cross-entropy
● VGRNN [9]	src/dst host identifiers	Temporal snapshots	Identity emb.	GCN + GC-LSTM	Adj. reconstruction	Recon. + KL-div. loss
● Anomal-E [3]	src/dst IP, flow statistics	Static graph	Flow features	GraphSAGE	Node feat. prediction	Node prediction loss

Table 2: Stage-level decomposition of the five systems implemented in GIDS-Eval. Unlike Table 1, which defines the broader study scope, this table maps only the implemented systems to the six stages exposed by the modular framework. Encoding lists the graph encoder and, where applicable, the temporal encoder separated by "+". emb. = embedding. Recon. = reconstruction.

- We identify nine recurring evaluation gaps that limit the practical security value of current GIDS, and classify each against the two prior studies as novel, inverted, confirmed, or adapted.
- We show, via a gradient-guided covering-edge attack, that two inserted edges hide every attack edge state-of-the-art GIDS catch, and that six of the eight detector–dataset pairs with anything to hide fall within twenty, exposing a graph-side attack surface their architectural complexity creates rather than mitigates.
- We introduce GIDS-Lite, a first-principles baseline that, in a single configuration fixed on validation data, ranks first by AP on half the datasets at a fraction of the runtime.

Availability. The source code and paper artifact are available at https://github.com/DART-Laboratory/GIDS-Eval.

2 Background & Study Scope

This section summarizes the design-space progression of recent GIDS and the systems we study. GIDS-Eval is introduced in §3.

2.1 Evolving GIDS Complexity

Recent GIDS occupy a spectrum of design choices. At the simpler end, systems such as E-GraphSAGE [19], Anomal-E [3], and ARGANIDS [31] construct a traffic graph, apply a graph encoder or reconstruction objective, and treat classification, reconstruction error, or representation quality as the anomaly signal. The modeling assumption is that malicious activity distorts local graph structure or endpoint neighborhoods in a way the encoder can pick up. Later designs add temporal modeling on top of this graph abstraction: Euler [15] predicts links over temporal host graphs, Pikachu [21] combines temporal walks with a GRU autoencoder, and VGRNN [9] uses variational graph recurrent dynamics over evolving snapshots, so the anomaly score depends jointly on graph construction, temporal windowing, sequence modeling, and the prediction target. At the most coupled end, Argus [35] combines message passing over temporal enterprise graphs with edge features, an attack-aware decoder, and an AP-oriented loss, and Jbeil [14] uses memory-based temporal graph modeling for inductive lateral-movement link prediction. Here every stage shapes the anomaly signal together.

2.2 Representative Systems

Table 1 lists the nine representative GIDS in our survey, spanning flow and host graphs, static and temporal graph construction, graph and temporal encoders, reconstruction and prediction objectives, and separate anomaly scoring components. From this set, we select five for in-depth evaluation. Three criteria governed inclusion: the system targets network-level intrusion detection, its design appeared at a peer-reviewed venue, and its release is complete enough, as runnable code or a fully specified description, to rebuild faithfully from interchangeable modules. VGRNN meets the first criterion indirectly. It was published as a variational graph recurrent model, not as an intrusion detector, and we evaluate the intrusion-detection instantiation that later GIDS work built on it [15, 32]. In this paper it is a temporal graph encoder backbone, not a purpose-built GIDS. The remaining systems stay in the survey scope but are excluded from full evaluation for insufficient public implementation detail, supervision requirements outside our protocol, or scale constraints. Faithful modularization matters because each implemented system is decomposed into independent components that GIDS-Eval re-combines and varies across configurations.

3 The GIDS-Eval Framework and Protocol

Modular design. GIDS-Eval starts from what a controlled comparison requires instead of adopting any single system's preprocessing, windowing, and thresholding conventions, and imposes those requirements uniformly. The staged decomposition that makes this possible follows the approach Bilot *et al.* [2] introduced for provenance-based systems, adapted here to the network-flow setting. To vary one pipeline stage while holding the rest fixed, we integrate the five studied detectors as interchangeable configurations of GIDS-Eval, whose shared infrastructure builds on PyTorch and PyTorch Geometric [7]. The columns of Table 2, which follows the form of their PIDS systemization with the stages redefined for network flows, define the six stages every integrated system shares. Raw network records enter at **feature extraction**, which selects the fields that populate the graph. **Graph transformation** builds a static or temporally snapshotted graph from those records. **Featurization** attaches numeric attributes to nodes and edges. **Encoding** learns vector representations with a graph encoder and, optionally, a temporal encoder. **Decoding** maps the representations to a per-edge, per-node, or per-graph anomaly signal. Finally, **optimization** supplies the loss that trains the encoder and decoder. Each cell in Table 2 corresponds to a module, and modules within the same stage are interchangeable.

A single YAML file selects, for each stage, one module and its hyperparameters, so every experiment is fully described and reproducible from that file (Figure 1). GIDS-Eval hashes each stage's inputs and caches the outputs of every stage but the last in content-addressed folders, so a stage whose inputs are unchanged is skipped and its cached output feeds directly to the next stage. Replacing a

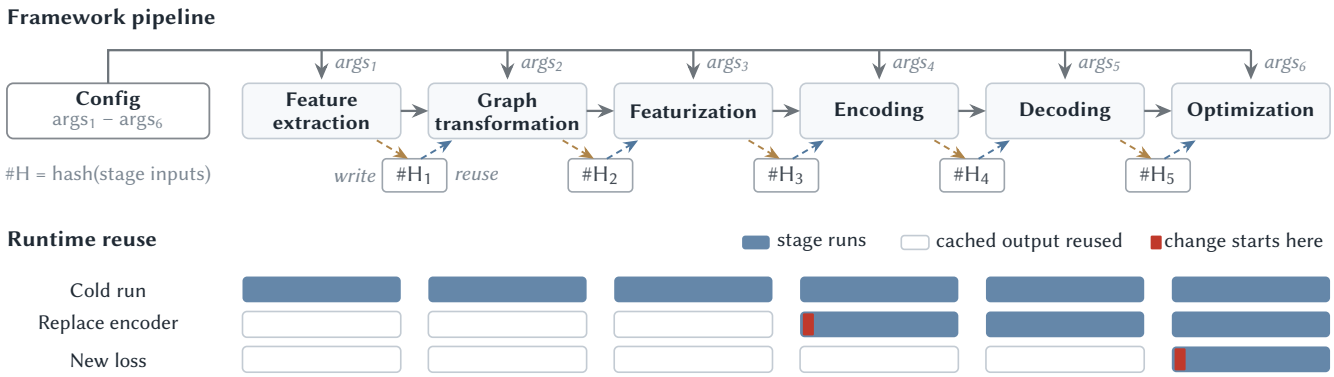


Figure 1: The GIDS-Eval pipeline. A configuration supplies one argument block per stage. Every stage but the last writes its output to a cache keyed by a hash of its inputs, and a stage with unchanged inputs is served from that cache: replacing the encoder re-runs only the encoding, decoding, and optimization stages, and changing only the loss re-runs the final stage alone.

Table 3: Datasets used in our evaluation. Hosts/IPs counts unique endpoints by native identifier (host names for Campus, LANL, and OpTC, and IP addresses for CI-2017).

Dataset	Hosts/IPs	Events	Days	Attack labels
Campus [32]	18.4M	10.3B	101	APT campaigns
LANL [13]	17.7K	1.18B	58	Red team
OpTC [6]	814	10.1M	8	Red team
CI-2017 [17]	19.1K	2.10M	5	Scripted attacks

graph encoder, for example, invalidates only the encoding, decoding, and optimization stages, while earlier stages are reused across configurations that share the same upstream choices. This caching is what makes the 240-configuration component-attribution sweep in EG7 tractable across Campus, CI-2017, LANL, and OpTC.

Datasets. We evaluate on four datasets that span distinct deployment environments, telemetry types, scales, and attack-labeling mechanisms. Campus is a large-scale enterprise gateway trace introduced in prior R+R work [32]. LANL [13] provides authentication and flow telemetry with human red-team activity, the setting in which Active Directory attacks unfold [18]. OpTC [6] provides mixed host and network telemetry from a simulated enterprise engagement, and CI-2017, the corrected-label variant [17] of CIC-IDS-2017 [25], provides flow-level records with scripted attacks in a controlled lab environment. We preserve temporal order when constructing train, validation, and test splits. Table 3 reports the per-dataset statistics.

Under the shared protocol, the unit of detection is defined per dataset as follows. On LANL, an edge is a source-to-destination authentication event, with no restriction to a particular authentication protocol, and it is malicious if it matches the red-team ground truth. On OpTC, an edge is a source-to-destination host event, malicious if its time and endpoint pair match a red-team event. On CI-2017, an edge is a flow between a source and a destination IP address, labeled directly by the corrected dataset labels. On Campus, an edge is a source-to-destination event from the released R+R trace [32], malicious if it belongs to one of the injected campaigns.

Under the GIDS-Lite split rule of §5.1 (chronological, validation fraction 0.33, 50,000-edge training cap), Campus has 29,274 edges for training, 14,107+384 benign+attack for validation, and 29,065+355 for test. LANL has 506, 4,563+649, and 8,971+53. CI-2017 has 50,000, 448,257+184,482, and 762,531+332,928. OpTC has 50,000, 12,597+3,494, and 230+9,389.

