Learning Transferable Representations from Operating System Entities via Provenance Graph Distillation

Tristan Bilot[1], **Xueyuan Han**[2], **Thomas Pasquier**[1,3]
[1]University of British Columbia, [2]AppGate, [3]University of Oxford
tbilot@cs.ubc.ca, michael.vanbastelaer@appgate.com, thomas.pasquier@cs.ox.ac.uk

Abstract

Provenance graphs capture causal interactions among operating-system entities, underpinning modern endpoint intrusion detection. Existing graph-based detectors use generic text encoders trained on small datasets to embed entities solely from their labels (e.g., file paths and process command lines). Embeddings are thus independent of the graph, missing behavioral semantics encoded in graph neighborhoods. However, much like a natural language that carries linguistic structures that can be learned once and transferred broadly, OS entities such as system binaries, configuration files, and common services engage in recurring types of activity with similar inter-entity relationships across executions. These behavioral regularities can be learned offline from provenance graphs and approximated at inference time from entity labels alone. Based on this insight, we introduce SPIDER (**S**ystem **P**rovenance-**I**nformed **D**istilled **E**ntity **R**epresentations), a text-only entity encoder distilled from provenance graphs. SPIDER uses a graph-based teacher that models *who* an entity interacts with via graph attention and *how* it interacts with them through behavioral signatures. These representations are distilled into a transformer-based student that maps raw entity text to embeddings in a single forward pass, providing pretrained behavioral priors that downstream detectors can use for real-time inference without requiring graph access. With only 5.2M parameters, SPIDER can be deployed directly on endpoints and optionally fine-tuned for specific detection tasks. We show that a single checkpoint improves detection performance as a drop-in replacement for four recent intrusion detection systems across Linux, FreeBSD, Windows, and Android, surpassing both GNN- and LLM-based baselines. Code is available at `https://github.com/ubc-provenance/PIDSMaker/tree/spider`.

Note: This is a preprint version of the paper accepted at the The 40th Annual Conference on Neural Information Processing Systems (NeurIPS'26) [Bilot et al., 2026a].

1 Introduction

Graph learning problems focus on improving feature embeddings to better represent nodes, edges, or the overall graph. Depending on the nature of the task at hand and the domain in which it operates, different aspects of graph data (e.g., degrees, labels, and temporality) contribute to such representations. In this work, we study *node representation learning* in the *endpoint intrusion detection* setting, where the quality of node embeddings plays a central role in the downstream classification task.

A graph in this setting represents an operating system's activity, where nodes and edges correspond respectively to OS entities (e.g., files, processes, and network sockets) and activities (e.g.,

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

system calls). Prior work [King and Chen, 2003, Pasquier et al., 2017, 2018] has shown that such a graph provides greater visibility into cyberattacks than traditional audit logs. Attack activity manifests as anomalies in the graph, and the goal of the learning algorithm is to identify those anomalies. Figure 1 shows a small, simplified snippet of an OS activity graph. To match the terminology used in the security community, we henceforth refer to such a graph as a *provenance graph*.

We see that nodes and edges in a provenance graph are typed; nodes also carry *textual labels* that identify unique OS entities. The graph illustrates an attack in which a compromised Firefox browser receives a drive-by exploit from a malicious IP and writes a payload to disk; the payload is then executed and communicates with another malicious IP. In a real OS workload, this attack would present itself as a subgraph embedded in a much larger provenance graph that dynamically grows as the OS runs.

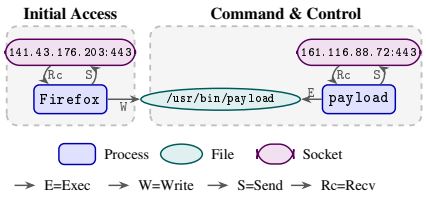

Figure 1: **Simplified provenance graph.** Example of a 2-stage attack.

Endpoint intrusion detection classifies nodes like in Figure 1 as anomalies, by training its model on *benign* graph data to learn statistically how OS entities interact with each other under no attack influence [Zipperle et al., 2022]. Learning node representation typically involves (1) using word embedding of an entity's textual label (e.g., a file path, a command line, or a socket address) as the initial node vector, which captures its lexical identity, and (2) subsequent message-passing-style iterations over its neighborhood. The goal of the latter is to infer an entity's behavior from the graph structure, so that entities with similar functional roles are close to each other in the embedding space, even if their labels are textually distinct. This has historically motivated prior work to adopt various graph learning architectures, ranging from standard GNNs [Li et al., 2024, Jia et al., 2024, Goyal et al., 2024, Rehman et al., 2024] to tailored temporal graph networks with attention [Cheng et al., 2023, Jiang et al., 2025]. Prior studies tightly couple learning node-embedding space with the intrusion detection objective, focusing on optimizing structure-aware entity encoding only to minimize detection-specific loss. However, doing so has two major shortcomings, which we address in this work by proposing a completely different learning paradigm that *decouples representation learning from intrusion detection*.

First, node embedding is learned always "from scratch" through entities' textual labels. This is not by choice, but stems from lack of meaningful representation that is independent from specific downstream detection mechanisms. However, just like a natural language carrying linguistic structures that can be learned once and transferred broadly, entities such as system binaries, configuration files, and common services and network sockets tend to participate in recurring types of activity with similar inter-entity relationships across executions. A general-purpose encoder can capture these stable and predictable interaction patterns from diverse workloads as *behavioral priors*. This not only reduces wasteful and expensive recomputation prevalent in existing work, but more importantly, encodes distributional regularities of OS activity across different environments. Such knowledge is missing in previous approaches as a result of their dataset-specific training. We show in this work that it improves downstream intrusion detection performance regardless of the design. Second, inference-time classification requires costly graph traversal to inject behavioral semantics into node embedding, creating a computational bottleneck that is infeasible to sustain at enterprise scale across many hosts for real-time intrusion detection [Dong et al., 2023]. This is not by choice either, because without graph convolution, a label-based lexical identity is only a poor proxy for an entity's behavior. Consider, for example, binary files such as `/usr/bin/man` and `/usr/bin/make` that have textually similar paths but entirely different interaction patterns. We show in this work that graph exploration can be optional when node embedding itself entails such patterns.

We propose SPIDER, a text-only entity encoder trained via *offline provenance graph distillation*. To train this encoder, a graph-aware teacher uses provenance graphs to understand what entities a node interacts with and behavioral signatures to summarize how these interactions take place, combining both to learn behavioral priors from graph context. A lightweight, transformer-based student is simultaneously trained to approximate entity representation from textual labels alone. A downstream application can use SPIDER to train a detection model with a graph-free classification head and run a single forward pass with no graph storage or iterative message passing for inference, taking full advantage of its behavior-aware entity features. Alternatively, SPIDER can replace the text backbone used by prior work, improving the performance of any graph-based detectors by providing better

initial entity embeddings without increasing their inference-time overhead. Overall, SPIDER takes a first step toward pretrained, transferable representations for real-time, OS-level endpoint security.

2 Problem Formulation

A provenance graph is a directed, labeled, temporal multi-graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. Each node $v \in \mathcal{V}$ has a type $\texttt{type}(v) \in \{\text{PROC}, \text{FILE}, \text{SOCK}\}$ and a textual label $\ell(v)$. Each edge $e = (u, v, \tau, t) \in \mathcal{E}$ connects a source node u to a destination node v with event type $\tau \in \mathcal{T}$ and timestamp $t \in \mathbb{R}_{\geq 0}$. The event type vocabulary contains ten system-call categories: $\mathcal{T} = \{\texttt{CLONE}, \texttt{CONNECT}, \texttt{EXECUTE}, \texttt{OPEN}, \texttt{READ}, \texttt{RECVFROM}, \texttt{RECVMSG}, \texttt{SENDMSG}, \texttt{SENDTO}, \texttt{WRITE}\}$

Valid edges are constrained to specific $(\texttt{type}(u), \tau, \texttt{type}(v))$ triplets, *e.g.*(PROC, WRITE, FILE); section K shows the complete list. Notably, our event type vocabulary is OS-agnostic: a process forking a child or writing to a file maps to the same token on Linux, Windows, FreeBSD, or Android, allowing a single pretrained model to transfer across platforms.

Objective. The goal is to learn a function $f : \mathcal{V} \to \mathbb{R}^d$ mapping each entity v to a dense vector $\mathbf{e}_v = f(v) \in \mathbb{R}^d$ such that (i) $\mathbf{e}_v$ captures the entity's expected interaction patterns while remaining computable from text alone, (ii) f is learned without supervision on any anomalies, and (iii) f depends only on $\texttt{type}(v)$ and its textual label $\ell(v)$, *not* on the runtime graph neighborhood of v. The last condition ensures that computing $\mathbf{e}_v$ is a single operation independent of the graph.

Anomaly-based intrusion detection. Given entity embeddings $\{\mathbf{e}_v\}_{v \in \mathcal{V}}$ learned from benign activity, downstream detectors flag intrusions as deviations from this baseline by scoring each edge $e = (u, v, \tau, t) \in \mathcal{E}$ via a function $g(\mathbf{e}_u, \mathbf{e}_v, \tau) \to \mathbb{R}$ that assigns high values to anomalous interactions. When an edge's score exceeds a threshold, its corresponding OS activity is flagged as a potential attack. SPIDER is agnostic to the choice of g: it provides $\mathbf{e}_v$ as a drop-in backbone, and a downstream detector can optionally incorporate graph-neighborhood features for additional context, though we favor graph-free computations for real-time deployment.

3 Method

SPIDER is built on a teacher-student knowledge distillation [Hinton et al., 2015] paradigm. Offline training uses a teacher encoder with access to the provenance graph to produce rich entity representations; a transformer student learns to reproduce them from entity labels alone. Deployment retains only the student: a single forward pass produces an entity embedding with no graph access.

3.1 System Entity Tokenization

Before entering the transformer encoder, raw entity labels are processed by a two-phase tokenizer tailored to system entity attributes. Table 8 (section E) shows some tokenization examples.