The four datasets differ in fidelity. Campus is the weakest case: its attack traffic is injected as scripted APT campaigns [32], so its attack distribution need not match how those attacks appear in production, and it is one of the two datasets on which no evaluated GIDS reaches a deployable operating point. OpTC is likewise generated in a simulated enterprise and carries synthesis artifacts of its own. For CI-2017 we use the corrected-label variant [17] rather than the original release, avoiding the labeling and construction problems documented for benchmark NIDS datasets [8]. We keep all four datasets, and all four are publicly available: their contrasts across telemetry type, scale, and labeling are what make a cross-dataset finding more than a property of one benchmark. No single dataset should be read as a proxy for deployment.

Metrics. We measure detector behavior along four dimensions: score ranking, thresholded detection, campaign coverage, and execution cost. For score ranking, we report AP and AUC before any threshold is chosen, treating AP, the area under the precision–recall curve, as the primary threshold-free metric because attack events are rare and ROC-style metrics can hide false-alert burden [29]. For thresholded detection, we select thresholds on the validation split, compute TP, FP, FN, and TN, and report TPR, FPR, precision, recall, and F_1. For execution cost, we record training time, inference time, wall-clock time, peak memory, and CPU utilization.

Event-level metrics do not capture whether a detector exposes a multi-stage intrusion campaign: a system can score well in aggregate while missing every step of an attack, or expose a campaign with a single high-confidence alert. Motivated by the ADP metric of Bilot *et al.* [2], we introduce CDR for GIDS evaluation. Let C be the set of attack campaigns in the evaluated test period and $\hat{C}$ the campaigns for which the detector flags at least one positive event. We define CDR as $|\hat{C}|/|C|$.

Hardware setup. We ran all experiments on virtual machines hosted on a dual-socket AMD EPYC 9654 server with 192 physical cores, 384 hardware threads, and 800 GB RAM, running Ubuntu 22.04 with Linux kernel 6.5.0. Each VM had 36 vCPUs and 128 GB RAM and only the software required for the evaluated system. Datasets were stored on the host and accessed through VFS, isolating each software environment on shared hardware.

Implementation and validation. All five studied systems provide public source code. We start from the released implementations of Argus, Euler, Pikachu, VGRNN, and Anomal-E, and preserve their model architectures, preprocessing, training objectives, hyperparameters, and scoring logic. Our changes are limited to wrapping each codebase behind the six-stage module interface above: the wrappers translate input and output formats, set dataset paths, and collect runtime logs, but do not replace the detector-specific learning code. We validate each wrapper by running the original code on the datasets and settings supported by its release and checking that the wrapper produces consistent results, using the configuration reported in the corresponding paper when a release exposes multiple settings. For experiments that test a specific design choice, such as the snapshot window, we change only that module while keeping the rest of the released pipeline fixed.

4 Evaluation Gaps

We use *evaluation gap* for an aspect of a GIDS that current evaluations fail to isolate or report, and each of the nine gaps below is stated in that form. Four are failures to **isolate** a design choice from the result it influences: the snapshot window (EG1), the edge-feature path (EG4), the individual pipeline components (EG7), and the preprocessing pipeline (EG8). One is a failure to **separate** calibration from test evaluation (EG3), which is also a design choice of the one system that uses it. The remaining four are failures to **measure or report** a dimension on which a deployed detector is judged: threshold variance (EG2), resource cost (EG5), adversarial robustness (EG6), and event-to-alert latency (EG9). The isolate and separate gaps concern experimental control, the measure-or-report gaps concern reporting coverage, and a system carries a mark in Table 7 when its own evaluation leaves that gap open, because its design confounds the dimension or because it never measured it.

The nine gaps examined below emerged from our component-isolation experiments on the five systems of §2.2, and §5 provides the complementary end-to-end comparison under the shared protocol. Space permits only the headline result for each gap, and Appendix F of the full version [40] reports the complete numbers.

EG1: Snapshot-Window Sensitivity

4/5 systems

The snapshot window used to group event streams into graphs is a hidden evaluation variable. Holding the detector and dataset fixed while varying only this window produces more than 20% relative AP swing in 11 of 16 detector–dataset pairs, showing that reported accuracy can depend heavily on an under-documented preprocessing choice.

Many GIDS first divide events into snapshot windows and build one graph per window. Short windows yield sparse graphs that may split related attack steps, while long windows yield dense graphs that can hide short-lived attack signals in normal activity, so the snapshot window is a model input, not an implementation detail.

Four of the five systems depend on such a window. Argus, Euler, and VGRNN construct fixed-duration graph snapshots. Pikachu's temporal preprocessing uses a window to assign events to snapshots and align training with the first attack snapshot. Anomal-E is excluded as a static-graph baseline. For each windowed system we sweep four snapshot window sizes per dataset: 30 min–2 h for LANL, 6 min–150 min for OpTC, 10 min–50 min for CI-2017, and 75 s–450 s for Campus. Among the studied systems, Euler and Argus already report window-size ablations in their original evaluations. Our sweep asks what a single-system ablation cannot answer, which is whether the sensitivity is model-specific or systemic.

We measure sensitivity as the relative AP swing across the tested windows, computed as (best − worst)/best. Figure 2(a) shows the effect is not limited to one detector or dataset: 11 of 16 detector–dataset pairs exceed a 20% relative AP swing, with mean 38.3%. The largest absolute changes are enough to change the conclusions of an evaluation: Argus on OpTC falls from 0.760 at its 6-minute default to 0.033 at a 150-minute window, Pikachu on CI-2017 changes by 0.358 AP, and Euler on OpTC changes by 0.259 AP.

The best window is not consistent across detectors or datasets. On OpTC, the best window is 6 min for Argus and Euler but 150 min for VGRNN and Pikachu. On Campus, it ranges from 75 s for VGRNN to 450 s for Argus and Pikachu. No single snapshot window is uniformly safe: a window that improves one system can degrade another, and a window that works on one dataset may be a poor choice on another. The risk grows when a detector has multiple preprocessing and temporal modules, because the same window choice can simultaneously alter graph density, temporal alignment, and the inputs passed to downstream encoders.

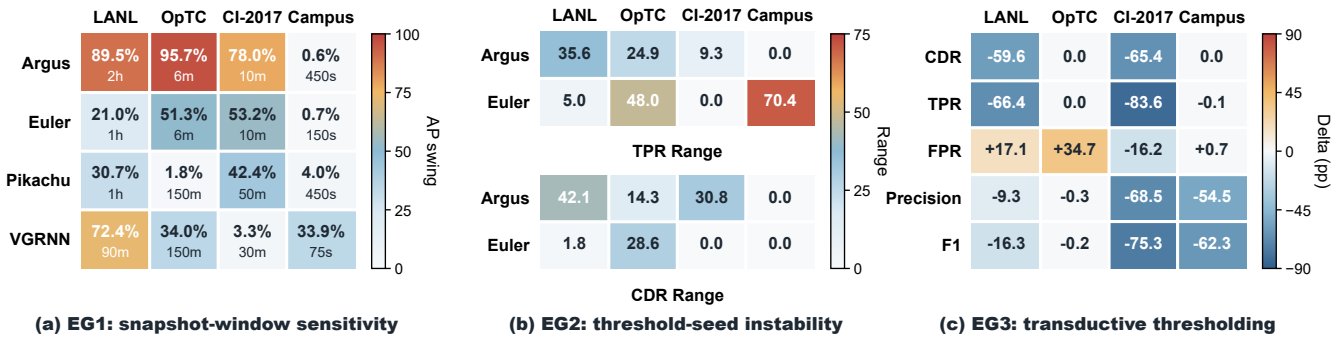
If a paper reports only one snapshot window, or omits it entirely, readers cannot tell whether the reported accuracy comes from the detector or from a favorable window. This problem is especially important for AP, our primary threshold-free metric under rare attacks. AUC is less sensitive (mean relative swing 12.7%), but it also hides false-alert burden that matters in intrusion detection.

Recommendations. A complete evaluation documents the snapshot window and its selection rule per dataset, treats the window as a tunable parameter, characterizes sensitivity across sizes, and never derives the window from the test set. When possible, evaluations should include snapshot-free or streaming baselines to separate temporal discretization from detector architecture.

EG2: Threshold-Selection Instability

2/5 systems

Detector scores become alerts only after a threshold is chosen. When that threshold is calibrated from a random validation sample, repeated runs can move the operating point and substantially change event-level and campaign-level detection. The largest TPR and CDR ranges in our experiments reach 70.4 and 42.1 percentage points.



(a) EG1: snapshot-window sensitivity (b) EG2: threshold-seed instability (c) EG3: transductive thresholding

Figure 2: Per-gap sensitivity heatmaps. Cell values report the main effect size for each setting: relative AP swing in (a), threshold-induced TPR/CDR range in percentage points in (b), and inductive-minus-transductive metric deltas in percentage points in (c). Smaller text in (a) reports the window with the highest AP.

Post-threshold metrics (TPR, FPR, precision, F_1, CDR) depend on a detection threshold that converts anomaly scores into binary alerts, so the threshold is part of the evaluation protocol: two runs with similar score rankings can produce different alerts if they choose different thresholds.

Argus and Euler calibrate this threshold from a random validation sample: both sample a subset of training events, compute an anomaly-score distribution on that subset, and use a percentile of that distribution as the threshold. Since the sample changes with the seed, the threshold also changes, even when the detector, dataset, snapshot window, and remaining pipeline settings are fixed. We exclude Pikachu and VGRNN because they do not use this rule (Pikachu has a separate thresholding issue covered in EG3), and Anomal-E because it is a static-graph detector without snapshot-level threshold calibration.