Phase 1: domain-specific pre-tokenization. System entity strings carry context-dependent semantics that a generic tokenizer would conflate (e.g., 22 as a port identifies SSH, but it is an arbitrary name in a text file path). This phase makes such context explicit before subword encoding. For *file paths*, recognized components are mapped to dedicated tokens (e.g., `/usr/lib` → `[/USR][/LIB]`), with separate tokens for extensions, registry hives, and separators. Camel-case and compound names are split. Semantically unstable segments (e.g., hashes, GUIDs) are collapsed to abstract tokens (`[HASH]`, `[GUID]`) to prevent overfitting to deployment-specific artifacts. Usernames are anonymized to `[USER]`, whereas root becomes `[ROOT]`. For *process command lines*, extensions (*e.g.*`.sh`, `.exe`) and flags (*e.g.*`-g`, `--force`) receive dedicated tokens, and embedded file paths follow the rules above. For *socket addresses*, well-known ports are mapped to named tokens (*e.g.*`[PORT_SSH]`, `[PORT_HTTPS]`), and high-range or ephemeral ports are grouped under a common token. IP addresses are replaced by `[PRIVATE_IP]` or `[PUBLIC_IP]` during pre-training to avoid spurious corpus biases; during continued pre-training on a target dataset, the original octets are reintroduced alongside these tokens (e.g., `[PRIVATE_IP] 192 168 1 1 [PORT_SSH]`), so the model can adapt to deployment-specific patterns while retaining bias-free representations as a prior. All tokenized entities are prefixed with a type token (`[PROC]`, `[FILE]`, `[SOCK]`).

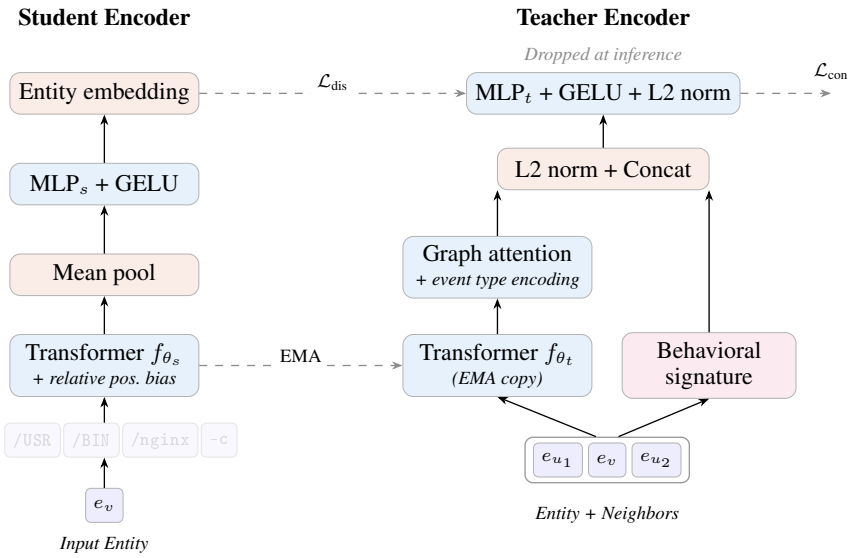

Figure 2: **SPIDER architecture.** A graph-aware teacher distills behavioral context from provenance neighborhoods into a text-only student; only the student is retained at inference.

Phase 2: BPE subword encoding. Remaining raw words are split by a domain-trained byte-pair encoding (BPE) vocabulary that guarantees no out-of-vocabulary tokens, producing sequences that mix reusable domain tokens with fine-grained subword pieces.

3.2 Teacher Architecture and Entity Profiling

The goal of the teacher (Figure 2, right) is to produce an embedding that captures an entity's *functional role*, not just its textual identity but the behavior it exhibits in the system. To achieve this, the teacher builds a behavioral profile from the provenance graph using two complementary signals. The graph-structural embedding captures *who* the entity interacts with by encoding its neighbors through message passing, while the behavioral signature captures *how* it interacts by summarizing the types and directions of events it participates in. These interaction patterns are strongly correlated with the entity's textual identity, enabling the student to approximate them from the text alone.

Graph-structural context. The first signal comes from message-passing over a subgraph of v's neighborhood $\mathcal{N}(v)$. Noisy edges, e.g., accesses to shared libraries or pseudo-filesystem paths (`/proc`, `/sys`), are first filtered out by heuristics. From the remaining edges, we draw a random sample of neighbors with replacement, stratified by edge type for diversity, and re-draw at each epoch so the model sees varying neighborhood views rather than overfitting a single fixed subgraph. We use a layer of graph attention [Shi et al., 2021] that natively supports edge features, so that the attention weight of each message is conditioned on the event type $\tau_{(u,v)}$ of the edge: the model learns, for instance, that a `write` edge from a process to a file carries different semantics than a `read` edge between the same pair.