We run Argus and Euler on all four datasets at their default EG1 windows, with ten seeds per detector–dataset pair, and measure each metric's range across seeds. Figure 2(b) shows the resulting spread. The instability is not limited to one detector or dataset. Euler on Campus has a TPR range of 70.4 percentage points, and Euler on OpTC has a TPR range of 48.0 and a CDR range of 28.6 percentage points. Argus on LANL has a TPR range of 35.6 and a CDR range of 42.1 percentage points, with up to 24 attack campaigns changing between detected and missed across seeds. Argus on CI-2017 has a CDR range of 30.8 percentage points.

This instability matters because it affects the operating point a deployment would use. A threshold-free metric hides the problem because it evaluates score rankings before a boundary is chosen, while TPR, FPR, precision, F_1, and CDR depend directly on that boundary. When the threshold is selected from a small random validation subset, the same system produces different alerts and detects different attack campaigns across runs, making the evaluation hard to reproduce or compare against another system.

Recommendations. Threshold selection should be deterministic and reported as part of the evaluation protocol. Systems should choose the validation set once and report which time period it covers rather than drawing a new random subset per run. Determinism does not guarantee that the chosen threshold transfers. A fixed validation period can be unlike the test period it precedes: on LANL, 12.5% of validation edges are malicious against 0.6% of test edges, so a threshold well calibrated on one is not automatically well calibrated on the other. What determinism buys is that the residual error is a property of a stated protocol rather than an unreported seed, and that the operating points in Table 8 can be reproduced exactly. Papers should report the threshold-selection rule, the validation data used for calibration, and the final threshold. If random calibration is unavoidable, evaluations should report variance across seeds for all post-threshold metrics, including CDR.

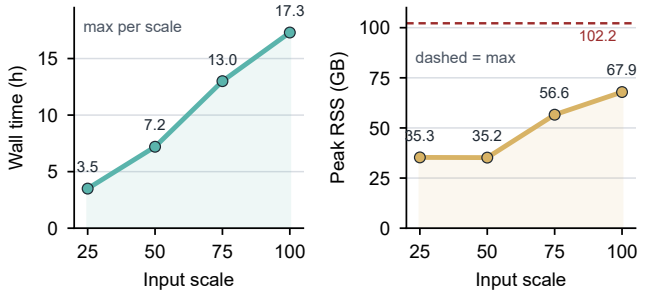


Figure 3: Runtime and memory scaling in EG5. Points report the largest value over detector–dataset pairs at each input scale. The dashed line marks the largest peak RSS observed across all measurements.

EG3: Transductive Thresholding

1/5 systems

A threshold chosen from evaluation data is not a deployable threshold. When the threshold is fixed before the test period, the same detector can lose campaign coverage entirely on some datasets and move to a very different operating point.

Pikachu selects its anomaly threshold from the evaluation scores themselves: the detector computes anomaly scores for post-training edges and then chooses the threshold from those same labeled edges. This is a transductive protocol [22] because the threshold is fitted to the data on which the detector is later evaluated, a choice unavailable in deployment, where the threshold must be chosen before the future test period is observed. Selecting the operating

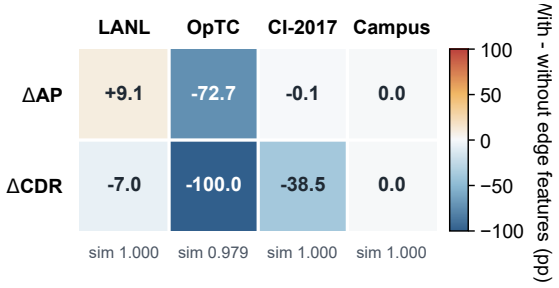


Figure 4: EG4 edge-feature effect. Cells report with-feature minus without-feature AP and CDR for Argus in percentage points. Smaller text gives attack–benign edge-feature cosine.

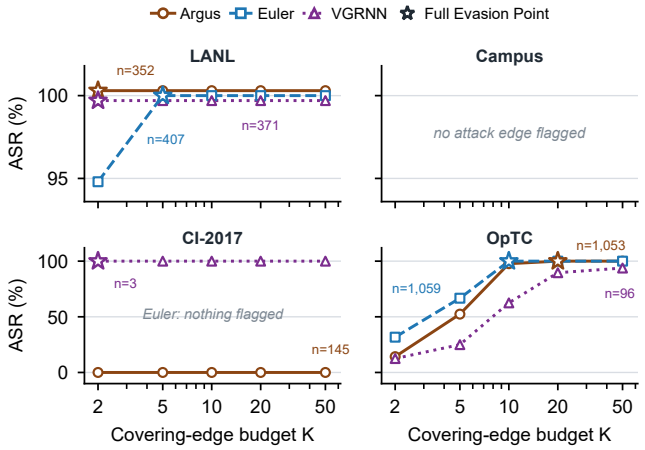


Figure 5: Adversarial covering-edge evasion in EG6. ASR is computed over the attack edges each detector flags, whose count n is printed beside its curve. A star marks the smallest budget at which no flagged edge survives. A detector with no line flagged no attack edge, so the attack has nothing to evade. The LANL panel uses a magnified axis, and its saturated curves carry a small vertical offset so all three stay visible.

point from the test period is a lapse of basic machine-learning evaluation hygiene rather than a failure specific to GIDS. The data-snooping pitfall is already documented in security evaluation [1]. We examine it here because Pikachu is the only one of the five studied systems that uses this protocol and because its effect on reported results is large.

We compare this transductive protocol with an inductive alternative. The inductive protocol splits the post-training snapshots chronologically, chooses the threshold on the first half, and keeps it fixed for the remaining test snapshots.

Figure 2(c) shows that the protocol change substantially shifts reported behavior. On LANL, CDR drops from 59.6% to 0.0% and TPR drops by 66.4 percentage points. On CI-2017, CDR also drops to 0.0% while TPR and F_1 decrease by 83.6 and 75.3 percentage points. The original threshold is therefore not a post-processing detail: it can determine whether any attack campaign is detected at all.

The change is not always a simple decrease: on Campus, TPR and FPR barely move under the inductive threshold while precision drops by 54.5 percentage points, and on OpTC, TPR and CDR remain unchanged while FPR rises by 34.7 percentage points. Transductive calibration tunes the threshold on the test period while inductive calibration fixes it beforehand, so the reported result reflects a more permissive protocol rather than a deployable one.

Recommendations. Threshold calibration should be separated from test evaluation: a paper should state where the threshold comes from, whether labels are used, and whether it is fixed before the test period, with labeled calibration data from a known time range before the test data. Transductive thresholding should be reported as an oracle setting, not as the main deployment result.

EG4: Untested Edge-Feature Value

2/5 systems

Evaluations treat edge features as a source of signal without isolating the feature path or testing whether it separates attack from benign traffic. Isolating it in Argus changes results in dataset-dependent ways and can remove campaign detection entirely, while attack and benign feature means stay near-collinear on every dataset.

Argus is the only system that provides an explicit edge-feature switch, isolating these features without changing the detector architecture. We compare two versions of Argus on each dataset, one using the original edge features and one with the edge-feature path disabled, with all other details fixed. Anomal-E carries the mark as well: its featurization is built on flow attributes that its released evaluation never isolates, and it exposes no comparable switch, so the same ablation cannot be run for it. The remaining three systems consume no edge features at all.

As shown in Figure 4, edge features are not a reliable source of improvement for AP or CDR. On OpTC, enabling edge features at the default 6-minute window reduces AP from 0.760 to 0.033 and CDR from 100.0% to 0.0%. This comparison and the EG1 window sweep both depart from the same baseline configuration (6-minute window, feature path off), so the 0.760 endpoints coincide while the two 0.033 endpoints are distinct measurements of different single-factor changes. On CI-2017, AP remains near zero but CDR drops from 65.4% to 26.9%. LANL shows the opposite metric pattern: AP improves from 0.014 to 0.105 while CDR still drops from 71.9% to 64.9%. Campus changes little in AP and detects no campaigns either way. Adding edge features does not simply make the detector stronger: it can change which attacks are found and in some cases removes campaign-level coverage entirely.

The mean attack and mean benign edge-feature vectors are nearly collinear on every dataset (cosine 0.979 to 1.000, in the smaller labels of Figure 4). Because these features are non-negative, collinearity of means is weak evidence on its own; the ablation above is what shows the feature path is not a reliable signal.

Recommendations. Evaluations should not assume edge features are beneficial just because they add traffic attributes. Authors should report ablations with and without edge features, and test whether

Table 4: Best-performing option in each GIDS design dimension. For each dataset and each dimension, we average AP over all configurations that use the same option and report the option with the highest average AP. Identity denotes a fixed Gaussian embedding keyed to the node identity. No single feature choice, encoder, temporal model, or training target is best across all datasets.

Dimension	Campus	CI-2017	LANL	OpTC
Featurization	id.+struct. (.216)	struct. (.553)	struct. (.239)	identity (.987)
Graph encoder	MPNN (.269)	MPNN (.601)	none (.254)	GraphSAGE (.971)
Temporal enc.	GRU (.198)	LSTM (.555)	none (.210)	GC-LSTM (.972)
Objective	reconstr. (.427)	reconstr. (.782)	reconstr. (.269)	link pred. (.989)

attack and benign feature distributions are separable before treating flow attributes as reliable detection signal. Feature overlap is not only a dataset artifact: an adversary can force it. In a replay attack, recorded benign flows are reused to carry malicious interactions, so edge-feature values match the benign distribution by construction rather than by accident. Under such an attack the feature path carries no signal by design, and the remaining evidence is structural: which hosts interact, and how often. Evaluations should therefore include replayed attack traffic alongside standard attacks and verify that detection does not depend on flow attributes alone.

EG5: Scalability Limitations

5/5 systems

Accuracy-only evaluations hide the cost of running GIDS at scale. Even on benchmark workloads with up to 49.3M input events, our runs can take more than 17 hours and reach 102.2 GB peak memory, making such costs hard to sustain on continuous enterprise traffic.

Most GIDS evaluations focus on accuracy and report little about resource cost, but graph-based detectors must build large graph inputs, extract features, train or apply models, and compute anomaly scores, all of which grow with event volume. A detector that is accurate on a fixed benchmark may still be hard to use if its runtime or memory grows quickly with the event stream.

We measure this cost by scaling detector-ready inputs. For each detector–dataset pair, we hold the detector's default snapshot window fixed and vary only the input scale across 25%, 50%, 75%, and 100% of the corresponding workload, recording wall-clock time and peak resident memory. Figure 3 shows scalability is not a minor detail. At 100% scale, the largest benchmark contains 49.3M input events, the longest run takes 17.3 hours, and peak memory in the scale experiment reaches 67.9 GB. Scaling is uneven: the largest 25%–100% runtime growth is 14.9× and the largest memory growth is 4.5×. Across all measurements, including the EG1 snapshot-window experiments, peak memory reaches 102.2 GB.

Reduced workloads can make a detector look more practical than it is at full scale. The largest runtime and memory growth occur on different detector–dataset pairs, so scalability is not a single system-wide constant, and a result on a reduced workload does not reliably predict full-scale cost. Peak memory also depends on preprocessing choices outside the core model: EG5 holds the snapshot window fixed to isolate input-scale effects, but EG1 shows that changing the snapshot window can raise memory further. The additional maximum marker in Figure 3 reflects this gap: the largest memory across all measurements is substantially higher than the largest memory under the EG5 default-window setting.

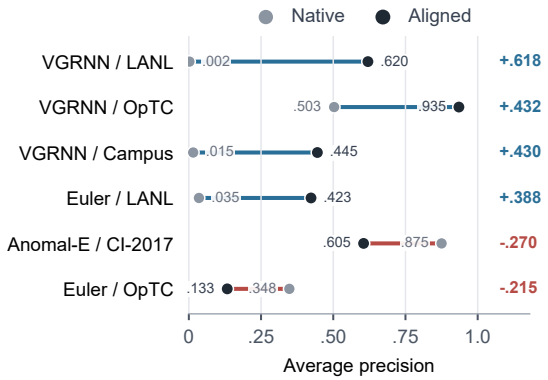


Figure 6: Impact of preprocessing alignment in EG8 for the six detector–dataset pairs with the largest AP shifts. Table 15 of the full version [40] lists all 18. Native AP uses each system's own pipeline, aligned AP the shared contract.

Table 7 marks a system for EG5 when its original evaluation does not report resource cost and how that cost grows with input scale, and all five systems carry the mark on those reporting grounds. Our measurements separate the reporting gap from the growth itself, which is uneven. The empirical runtime exponents in Table 10 of the full version [40] stay at or below 0.6 for Euler and 0.8 for VGRNN on every dataset, and Anomal-E is close to linear on the two datasets where it runs, so these systems scale sub-linearly to linearly in our sweep. The one case above the 1.5 super-linear threshold is Pikachu on LANL, whose exponent of 2.0 corresponds to the 14.9× runtime growth noted above. Benign scaling in our sweep does not clear the mark: without reported cost, an evaluation cannot show that a detector remains affordable at deployment scale.

Recommendations. GIDS evaluations should report resource usage together with accuracy. At a minimum, papers should report input scale, snapshot window, wall-clock time, peak memory, and hardware configuration for each evaluated detector. When reduced workloads are used, evaluations should also report how runtime and memory change with input scale rather than treating scalability as an implementation detail.

EG6: Unevaluated Adversarial Robustness

3/5 systems

Adversarial robustness is missing from GIDS evaluation: the studied systems are assessed on clean traffic only. Instantiating a single covering-edge attack shows what this omission hides. Four of the 12 detector–dataset pairs flag no attack edge to begin with. Of the eight that do, six lose every flagged edge within 20 inserted edges and three within only two.

The gap here is broader than any single attack: the original evaluations of the systems we study assess detection on clean traffic only, so their deployability claims say nothing about an adversary who adapts. One attack cannot settle how robust these systems are, but it can show what the missing evaluation hides.

We demonstrate this with the covering-edge evasion attack of Xu *et al.* [36], which recent R+R work [32] uses on GIDS as a state-of-the-art robustness test. It applies when a detector scores suspicious behavior as graph edges and an adversary can add benign-looking covering edges during evaluation. In GIDS those are network or host interactions inserted around an attack event to change the graph evidence the detector sees. From a flagged target edge, the attack inserts covering edges until the target is stealthy while the inserted edges stay undetected. This is no random edge-insertion test: it uses the detector's own graph-scoring path to pick perturbations that reduce the evidence around the target.

For each detector–dataset pair we take every attack edge in the test period as a candidate and record whether the detector flags it at that pair's own threshold: the threshold of the released checkpoint where the authors publish one (LANL, OpTC), and the threshold our reproduction calibrates otherwise (Campus, CI-2017). An edge already missed cannot be evaded, so we run the attack only on the flagged ones, with budgets $K \in \{2, 5, 10, 20, 50\}$. Attack success rate (ASR) is the fraction of *flagged* targets made stealthy within K, so full evasion, ASR 100%, means no edge the detector caught survives. A detector that flags nothing has no ASR to report.

We instantiate it for Argus, Euler, and VGRNN, whose released implementations expose graph-scoring paths suitable for gradient-guided insertion. Its absence for Pikachu and Anomal-E is not evidence of robustness: the protocol simply cannot be instantiated against their released evaluation paths.

How much there is to evade varies sharply. Each attack run uses the detector's own preprocessing, default window, and calibrated threshold, not the shared contract of Table 8. On Campus none of the three detectors flags an attack edge at its own threshold, and on CI-2017 Euler flags none, so on those four pairs there is nothing to hide. VGRNN on CI-2017 flags 3 of 119. The eight pairs with a flagged target are those an evasion budget can be measured on.

Figure 5 shows that small budgets suffice on most of them. Two covering edges hide every flagged attack edge for Argus on LANL, across 352 targets, and for VGRNN on LANL across 371. VGRNN on CI-2017 also falls at two edges, over its three flagged targets. Euler on LANL reaches full evasion at $K = 5$ over 407 targets, Euler on OpTC at $K = 10$ over 1,059, and Argus on OpTC at $K = 20$ over 1,053. Six of the eight measurable pairs therefore lose every edge they caught within 20 inserted edges. Because the attack runs only on flagged edges, evasion means the covering edges changed a decision rather than exploiting an existing miss: clean benchmark accuracy is not sufficient evidence of robustness.

The two pairs that resist are also informative. Argus on CI-2017 converts none of its 145 flagged edges at any budget, and VGRNN on OpTC rises from 12.5% at $K = 2$ to 93.8% at $K = 50$ without reaching full evasion. Adversarial robustness is therefore dataset and system dependent and must be measured explicitly rather than inferred from architecture or clean-data performance.

The capability this assumes divides in two. The action is modest: a covering edge is an innocuous-looking flow originating from a host the adversary already controls, and two to twenty of them against millions of benign events sit far below any practical rate limit. The guidance is not: choosing which edges to insert needs gradients through the scoring path, so these budgets are a white-box upper bound on how cheap evasion can be. A weaker adversary needs a surrogate, and the black-box evasion Wang *et al.* [32] observe on LANL indicates that path is open. Both variants assume the telemetry faithfully records what crossed the network; an adversary who instead alters the log is outside our threat model and is the concern of tamper-evident auditing [20, 41].

Table 5: Streaming-readiness replay results in EG9.

Criterion	Result
Replay jobs	54
Detector–dataset pairs	18
Batch-only pairs	14
Near-real-time pairs	4
Real-time-ready pairs	0
Median buffering delay at 1×	40 min

Recommendations. Adversarial robustness should be a standard axis of GIDS evaluation rather than an optional extra. When a detector uses graph structure during scoring, the evaluation should include covering-edge tests and report the attack model, ASR across increasing budgets, and the minimum budget for full evasion. When an attack does not apply to a design, that non-applicability should be stated explicitly and not counted as evidence of robustness.

EG7: Lack of Systematic Component Attribution

5/5 systems

GIDS evaluations often report end-to-end gains, but do not systematically attribute those gains to individual pipeline components. Local ablations can show that a proposed model degrades when modified, but not whether the gain comes from graph reasoning, feature construction, temporal modeling, the objective, or dataset-specific signals.

Ablation is an attribution tool, not a checklist. Existing GIDS papers may include local variants or sensitivity tests, but these are scoped around one proposed architecture: they show whether a final recipe is fragile to modification, not which part of the pipeline is responsible for the result. Argus goes furthest, reporting a component ablation over its own architecture, but even that stays within one design. A graph neural network module may appear important because it interacts with a particular feature representation, objective, snapshot window, or dataset artifact, so the same end-to-end gain can support several incompatible explanations. We therefore factorize GIDS within GIDS-Eval into four interchangeable components, featurization, graph encoder, temporal encoder, and training objective, yielding 240 configurations, each run on Campus, CI-2017, LANL, and OpTC. The goal is not to tune a new detector but to test whether common GIDS design choices have

stable, attributable effects across datasets. Table 4 reports the best marginal option in each dimension after averaging over the others.