Learnable edge-type embeddings are injected into the attention computation. The attention layer aggregates every neighbor u's feature from a teacher encoder f_{θ_t} (Section 3.3), which encodes the nodes' token sequences $t(u)$.

$$\mathbf{h}_v^{\text{gnn}} = \text{GraphAttn}\big(\{f_{\theta_t}(t(u))\}_{u\in\mathcal{N}(v)\cup\{v\}},\ \{\tau_{(u,v)}\}_{u\in\mathcal{N}(v)}\big)\,. \tag{1}$$

Behavioral signatures. The second signal is a *behavioral signature* $\mathbf{s}_v \in \{0,1\}^{\mathcal{B}}$, a compact summary of interaction patterns in the one-hop neighborhood $\mathcal{N}(v)$. Each neighbor interaction is encoded as a triplet $(\text{dir}, \tau, c(v))$ combining (i) the edge direction $\text{dir} \in \{\text{IN}, \text{OUT}\}$, indicating whether the event is *received* or *emitted* by v (because e.g., being read vs. initiating a read carries different semantics); (ii) the event type $\tau \in \mathcal{T}$; and (iii) its semantic class $c(v)$.

$c(v)$ is derived from the mapping $\{\texttt{type}(v), \ell(v)\} \to c(v)$ where $\ell(v)$ is a process name, file path prefix, file extension, or well-known port number. For example, $\{\texttt{PROC}, \texttt{nginx}\} \to \texttt{webserver}$,

$\{\texttt{FILE}, \texttt{/etc/*}\} \rightarrow \texttt{config\_file}$, and $\{\texttt{SOCK}, 22\} \rightarrow \texttt{ssh\_socket}$ (see section J). The intuition is that entities sharing the same semantic class are behaviorally similar regardless of host-specific naming differences: two `webserver` processes, however their binaries are named or installed, should be embedded closely.

As an example, the behavioral signature of a browser process that receives a creation event from a process yields the signature entry `IN:CLONE:browser`, while a process that read a temporary file produces `OUT:READ:tmp`. The set of all distinct triplets observed across the dataset forms a global signature vocabulary $\mathcal{B}$ and $\mathbf{s}_v$ is a binary indicator over this vocabulary. For each entity v, we collect all distinct triplets observed across its one-hop edges and merge them into a single multi-hot vector $\mathbf{s}_v$ over $\mathcal{B}$.

Combining the two signals. The two signals are complementary by design: the GNN embedding $\mathbf{h}_v^{\text{gnn}}$ carries rich but potentially distribution-specific information tied to the exact neighbors seen during training, while the behavioral signature $\mathbf{s}_v$ encodes a succinct summary of interaction *types* that generalizes across deployments. We concatenate them after independent L2 normalization to ensure that neither dominates the joint representation. A two-layer MLP with GELU activation and a final L2 normalization produces the teacher projection, encoding the entity's full behavioral profile:

$$\tilde{\mathbf{h}}_v = [\text{L2Norm}(\mathbf{h}_v^{\text{gnn}}) \parallel \text{L2Norm}(\mathbf{s}_v)], \tag{2}$$

$$\mathbf{z}_v^t = \text{L2Norm}\Big(\text{MLP}_t\Big(\tilde{\mathbf{h}}_v\Big)\Big). \tag{3}$$

Supervised contrastive loss. To structure the teacher's embedding space, we train with a supervised contrastive (SupCon) loss [Khosla et al., 2020] that pulls entities of the same semantic class together while pushing apart entities from different classes. Over a batch B of teacher projections $\mathbf{z}_v^t$:

$$\mathcal{L}_{\text{con}} = \frac{-1}{|B|} \sum_{v \in B} \frac{1}{|P(v)|} \sum_{p \in P(v)} \log \frac{\exp(\mathbf{z}_v^t \cdot \mathbf{z}_p^t / T)}{\sum_{a \in B \setminus \{v\}} \exp(\mathbf{z}_v^t \cdot \mathbf{z}_a^t / T)}, \tag{4}$$

where $P(v) = \{p \in B \mid c(p) = c(v), p \neq v\}$ is the set of same-class positives and T is the temperature. Batches are constructed with $P \times K$ sampling: P classes are drawn uniformly with $K = 4$ entities from each, guaranteeing in-batch positives for every anchor. Class imbalance is handled by oversampling small classes and capping the large ones.

3.3 Student Architecture and Distillation

The student encoder (Figure 2, left) operates on entity labels alone and is the only component retained at inference. Its goal is to approximate the teacher's projections $\mathbf{z}_v^t$ without access to the provenance graph.

Student encoder. The core component is the transformer f_{θ_s} with relative position bias [Raffel et al., 2020], which we prefer over absolute positional encodings since entity labels lack a meaningful global token order: salient tokens in `/usr/bin/nginx` or `192.168.1.1` are defined by their position within a path segment or address group, not by absolute sequence index. Given the token sequence $t(v)$ of an entity v, the encoder output is mean-pooled over non-padding positions and projected through a two-layer MLP with GELU activation:

$$\mathbf{e}_v^s = \text{MLP}_s(\text{mean-pool}(f_{\theta_s}(t(v)))) \in \mathbb{R}^d. \tag{5}$$

EMA teacher encoder. Following Caron et al. [2021], the teacher encoder f_{θ_t} is built as an exponential moving average (EMA) of the student: $\theta_t \leftarrow \lambda\, \theta_t + (1 - \lambda)\, \theta_s$ with $\lambda = 0.999$. During training, f_{θ_t} encodes all nodes in the local subgraph to provide stable node features for the teacher encoder. The teacher receives no gradients, preventing it from chasing a rapidly-shifting target [He et al., 2020, Grill et al., 2020].