Graph structure provides dataset-dependent value. Removing the graph encoder, so that featurized inputs bypass message passing entirely, is the best marginal choice on LANL and stays within 0.005 AP of the best option on OpTC, while Campus and CI-2017 favor MPNN. Graph reasoning can help, but does not reliably explain gains across datasets, and its value must be demonstrated under controlled component variation.

The training target changes the winner. Reconstruction is best on Campus, CI-2017, and LANL, and link prediction is best on OpTC. The same model architecture can look effective under one target and weak under another, so without varying the target an evaluation cannot tell whether the gain comes from the encoder or from what the model is trained to predict.

There is no universal GIDS recipe. Best choices change across datasets, and the winner is an interaction among data representation, model structure, temporal context, and objective. Single-dataset ablations therefore do not generalize: a component that helps on one benchmark may add little, or even hurt, on another.

Recommendations. GIDS evaluations should report systematic component attribution, not just local ablations. A minimum framework should vary the input representation, graph encoder, temporal encoder, and training objective under the same protocol, and include non-graph neural network and non-temporal baselines, since these controls are necessary to justify claims about graph reasoning or temporal modeling. Without this structure, reported improvements may be real but their explanations remain unproven.

EG8: No Common Preprocessing Contract

5/5 systems

GIDS papers often claim to evaluate systems on the same datasets, but each system turns the raw logs into different model inputs. These pipelines change rows, features, graph semantics, labels, and splits, so cross-system comparisons do not always measure the same benchmark.

EG1 concerns one preprocessing parameter (the snapshot window) and EG4 isolates one feature-path switch in Argus. EG8 is a distinct problem: the absence of a common preprocessing contract defining how raw logs become model inputs. Here preprocessing is no neutral loading step: it defines the benchmark itself.

The same raw dataset can become different learning tasks: systems keep different rows, extract different features, normalize differently, collapse or preserve repeated edges, treat graphs as directed or undirected, and attach labels to different units, so the pipeline changes both the input and what the detector is asked to predict.

Prior cross-system comparisons show the problem. When Argus [35] reports Euler as a baseline, their native pipelines share no preprocessing contract, and even the systematic re-evaluation of Wang *et al.* [32] leaves each native pipeline intact, so numbers from different studies need not measure the same benchmark.

To measure the effect, we compare each system's native preprocessing against an aligned contract that fixes a shared event table, graph construction rule, label semantics, and split per dataset, and runs the original detector on that common input. This is not meant to tune the detectors. It asks whether the reported result is stable when the preprocessing layer is held fixed. Because EG1 shows the window moves AP on its own, both arms hold one window per dataset, common to every system: 5 min Campus, 50 min CI-2017, 1 h LANL and OpTC, so preprocessing is the only variable. A native score here is therefore measured at that window rather than at the system's own default: Argus on OpTC scores .334 at 1 h and .760 at the 6 min default that EG1 and EG4 sweep from. Figure 6 shows the six largest AP shifts.

The AP shifts are large enough to change the interpretation of several systems. VGRNN on LANL increases from .002 to .620 AP under the aligned contract. VGRNN on Campus increases from .015 to .445, and Euler on LANL increases from .035 to .423. The effect is not always positive: Anomal-E on CI-2017 drops from .875 to .605, and Euler on OpTC drops from .348 to .133. Preprocessing can therefore make a detector look stronger or weaker before the model architecture is even considered, and releasing only model code is insufficient: small differences in row filtering, feature extraction, timestamp handling, graph construction, or label assignment can move the evaluation target, so the preprocessing path must be precise enough to reconstruct the generated data byte-for-byte.

Recommendations. GIDS evaluations should define the full raw-to-model preprocessing contract. At a minimum, papers should report the input schema, feature extraction rules, graph construction rules, label unit, train/test split, and snapshot policy for each dataset. Authors should release executable preprocessing scripts and, when feasible, checksums or snapshots of the generated artifacts. Cross-system comparisons should report results under a shared aligned contract alongside each system's native pipeline.

EG9: Missing Real-Time Detection

5/5 systems

Existing GIDS are evaluated as batch graph detectors rather than streaming intrusion detectors. They first buffer events into snapshots, scenarios, or edge batches before scoring, so an attack cannot be flagged when the malicious event arrives.

EG5 asks whether a detector can finish processing a workload at scale, whereas EG9 asks whether a newly arriving event can be scored immediately. A detector can have acceptable batch runtime and still be unusable for real-time response if it must wait for a snapshot or scenario graph before producing any alert.

We replay the archived executions from EG1, EG3, and EG5 under a streaming arrival model. The replay does not retrain the detectors. It reuses each detector's runtime, input scale, batching unit, and offline detection result, and simulates whether the detector keeps up as events arrive at 1×, 2×, and 5× the native ingest rate. For each detector–dataset pair we record the buffering delay before scoring begins, the time to the first possible alert, the final processing lag, and whether the detector keeps up with the replay stream.

Throughput alone does not imply real-time detection. Table 5 summarizes the replay outcome. Across 54 replay jobs covering 18 detector–dataset pairs, all jobs complete and most systems keep up at the tested replay rates, yet none of the 18 pairs is real-time ready.

Table 6: ● GIDS-Lite candidates. Mean AP is averaged over the four validation splits, and the configuration with the highest mean is used unchanged on every dataset.

Endpoint state	Head	Dim.	Mean val. AP
Identity emb. + struct.	Reconstruction	296	**0.809**
Structural statistics	Reconstruction	40	0.791
Connectivity history	Reconstruction	40	0.763
Identity emb.	Reconstruction	264	0.632
Structural statistics	Contrastive	40	0.351
Connectivity history	Contrastive	40	0.349
Identity emb. + struct.	Contrastive	296	0.334
Identity emb.	Contrastive	264	0.283

Four short-window Campus pairs are only near real-time (Argus, Euler, and VGRNN use 150-second windows, while Pikachu uses a 300-second window). The remaining 14 are batch only. The median buffering delay at 1× replay is 40 minutes: alerts are delayed by the need to form a graph input even when the detector can process that input fast enough.

Argus on LANL illustrates the pattern: it keeps up even at 5× replay speed but requires a 1-hour snapshot before scoring can begin. The issue is not slow inference but that the unit of detection is larger than the unit of arrival: real networks produce events continuously, while current GIDS score only after a larger graph object has been formed.

Euler comes closest to a streaming design among the studied systems: it replicates the graph encoder across snapshot workers, with scalable near-real-time detection as an explicit goal [15]. Our replay bears out the throughput half of that design, since Euler keeps up at 5× the native ingest rate on all four datasets. It does not resolve the latency half: scoring still waits for a window to close, so Euler is near real-time only on the 150-second Campus window and batch only elsewhere.

This makes current GIDS closer to offline forensic analyzers than real-time intrusion detectors. They may still be useful for retrospective analysis where delay is acceptable, but their evaluations do not show timely response in a live network, where the question is not only whether an attack is detected but when the alert arrives.

Recommendations. GIDS evaluations should report the detection unit and arrival unit separately. Papers should state whether the system scores edges, nodes, snapshots, scenarios, or batches, and whether buffering is required before scoring can begin. When real-time detection is claimed, authors should report event-to-alert latency under a streaming arrival model, including snapshot window, buffer size, graph construction delay, and per-event scoring time. Batch-mode systems should be described as offline or delayed detectors unless they can produce alerts as events arrive.

5 GIDS-Lite vs. Existing GIDS

The analysis above motivates a simpler question: how much design complexity is actually needed for GIDS detection? We answer with GIDS-Lite, a deliberately simple baseline without graph or temporal encoders, derived from the gaps of §4 (§5.1). We then compare

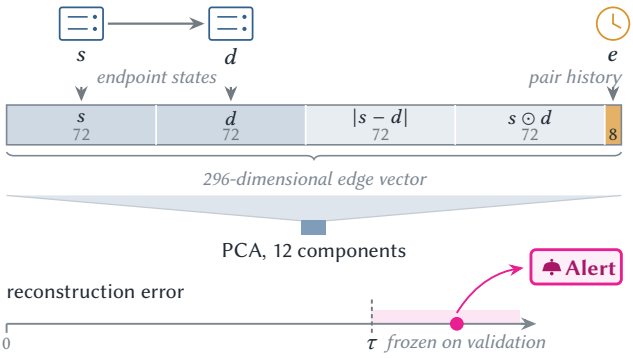


Figure 7: GIDS-Lite architecture. Endpoint states and pair-history scalars form a 296-dimensional edge vector, which a 12-component PCA reconstruction head scores. The same configuration is used on all four datasets.

GIDS-Lite with the five studied systems under the shared GIDS-Eval protocol on detection (§5.2) and computational cost (§5.3), isolate the effect of snapshot-window selection (§5.4), and analyze adversarial covering-edge robustness (§5.5).

5.1 GIDS-Lite Design

Purpose. GIDS-Lite is a minimal reference point for GIDS evaluation: it keeps the parts needed to predict from network events and drops the encoders whose value should be demonstrated rather than assumed. A strong GIDS-Lite result does not show graph reasoning is never useful. It shows that a more complex GIDS must provide clear gains over a simple detector under the same protocol.

Design principles. GIDS-Lite rests on three principles. It makes no evaluation-time choices that move the operating point after seeing test data, so thresholds are calibrated on a fixed chronological validation split. Its model is simple enough to serve as a control, so a direct predictor replaces the graph and temporal encoders. It keeps the same modular pipeline as the studied systems, so the shared GIDS-Eval featurization and thresholding protocol holds.

Model. GIDS-Lite contains no neural network, no message passing, and no recurrence. Its predictor is a PCA projection (12 components, fixed seed) fitted to the edge vectors of the earliest part of the trace, and each event is scored by its reconstruction error. Shallow is meant literally: zero hidden layers, and no graph or temporal encoder in the pipeline. Figure 7 summarizes it.

One configuration is used on all four datasets.[1] We fix it with a rule that never reads the test period: among the encoder-free candidates of Table 6, which pair four endpoint representations with a reconstruction or contrastive head, we take the one with the highest mean AP over the four validation splits. That rule selects the 64-dimensional identity embedding concatenated with eight structural statistics, scored by the reconstruction head, and we use

[1] Choosing a head per dataset would turn the control into a tuned competitor: a per-dataset contrastive head reaches 0.79 AP on LANL, above the entry a single configuration earns in Table 8. We report the single validation-selected configuration precisely so that no per-dataset choice enters the control, and accept the lower LANL score that follows. The resulting split sizes are given in Appendix E of the full version [40].

it unchanged on every dataset. GIDS-Lite is thus not tuned per dataset, which makes it a control, not a competitor: each studied GIDS keeps its published architecture, and GIDS-Lite keeps one.

Features. Each event is represented by a numeric edge vector $[\, s,\ d,\ |s-d|,\ s \odot d,\ e\,]$, z-scored on training data, where s and d are the states of the source and destination endpoints and e collects eight edge-level scalars describing pair history and recency (prior pair count, time since last seen, common neighbors, in/out imbalance). The endpoint state is a fixed 64-dimensional Gaussian embedding keyed to the node identity, concatenated with eight structural statistics (degree, reciprocity, neighbor statistics), giving a 296-dimensional edge vector on every dataset. GIDS-Lite does not consume the traffic-attribute edge features that EG4 shows to be collinear across attack and benign traffic. Its signal comes from connectivity and pair history, so it avoids those features.

Training and calibration. Training uses no attack labels. Each head is fit on the chronologically earliest snapshots of the trace, with a validation fraction of 0.33 and a cap of 50,000 training edges, and the detection threshold maximizes MCC over a 201-point grid on the chronological validation split, then stays frozen for testing. Both steps run under fixed random seeds, so training and calibration are deterministic end to end. The fitted model is correspondingly small: a 12×296 projection plus a 296-dimensional mean, about 3,800 stored values in total.

Scope and granularity. GIDS-Lite performs no link prediction. It assigns an anomaly score to every observed event, so decisions are per edge rather than per node or per graph. Because its features derive from snapshot structure, GIDS-Lite buffers the current snapshot before scoring and therefore shares the EG9 constraint with the studied systems. We claim per-event scoring, not real-time detection. Throughout, graph-based refers to the input representation rather than to a graph model, and GIDS-Lite is the encoder-free control within that setting. Appendix E of the full version [40] defines the edge unit and label assignment for each dataset.

Response to the evaluation gaps. GIDS-Lite is a control baseline, not a universal replacement for GIDS designs. Its purpose is to test whether the additional complexity in existing systems provides measurable value once the evaluation protocol is held fixed. Table 7 summarizes the resulting coverage: GIDS-Lite closes six of the nine gaps, EG2, EG3, and EG8 through the shared protocol, EG4 by avoiding the feature path, EG5 through the cost reported in §5.3, and EG7 through component attribution. Two remain open. Window sensitivity (EG1, §5.4) persists for GIDS-Lite, and snapshot buffering leaves EG9 unresolved. The covering-edge attack of EG6 cannot be instantiated against a detector without a graph-scoring path (§5.5). This is about attack applicability, not general robustness.

5.2 Detection Performance

Using the protocol of §3, we compare GIDS-Lite with the five studied systems under the same preprocessing, chronological-split and threshold-calibration rules, and reporting pipeline. Table 8 reports the end-to-end comparison on all four datasets.

By AP, one fixed encoder-free configuration is the strongest system on CI-2017 and OpTC, where GIDS-Lite reaches 0.980 and 0.993 without a graph encoder or a temporal encoder. The other two datasets go the other way: Pikachu reaches 0.915 AP on Campus against 0.322 for GIDS-Lite, and VGRNN reaches 0.620 on LANL against 0.510. The detection benefit of added complexity is therefore dataset dependent, and the split is not the one the simplicity argument would predict: complexity pays on the two datasets with the sparsest attack signal and is redundant on the two where a linear reconstruction of connectivity features already separates attack from benign. Where it does pay, the gain must still be weighed against the runtime cost quantified in §5.3.

Threshold-free and post-threshold results do not always move together, and no system is uniformly good on both. VGRNN has non-trivial AP on Campus and CI-2017 but zero CDR and zero precision after thresholding on both. GIDS-Lite shows the same gap from the other side: its thresholded operating point is attractive only on CI-2017 (97.6% precision at a 0.9% FPR). On Campus it exposes its test period's single campaign at 17.8% precision, on LANL its precision is 24.5% where Euler reaches 49.8%, and on OpTC it flags 59% of the benign events, a rate Pikachu avoids while scoring almost as well. Ranking quality alone does not identify the system an operator would deploy.

Reported detection rates should also be read against a realistic alert budget, since a high false-positive rate can make an otherwise accurate detector unusable in practice [1, 11, 29]. We therefore sweep the GIDS-Lite operating point and report recall at analyst-relevant false-positive rates (Figure 8). These curves reflect the

Table 7: Evaluation-gap status across the studied systems and ● GIDS-Lite. ✗ marks a gap that remains open for the pipeline, ✓ one that its protocol or design closes, and a dash a gap whose test cannot be instantiated for that design; §4 states the criterion applied for each gap.

System	EG1	EG2	EG3	EG4	EG5	EG6	EG7	EG8	EG9
● Argus	✗	✗	✓	✗	✗	✗	✗	✗	✗
● Euler	✗	✗	✓	✓	✗	✗	✗	✗	✗
● Pikachu	✗	✓	✗	✓	✗	–	✗	✗	✗
● VGRNN	✗	✓	✓	✓	✗	✗	✗	✗	✗
● Anomal-E	✓	✓	✓	✗	✗	–	✗	✗	✗
● GIDS-Lite	✗	✓	✓	✓	✓	–	✓	✓	✗

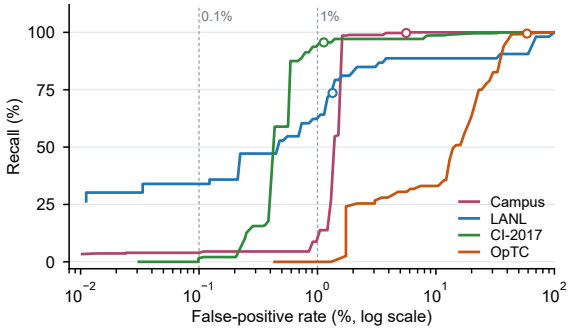


Figure 8: GIDS-Lite recall against the alert budget. Circles mark the operating point reported in Table 8, and dashed lines mark the 0.1% and 1% FPR budgets. No dataset retains its reported recall once the budget tightens to 0.1%.

Table 8: Detection results under the shared GIDS-Eval protocol. CDR counts the campaigns in each pipeline's test period, and the denominators of ● Anomal-E and ● GIDS-Lite differ from the shared-split campaign sets of the studied systems. ● Anomal-E is not applicable to LANL and OpTC, whose authentication and process events carry none of the flow attributes it consumes.

	Campus				LANL				CI-2017				OpTC			
System	**AP**	**CDR**	**Prec.**	**FPR**	**AP**	**CDR**	**Prec.**	**FPR**	**AP**	**CDR**	**Prec.**	**FPR**	**AP**	**CDR**	**Prec.**	**FPR**
● Argus	0.017	0.000	0.000	0.001	0.104	0.105	0.031	0.014	< .001	0.000	0.000	0.209	0.521	1.000	0.439	0.009
● Euler	0.004	1.000	0.007	1.000	0.423	0.649	0.498	0.024	< .001	0.000	0.000	0.042	0.133	0.714	0.149	0.048
● Pikachu	0.915	1.000	0.616	0.066	0.059	0.702	0.071	0.404	0.808	0.654	0.683	0.166	0.987	1.000	0.994	0.006
● VGRNN	0.445	0.000	0.000	< .001	0.620	0.912	0.122	0.366	0.280	0.000	0.000	< .001	0.935	1.000	0.742	0.231
● Anomal-E	0.048	1.000	0.048	1.000	–	–	–	–	0.605	1.000	0.658	0.192	–	–	–	–
● GIDS-Lite	0.322	1.000	0.178	0.056	0.510	0.529	0.245	0.013	0.980	0.714	0.976	0.009	0.993	1.000	0.986	0.587

single configuration of §5.1, which is fixed on validation data and never tuned per dataset. No dataset survives a strict budget. At a 1% FPR, recall is 94% on CI-2017 but 62% on LANL, 9% on Campus, and zero on OpTC. At 0.1% it is 34% on LANL and under 4% everywhere else. The reported operating points in Table 8 rest on alert budgets an analyst would question: roughly 1% on CI-2017 and LANL, 5% on Campus, and tens of percent on OpTC. OpTC also has a 97.6% attack base rate in its test period, which is what lifts every system's AP there and is the reason we read its column as the least informative of the four. These trade-offs, not single-threshold summaries, are what a deployment decision depends on.