Distillation loss. The student is trained to match the teacher's projections (treated as a fixed target via stop-gradient) using scaled cosine error (SCE) [Hou et al., 2022]:

$$\mathcal{L}_{\text{dis}} = \frac{1}{|B|} \sum_{v \in B} \left(1 - \frac{\mathbf{e}_v^s \cdot \text{sg}(\mathbf{z}_v^t)}{\|\mathbf{e}_v^s\|_2 \, \|\text{sg}(\mathbf{z}_v^t)\|_2}\right)^2. \tag{6}$$

SCE is preferred over MSE because it measures directional alignment only, discarding magnitude differences irrelevant for cosine-similarity tasks, while the exponent focuses training on harder, poorly-aligned pairs. Through distillation, the student learns to approximate from text alone the behavioral structure learned by the teacher, producing embeddings where textual similarity correlates with behavioral similarity. The two terms are combined as $\mathcal{L} = \mathcal{L}_{\text{con}} + \mathcal{L}_{\text{dis}}$.

Inference. At test time, only the student encoder is used. Embeddings for frequent entities can be precomputed and cached, so scoring a new event reduces to two hash-table lookups plus a lightweight detection head forward pass.

4 Experimental Setup

4.1 Datasets

Pretraining data. We pretrain SPIDER on a raw system-provenance corpus of 42.3M entities, which we built from benign activity collected inside monitored environments spanning diverse workload categories (web browsing, daemons, web servers, package management) across Linux, FreeBSD, Windows, and Android. Raw provenance data is highly redundant; we apply filtering, neighborhood deduplication, and post-tokenization entity merging to retain a subset of high-quality training examples. Full dataset statistics and preprocessing details are in section F. The corpus is disjoint from all evaluation data below (no shared hosts, organizations, capture frameworks, or time periods), so frozen results measure cross-environment transfer.

Evaluation data.

We evaluate on the benchmarks commonly used by the intrusion detection community (Table 1): DARPA TC Engagement E3 [DARPA, 2018], E5 [DARPA, 2019] and OpTC [DARPA, 2019], spanning FreeBSD, Linux, Android, and Windows. All contain real background activity with multi-stage attacks and node-level ground truth [Jiang et al., 2025].

Table 1: **Evaluation dataset statistics.** N_{atk} is the number of attacks and $|\mathcal{V}_{\text{atk}}|$ the number of attack nodes.

Dataset	Abbr.	OS	$\lvert\mathcal{V}\rvert$	$\lvert\mathcal{E}\rvert$	N_{atk}	$\lvert\mathcal{V}_{\text{atk}}\rvert$
CADETS-E3	CD-E3	FreeBSD	643.5K	5.9M	3	75
THEIA-E3	TH-E3	Linux	1.1M	9.4M	2	119
THEIA-E5	TH-E5	Linux	1.6M	31.0M	1	70
CLEARSCOPE-E5	CS-E5	Android	335.5K	36.6M	2	34
OpTC-051	OpTC	Windows	3.8M	15.4M	1	749

4.2 Implementation Details

Released model. Knowledge distillation naturally yields a lightweight deployment artifact: once training is complete, the teacher is discarded and only the compact student transformer is retained. We release a pretrained checkpoint with 5.2M parameters (d=256, 4 attention layers, 4 heads). This lightweight footprint enables SPIDER to be deployed directly on endpoints, where entity embeddings are computed in a single forward pass, cached for frequent entities, and passed to a downstream detector for real-time detection. SPIDER is integrated natively into PIDSMaker [UBC Provenance, 2025], an open-source framework to build and evaluate provenance-based intrusion detectors.

Baselines. For fair and consistent evaluation of different detector baselines, we use PIDSMaker [UBC Provenance, 2025, Bilot et al., 2026b], which is the established framework for intrusion detection research, using the same data splits and hyperparameter tuning approach used by the community [Bilot et al., 2025]. All experiments are trained with the AdamW optimizer [Loshchilov and Hutter, 2017]. Full hyperparameters are in section D.

Setup. All models are trained on Ubuntu 22.04 servers equipped with AMD EPYC 7343 CPUs and NVIDIA A100 GPUs. Downstream intrusion detection experiments use the same hardware and follow each system's original training protocol, differing only in the entity embedding input.

Evaluation Metrics. We report the attack detection precision (ADP) [Bilot et al., 2025], denoted AP in this paper: the average precision of detecting the attacks of a dataset, *i.e.*, the area under the curve of the fraction of attacks with at least one flagged node versus node-level precision. Like AP, it is threshold-independent and suited to extreme class imbalance (Table 1) [Davis and Goadrich, 2006].

5 Results and Analysis

We evaluate along five axes: **(Q1)** whether SPIDER embeddings improve state-of-the-art detectors as a drop-in replacement (§5.1), **(Q2)** how SPIDER compares against alternative encoders pretrained on the same corpus as the frozen entity embedding backbone of a fixed detector (§5.2), **(Q3)** which components contribute most to performance (§5.3), **(Q4)** whether SPIDER meets inference requirements for real-time deployment (§5.4), and **(Q5)** whether it generalizes beyond its pretraining data (§5.5).

Table 2: **AP score across downstream detectors.** ✗: native text encoder; ❄: SPIDER frozen embeddings without fine-tuning; 🔥: SPIDER embeddings with fine-tuning.

	VELOX			ORTHRUS			FLASH			KAIROS		
Dataset	✗	❄	🔥	✗	❄	🔥	✗	❄	🔥	✗	❄	🔥
CD-E3	$0.70_{\pm.10}$	$0.84_{\pm.19}$	$0.98_{\pm.01}$	$0.74_{\pm.06}$	$0.79_{\pm.04}$	$0.89_{\pm.05}$	$0.06_{\pm.03}$	$0.34_{\pm.00}$	$0.36_{\pm.32}$	$0.12_{\pm.02}$	$0.43_{\pm.04}$	$0.47_{\pm.09}$
TH-E3	$0.77_{\pm.12}$	$1.00_{\pm.00}$	$1.00_{\pm.00}$	$0.38_{\pm.11}$	$0.42_{\pm.16}$	$0.44_{\pm.07}$	$0.03_{\pm.04}$	$0.23_{\pm.05}$	$0.27_{\pm.08}$	$0.45_{\pm.06}$	$0.57_{\pm.08}$	$0.69_{\pm.07}$
TH-E5	$0.96_{\pm.02}$	$1.00_{\pm.00}$	$1.00_{\pm.00}$	$0.07_{\pm.07}$	$0.10_{\pm.08}$	$0.29_{\pm.05}$	$0.02_{\pm.01}$	$0.06_{\pm.03}$	$0.17_{\pm.10}$	$0.17_{\pm.06}$	$0.19_{\pm.04}$	$0.20_{\pm.05}$
CS-E5	$0.14_{\pm.10}$	$0.91_{\pm.02}$	$0.74_{\pm.08}$	$0.67_{\pm.09}$	$0.66_{\pm.11}$	$0.67_{\pm.06}$	$0.10_{\pm.10}$	$0.21_{\pm.04}$	$0.23_{\pm.08}$	$0.33_{\pm.00}$	$0.35_{\pm.01}$	$0.38_{\pm.05}$
OpTC	$0.08_{\pm.03}$	$0.36_{\pm.08}$	$0.56_{\pm.09}$	$0.61_{\pm.08}$	$1.00_{\pm.00}$	$0.90_{\pm.05}$	$0.00_{\pm.00}$	$1.00_{\pm.00}$	$0.82_{\pm.10}$	$0.49_{\pm.02}$	$0.64_{\pm.02}$	$0.66_{\pm.00}$

5.1 Intrusion Detection Improvement

We evaluate whether SPIDER embeddings can act as a drop-in text encoder for recent detectors (Table 2). We consider four representative systems: FLASH [Rehman et al., 2024], KAIROS [Cheng et al., 2023], ORTHRUS [Jiang et al., 2025], and VELOX [Bilot et al., 2025]; full details in section I. We evaluate three settings. ✗: each system uses its native text encoder, *i.e.*hierarchical feature hashing for KAIROS and Word2Vec for the others. ❄: we replace the text encoder with the pretrained SPIDER checkpoint, kept frozen, so the detector receives out-of-the-box priors at zero adaptation cost. 🔥: we continue SPIDER's pretraining on each dataset's training split with the same objective, a lower learning rate ($3 \cdot 10^{-5}$), and 5 epochs (section D). The ❄ setting measures out-of-the-box generalization, while 🔥 reflects a more realistic deployment in which a customer adapts SPIDER to domain-specific activity.

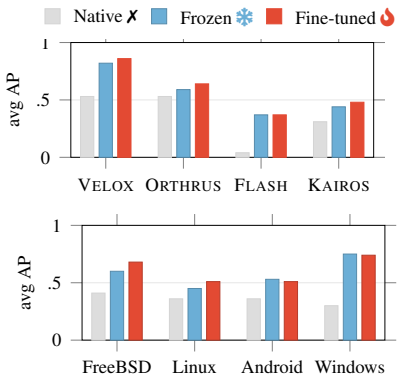

Figure 3: **Average AP per detector (top) and per OS (bottom)** with native (✗), frozen (❄), and fine-tuned (🔥) entity encoders. Top: averaged over the five datasets; bottom: averaged over the four detectors and the OS's datasets (Table 2).