Under a shared protocol, the return on added design complexity is inconsistent rather than absent. A single encoder-free configuration leads on two of four datasets, and on the other two the systems that win do so with different architectures, Pikachu on Campus and VGRNN on LANL, so no one design is vindicated either. The post-threshold columns add a second caveat: strong ranking must still be checked against campaign exposure and alert burden.

5.3 Runtime and Memory

We record end-to-end wall-clock time and peak resident memory for every detector under the shared contract. GIDS-Lite takes 107.2 seconds and 11.0 GB while Pikachu takes 17.1 hours and several systems exceed 27 GB. Figure 9 places every system on the same time–memory cost plane. As shown in §4, other valid preprocessing choices can raise observed peak memory to 102.2 GB.

The cost ranking does not match the detection ranking. Pikachu has the highest wall-clock time, but GIDS-Lite still achieves stronger AP on multiple datasets, and Euler and VGRNN incur substantially higher memory without a consistent detection advantage.

5.4 Snapshot-Window Sensitivity

EG1 shows that snapshot-window choice can substantially change the reported performance of existing GIDS. We test whether GIDS-Lite escapes it by holding the configuration fixed and varying only the snapshot-window size over the EG1 window grid, with the chronological split and validation-calibrated threshold of §5.2.

Table 11 of the full version [40] reports the full window sweep. The largest GIDS-Lite AP swing is 78.1% on LANL and the smallest 0.8% on CI-2017, a range comparable to the studied systems in EG1, and on LANL campaign exposure moves as well, by 23.5 percentage points of CDR. Removing the encoders does not remove the sensitivity: EG1 is a property of turning a continuous stream into graphs, not of the architecture that consumes them, and GIDS-Lite inherits it. The two datasets where GIDS-Lite ranks first are also the two where the window barely matters, which is a reason to report the setting rather than a reason to trust the ranking.

5.5 Attack Surface

EG6 shows that the cost of added architectural complexity is not limited to computation: it can also include an added attack surface. Recent graph attacks [26, 34, 36, 39] exploit exactly such interfaces. What they share is a graph-scoring path that can be influenced by changing graph structure, injecting graph elements, or probing graph-based model responses.

In our codebase, the released implementations of Argus, Euler, and VGRNN expose graph-scoring paths, so the EG6 covering-edge attack [36] applies directly to them. GIDS-Lite scores each edge from handcrafted snapshot features and a shallow predictor, with no multi-hop aggregation at inference, so the EG6 attack cannot be directly instantiated against it. The narrow conclusion is not that GIDS-Lite is adversarially robust in general, but that its simpler scoring path avoids this graph-propagation attack surface.

We therefore test the attack its design does admit. Because GIDS-Lite scores each edge from a standardized feature vector, an adversary who can shape those features faces a minimum-perturbation problem, which we solve exactly in the white-box setting: for every detected attack edge we compute the smallest perturbation, in standardized feature units, that pushes the edge below the frozen threshold. Because this attacker is exact rather than gradient-approximate, its budgets are an upper bound on what GIDS-Lite withstands. Starting from the operating points of Table 8, the median minimum perturbation is 12.4 on Campus, 3.3 on LANL, 1.8 on CI-2017, and 1.9 on OpTC, and the datasets where GIDS-Lite scores best are the ones an attacker breaks most cheaply: a budget of two standardized units removes 78 percentage points of recall on CI-2017 and 60 on OpTC, while Campus loses one point. Restricting the attacker to the pair-history scalars, with the endpoint identity block held fixed, changes the picture again: the median cost rises to 7.7 on CI-2017 and 4.7 on OpTC, and on Campus 98% of the detected attack edges

become unevadable at any budget. The attack therefore depends on being able to move the endpoint representation, not the pair-history scalars alone. GIDS-Lite is not adversarially robust, and it trades one attack surface for another.

More general evasion strategies [27, 28, 38], from traffic-feature manipulation to timing reshaping and semantics-preserving rewriting, remain relevant for encoder-light detectors and are outside our scope. The narrower claim is about attack applicability: added graph and temporal components introduce concrete attack surfaces that GIDS-Lite does not expose, so architectural complexity should be judged not only by clean-data accuracy and computational cost, but also by the attack surface it introduces.

6 Discussion & Future Work

The main lesson of this study is not that simpler models always win. Under a shared evaluation protocol, added graph and temporal complexity must be justified by clear net benefit, assessed not only through clean-data detection quality but also through computational cost, stability, and attack applicability.

Relation to Wang et al. [32] and Bilot et al. [2]. Our methodology adapts the staged PIDS decomposition of Bilot *et al.* to network flows, extending it to per-component attribution across systems so that symptoms Wang *et al.* could only observe end to end are traced back to individual stages (EG7). Our scope overlaps their black-box re-evaluation; Table 17 of the full version [40] summarizes which findings are new and which are not. The two data models also differ in ways that shape what can transfer: PIDS graphs inherit structure from kernel-event causality [12] and use symbolic features such as file paths [24], whereas GIDS must impose a snapshot window on a continuous flow stream, rely on statistical edge aggregates, and share no canonical preprocessing across papers. *Novel:* the cross-system snapshot-window sensitivity of EG1 (mean 38.3% relative AP swing: PIDS evaluations batch at fixed durations they never vary, and Wang *et al.* flag the window without quantifying it across systems), and the contract-aligned preprocessing attribution of EG8 (Euler on LANL moves from .035 to .423, where prior re-runs were per-paper and ad hoc). *Inverts:* flow attributes do not carry the separating signal that PIDS symbolic features do, and enabling Argus's edge-feature path on OpTC drops AP from 0.760 to 0.033 and CDR from 100.0% to 0.0% (EG4). Two covering edges fully evade the graph-scoring path on LANL, and six of the eight pairs with a flagged target fall within twenty inserted edges, because the GIDS graph is built from attacker-observable traffic (EG6), which also explains the black-box evasion Wang *et al.* observe on LANL. No single recipe wins across datasets, against the one-recipe conclusion for PIDS (EG7). *Confirms:* published GIDS results are fragile when re-run, as Wang *et al.* report, and a simple baseline is competitive once the evaluation is matched, as Bilot *et al.* found for PIDS. The threshold- and cost-hygiene concerns of EG2, EG3, and EG5 are shared with both studies and sharpened into per-pair measurements. *Adapts:* the deployment axis Bilot *et al.* address for PIDS with VELOX [2] becomes a replay measurement for GIDS, with none of 18 detector–dataset pairs real-time ready (EG9).

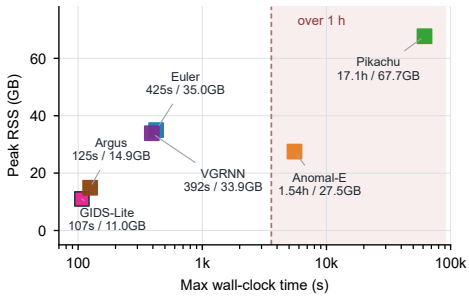


Figure 9: Computational cost under the EG8 shared preprocessing contract. Each point is the detector's largest wall-clock time and peak resident memory across the four datasets on the common input (two for ● Anomal-E); the ● GIDS-Lite point is its CI-2017 run in the EG7 grid. Table 10 of the full version [40] instead prices each system on its native pipeline.

Added Complexity Needs Clear Benefits. §5.2–§5.5 show that architectural complexity in current GIDS is not free under our protocol: added graph and temporal components do not consistently improve clean detection, but they increase runtime and memory cost, make the operating point sensitive to threshold calibration, and may expose additional attack surfaces. That does not make simple designs always preferable. It makes complexity a claim to be earned empirically rather than assumed.

Current Benchmarks Still Mix Key Factors. Each dataset bundles network setting, logging source, feature visibility, attack type, and temporal granularity, so when a system improves on one dataset but not another, the change may come from the model, the features, the attack behavior, or the representation. Future benchmarks should vary these factors deliberately while holding the rest fixed. Without it, claims about when graph structure helps stay unresolved.

Interpretability Still Matters in Practice. Detection quality is only part of operational usefulness. Analysts must also trace alerts to concrete evidence, which gets harder as scoring spreads across graph and temporal components. We do not evaluate explanation quality, so future GIDS evaluations should include alert traceability alongside detection metrics.

Robustness Must Be Evaluated Explicitly. The adversarial result in §5.5 adds another dimension. Under the current protocol, several existing GIDS expose graph-scoring paths that a covering-edge attack can exploit, while GIDS-Lite does not rely on that path. The conclusion is not that simpler detectors are universally robust, but that robustness cannot be assumed as a by-product of model complexity. Attack success and attack applicability should be reported together. Future work should study detector-agnostic attack models, defenses for both graph-based and encoder-light detectors, and robustness metrics that reflect the NIDS threat model.

Shared Evaluation Reduces Repeated Work. Progress in GIDS depends on evaluation infrastructure as much as on models. When papers use incompatible splits, preprocessing pipelines, or reporting rules, the cost of rigorous comparison is paid many times over without producing cumulative evidence. Shared preprocessing artifacts,

canonical splits, common metric suites, and reusable harnesses would make results comparable. That infrastructure is part of the research agenda, not auxiliary to it.