Results. SPIDER embeddings improve average AP across all detectors (Figure 3). As shown in Table 2, out of the box (❄), SPIDER matches or exceeds native featurization on 19/20 detector–dataset pairs, and fine-tuning (🔥) improves over the frozen variant on 17/20 pairs. The frozen variant helps because SPIDER replaces lexical-only encoders (Word2Vec, feature hashing) with behavior-aware priors learned offline from millions of provenance neighborhoods, giving each detector a stronger starting point at no inference-time cost. Fine-tuning then adapts these priors to deployment-specific entities that are missing or underrepresented in pretraining (*e.g.*concrete IP addresses and entities tied to specific software versions), recovering signal that frozen embeddings cannot capture. Gains are largest on Windows (OpTC), where long registry-style paths and command lines with embedded GUIDs are poorly handled by native encoders. High variance in a few entries reflects extreme class imbalance, where AP can depend on the relative ranking of a small number of attack nodes [Bilot et al., 2025]. Overall, these results support the claim that stable interaction patterns can be learned as transferable priors and reused at inference without graph access.

5.2 Encoder Evaluation

We compare SPIDER against alternative encoders pretrained on the same corpus, and evaluate all methods using frozen embeddings in a downstream detection setting (Table 3). We group baselines into two categories. **Graph-level** methods leverage graph structure during both training and inference, which requires storing and traversing the full provenance graph at deployment time. These methods can achieve stronger performance, but their runtime cost makes them impractical in realistic deployments [Dong et al., 2023]. **Entity-level** methods compute embeddings from entity text alone at inference. They scale to enterprise settings, but lack direct access to neighborhood information. Some entity-level methods still inject graph context during training, for example via random walks or graph-supervised objectives. SPIDER falls into this subset: it uses graph-level supervision during training while retaining entity-level efficiency at deployment.

We compare against twelve baselines spanning these categories (see the full descriptions in Section I.2). All methods are pretrained on the same corpus (section 4) and evaluated as backbone encoders for VELOX [Bilot et al., 2025], the strongest detector in Section 5.1, with hyperparameter tuning detailed in section D.

Results. Among the baselines, graph-level methods perform well on some individual datasets (that is, specific operating systems), but their results are less consistent across benchmarks and they require full graph access at inference. The best of these, GraphMAE, reaches 0.76 avg AP. In contrast, SPIDER achieves 0.82 avg AP across all datasets while using only entity text at inference, suggesting that offline graph pretraining captures structure that transfers across settings. This also shows that SPIDER remains robust under the domain gap between pretraining and evaluation data. Entity-level methods trained only on text, as in many recent detectors, are already competitive, indicating that textual attributes carry substantial semantic signal for detection. Notably, GPT-2 1.5B reaches 0.80 avg AP, but SPIDER exceeds it with roughly 290× fewer parameters. Methods trained on graph walks often improve performance on individual datasets by incorporating causal context, yet they underperform on average ($\leq$ 0.70). Overall, SPIDER achieves the best average AP, matches or surpasses all baselines on four of the five datasets, and is the only encoder that is consistently strong across all four operating systems. These gains do not stem from tokenization alone: with SPIDER's tokenizer, Word2Vec and FastText reach only 0.62 and 0.60 avg AP (Section L.5).

Table 3: **Detection performance of various backbones (frozen ❄).** AP as mean ± standard deviation over 10 runs. VELOX is used as downstream detector. Best per column in **bold**.

Method	#Params	CD-E3	TH-E3	TH-E5	CS-E5	OpTC	Avg
Graph-level methods							
GAE	5.3 M	$0.78_{\pm.08}$	$0.96_{\pm.02}$	$\mathbf{1.00}_{\pm.00}$	$0.67_{\pm.15}$	$0.09_{\pm.02}$	0.70
DGI	5.4 M	$\mathbf{0.98}_{\pm.00}$	$\mathbf{1.00}_{\pm.00}$	$0.26_{\pm.03}$	$0.53_{\pm.16}$	$0.04_{\pm.02}$	0.56
GraphMAE	5.5 M	$0.90_{\pm.03}$	$0.94_{\pm.02}$	$\mathbf{1.00}_{\pm.00}$	$0.87_{\pm.04}$	$0.07_{\pm.03}$	0.76
Entity-level methods							
Word2Vec	4.1 M†	$0.70_{\pm.10}$	$0.77_{\pm.12}$	$0.96_{\pm.02}$	$0.14_{\pm.10}$	$0.08_{\pm.03}$	0.53
FastText	4.1 M†	$0.60_{\pm.08}$	$0.80_{\pm.00}$	$0.12_{\pm.10}$	$0.89_{\pm.02}$	$0.26_{\pm.03}$	0.53
GPT-2 1.5B	1.5 B	$0.97_{\pm.01}$	$0.80_{\pm.00}$	$\mathbf{1.00}_{\pm.00}$	$0.90_{\pm.02}$	$0.31_{\pm.10}$	0.80
Llama 3.2 1B	1 B	$0.43_{\pm.20}$	$\mathbf{1.00}_{\pm.00}$	$0.92_{\pm.02}$	$0.55_{\pm.13}$	$0.05_{\pm.01}$	0.59
DeepWalk	4.1 M†	$0.29_{\pm.03}$	$\mathbf{1.00}_{\pm.00}$	$\mathbf{1.00}_{\pm.00}$	$0.85_{\pm.06}$	$0.20_{\pm.00}$	0.67
Node2Vec	4.1 M†	$0.31_{\pm.05}$	$\mathbf{1.00}_{\pm.00}$	$\mathbf{1.00}_{\pm.00}$	$0.83_{\pm.09}$	$0.25_{\pm.07}$	0.68
BERT	5.3 M	$0.28_{\pm.04}$	$0.89_{\pm.07}$	$\mathbf{1.00}_{\pm.00}$	$0.77_{\pm.07}$	$0.31_{\pm.05}$	0.65
ModernBERT	6.3 M	$0.66_{\pm.13}$	$0.92_{\pm.03}$	$\mathbf{1.00}_{\pm.00}$	$0.76_{\pm.17}$	$0.02_{\pm.00}$	0.67
CyberGFM	5.5 M	$0.77_{\pm.02}$	$0.14_{\pm.07}$	$0.87_{\pm.06}$	$0.65_{\pm.12}$	$0.30_{\pm.09}$	0.55
SPIDER	5.2 M	$0.84_{\pm.19}$	$\mathbf{1.00}_{\pm.00}$	$\mathbf{1.00}_{\pm.00}$	$\mathbf{0.91}_{\pm.02}$	$\mathbf{0.36}_{\pm.32}$	**0.82**