7 Conclusion

We presented GIDS-Eval, an evaluation framework that attributes GIDS performance to pipeline stages rather than whole systems and measures nine evaluation gaps current studies leave unexamined. Under a matched protocol, added graph and temporal complexity does not reliably buy detection quality, yet it costs runtime and calibration stability and adds attack surface, and our encoder-free control, GIDS-Lite, competes on half the datasets at a fraction of the runtime. Complexity in GIDS should be introduced only when it earns its cost under a shared protocol.

Acknowledgments

We sincerely thank our shepherd and the anonymous reviewers for their insightful feedback on this work. This material is based upon work supported by the National Science Foundation (NSF) under grant numbers #2339483, #2530655, and #2622986.

References

[1] D. Arp, E. Quiring, F. Pendlebury, A. Warnecke, F. Pierazzi, C. Wressnegger, L. Cavallaro, and K. Rieck. Dos and don'ts of machine learning in computer security. In *USENIX Security Symposium*, 2022.

[2] T. Bilot, B. Jiang, Z. Li, N. El Madhoun, K. Al Agha, A. Zouaoui, and T. Pasquier. Sometimes simpler is better: A comprehensive analysis of state-of-the-art provenance-based intrusion detection systems. In *USENIX Security Symposium*, 2025.

[3] E. Caville, W. W. Lo, S. Layeghy, and M. Portmann. Anomal-E: A self-supervised network intrusion detection system based on graph neural networks. *Knowledge-Based Systems*, 258:110030, 2022.

[4] K. Cho, B. Van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, 2014.

[5] DARPA. Transparent computing engagement 3 data release. https://github.com/darpa-i2o/Transparent-Computing, 2020.

[6] DARPA. Operationally transparent cyber (OpTC) dataset. https://github.com/FiveDirections/OpTC-data, 2020.

[7] M. Fey and J. E. Lenssen. PyTorch Geometric. https://pytorch-geometric.readthedocs.io/, 2019.

[8] R. Flood, G. Engelen, D. Aspinall, and L. Desmet. Bad design smells in benchmark NIDS datasets. In *IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 658–675, 2024.

[9] E. Hajiramezanali, A. Hasanzadeh, K. Narayanan, N. Duffield, M. Zhou, and X. Qian. Variational graph recurrent neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[10] W. Hamilton, Z. Ying, and J. Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017.

[11] W. U. Hassan, S. Guo, D. Li, Z. Chen, K. Jee, Z. Li, and A. Bates. NoDoze: Combatting threat alert fatigue with automated provenance triage. In *Network and Distributed System Security (NDSS)*, 2019.

[12] M. A. Inam, Y. Chen, A. Goyal, J. Liu, J. Mink, N. Michael, S. Gaur, A. Bates, and W. U. Hassan. Sok: History is a vast early warning system: Auditing the provenance of system intrusions. In *IEEE Symposium on Security and Privacy (S&P)*, 2023.

[13] A. D. Kent. Comprehensive, multi-source cyber-security events data set. Los Alamos National Laboratory, https://csr.lanl.gov/data/cyber1/, 2015.

[14] J. Khoury, Đ. Klisura, H. Zanddizari, G. D. L. T. Parra, P. Najafirad, and E. Bou-Harb. Jbeil: Temporal graph-based inductive learning to infer lateral movement in evolving enterprise networks. In *IEEE Symposium on Security and Privacy (S&P)*, pages 3644–3660, 2024.

[15] I. J. King and H. H. Huang. Euler: Detecting network lateral movement via scalable temporal graph link prediction. In *Network and Distributed System Security Symposium*, 2022.

[16] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2017.

[17] L. Liu, G. Engelen, T. Lynar, D. Essam, and W. Joosen. Error prevalence in NIDS datasets: A case study on CIC-IDS-2017 and CSE-CIC-IDS-2018. In *IEEE Conference on Communications and Network Security (CNS)*, pages 254–262, 2022.

[18] Q. Liu, K. Bao, W. U. Hassan, and V. Hagenmeyer. HADES: Detecting and investigating active directory attacks via whole network provenance analytics. *IEEE Transactions on Dependable and Secure Computing*, 23(1):936–953, 2026.

[19] W. W. Lo, S. Layeghy, M. Sarhan, M. Gallagher, and M. Portmann. E-GraphSAGE: A graph neural network based intrusion detection system for IoT. In *IEEE/IFIP Network Operations and Management Symposium (NOMS)*. IEEE, 2022.

[20] R. Paccagnella, P. Datta, W. U. Hassan, A. Bates, C. W. Fletcher, A. Miller, and D. Tian. Custos: Practical Tamper-Evident Auditing of Operating Systems Using Trusted Execution. In *NDSS*, 2020.

[21] R. Paudel and H. H. Huang. PIKACHU: Temporal walk based dynamic graph embedding for network anomaly detection. In *IEEE/IFIP Network Operations and Management Symposium (NOMS)*, 2022.

[22] F. Pendlebury, F. Pierazzi, R. Jordaney, J. Kinder, and L. Cavallaro. TESSERACT: Eliminating experimental bias in malware classification across space and time. In *USENIX Security Symposium*, 2019.

[23] J. Piet, A. Sharma, V. Paxson, and D. Wagner. Network detection of interactive SSH impostors using deep learning. In *USENIX Security Symposium*, pages 4283–4300, 2023.

[24] M. U. Rehman, H. Ahmadi, and W. U. Hassan. FLASH: A comprehensive approach to intrusion detection via provenance graph representation learning. In *IEEE Symposium on Security and Privacy (S&P)*, 2024.

[25] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani. CIC-IDS-2017 dataset. https://www.unb.ca/cic/datasets/ids-2017.html, 2017.

[26] K. Sharma, R. Trivedi, R. Sridhar, and S. Kumar. Temporal dynamics-aware adversarial attacks on discrete-time dynamic graph models. In *ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 2023–2035, 2023.

[27] Y. Sharon, D. Berend, Y. Liu, A. Shabtai, and Y. Elovici. TANTRA: Timing-based adversarial network traffic reshaping attack. *IEEE Transactions on Information Forensics and Security*, 17:3225–3237, 2022.

[28] M. Shoaib, H. S. Muthusamy, T. Alkhatib, and W. U. Hassan. Catch me if you can: Detector-resistant evasion via semantics-preserving command re-realization. In *IEEE Symposium on Security and Privacy (S&P)*, 2026.

[29] R. Sommer and V. Paxson. Outside the closed world: On using machine learning for network intrusion detection. In *IEEE Symposium on Security and Privacy*, 2010.

[30] R. Uetz, M. Herzog, L. Hackländer, S. Schwarz, and M. Henze. You cannot escape me: Detecting evasions of SIEM rules in enterprise networks. In *USENIX Security Symposium*, pages 5179–5196, 2024.

[31] A. Venturi, M. Ferrari, M. Marchetti, and M. Colajanni. ARGANIDS: A novel network intrusion detection system based on adversarially regularized graph autoencoder. In *ACM/SIGAPP Symposium on Applied Computing (SAC)*, pages 1540–1548, 2023.

[32] C. Wang, P. Zheng, J. Gui, C. Hua, and W. U. Hassan. R+R: From claims to crashes: A systematic re-evaluation of graph-based network intrusion detection systems. In *Annual Computer Security Applications Conference (ACSAC)*, 2025.

[33] F. Wei, H. Li, Z. Zhao, and H. Hu. xNIDS: Explaining deep learning-based network intrusion detection systems for active intrusion responses. In *USENIX Security Symposium*, pages 4337–4354, 2023.

[34] J. Xu, Y. Sun, X. Jiang, Y. Wang, C. Wang, J. Lu, and Y. Yang. Blindfolded attackers still threatening: Strict black-box adversarial attacks on graphs. *AAAI Conference on Artificial Intelligence*, 36(4):4299–4307, 2022.

[35] J. Xu, X. Shu, and Z. Li. Understanding and bridging the gap between unsupervised network representation learning and security analytics. In *IEEE Symposium on Security and Privacy (S&P)*, 2024.

[36] X. Xu, Q. Hao, Z. Yang, B. Li, D. Liebovitz, G. Wang, and C. A. Gunter. How to cover up anomalous accesses to electronic health records. In *USENIX Security Symposium*, pages 229–246, Anaheim, CA, 2023. USENIX Association.

[37] W. Yu, W. Cheng, C. C. Aggarwal, K. Zhang, H. Chen, and W. Wang. NetWalk: A flexible deep embedding approach for anomaly detection in dynamic networks. In *ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 2672–2681, 2018.

[38] C. Zhang, X. Costa-Pérez, and P. Patras. Adversarial attacks against deep learning-based network intrusion detection systems and defense mechanisms. *IEEE/ACM Transactions on Networking*, 30(3):1294–1311, 2022.

[39] M. Zhao and J. Zhang. Highly imperceptible black-box graph injection attacks with reinforcement learning. *AAAI Conference on Artificial Intelligence*, 39(12): 13357–13364, 2025.

[40] R. Zhao and W. U. Hassan. A first-principles evaluation of graph-based network intrusion detection systems. arXiv preprint arXiv:2609.12263, 2026. URL https://arxiv.org/abs/2609.12263.

[41] R. Zhao, M. Shoaib, V. T. Hoang, and W. U. Hassan. Rethinking tamper-evident logging: A high-performance, co-designed auditing system. In *ACM Conference on Computer and Communications Security (CCS)*, 2025.