† Transductive: parameter count scales with vocabulary size.

5.3 Ablation Study

We ablate the core components of SPIDER (Table 4): (i) the domain-specific tokenizer, replaced with a plain BPE; (ii) the SCE distillation loss, replaced with mean squared error (MSE); (iii) the supervised contrastive loss, replaced with cross-entropy (CE) over entity classes; (iv) behavioral signatures, removed so the teacher uses only graph-based node embeddings; and (v) graph embeddings, removed so the teacher uses only behavioral signatures. We also remove the teacher entirely, evaluating a pretrained student that predicts behavioral signatures, classifies semantic classes, or clusters them via SupCon, the latter isolating the teacher's contribution from the choice of loss.

Table 4: **Ablation study.** Each row disables one component.

Tok.	$\mathcal{L}_{\text{dis}}$	$\mathcal{L}_{\text{con}}$	Sign.	Emb.	CD-E3	TH-E3	TH-E5	CS-E5	OpTC
✓	✓	✓	✓	✓	$0.84_{\pm.19}$	$\mathbf{1.00}_{\pm.00}$	$\mathbf{1.00}_{\pm.00}$	$\mathbf{0.91}_{\pm.02}$	$\mathbf{0.36}_{\pm.32}$
BPE	✓	✓	✓	✓	$0.85_{\pm.08}$	$0.77_{\pm.12}$	$0.76_{\pm.03}$	$0.70_{\pm.11}$	$0.02_{\pm.00}$
✓	**MSE**	✓	✓	✓	$0.95_{\pm.03}$	$\mathbf{1.00}_{\pm.00}$	$0.91_{\pm.00}$	$0.53_{\pm.17}$	$0.03_{\pm.01}$
✓	✓	**CE**	✓	✓	$0.69_{\pm.30}$	$0.91_{\pm.09}$	$0.68_{\pm.05}$	$0.74_{\pm.07}$	$0.06_{\pm.07}$
✓	✓	✓	✗	✓	$0.38_{\pm.09}$	$0.71_{\pm.03}$	$0.71_{\pm.02}$	$0.68_{\pm.03}$	$0.28_{\pm.09}$
✓	✓	✓	✓	✗	$\mathbf{0.96}_{\pm.01}$	$0.66_{\pm.00}$	$0.57_{\pm.01}$	$0.76_{\pm.14}$	$0.01_{\pm.01}$
Teacher ablation (only student)									
Student predicts signature (CE)					$0.80_{\pm.14}$	$0.94_{\pm.00}$	$0.32_{\pm.06}$	$0.70_{\pm.04}$	$0.08_{\pm.05}$
Student predicts class (CE)					$0.73_{\pm.03}$	$0.67_{\pm.01}$	$0.37_{\pm.02}$	$0.66_{\pm.07}$	$0.17_{\pm.04}$
Student clusters class (SupCon)					$0.76_{\pm.04}$	$0.77_{\pm.04}$	$0.65_{\pm.06}$	$0.74_{\pm.06}$	$0.20_{\pm.03}$

Results. The contrastive loss $\mathcal{L}_{\text{con}}$ is the most critical component. Replacing it with CE reduces mean AP by 0.20, suggesting that classification objectives collapse within-class variation that is important for anomaly detection. The teacher's two input channels are complementary: (1) removing behavioral signatures produces the steepest single-dataset drop (CD-E3: 0.84 → 0.38) because it erases functional distinctions, whereas (2) removing graph embeddings instead hurts CS-E5 and OpTC the most. Removing the graph embedding hurts average performance because the model loses information about *who* the entity interacts with, making it harder to distinguish entities with similar interaction patterns. Replacing the domain-specific tokenizer with standard BPE degrades performance on four of five datasets, since BPE fragments system-specific tokens into semantically

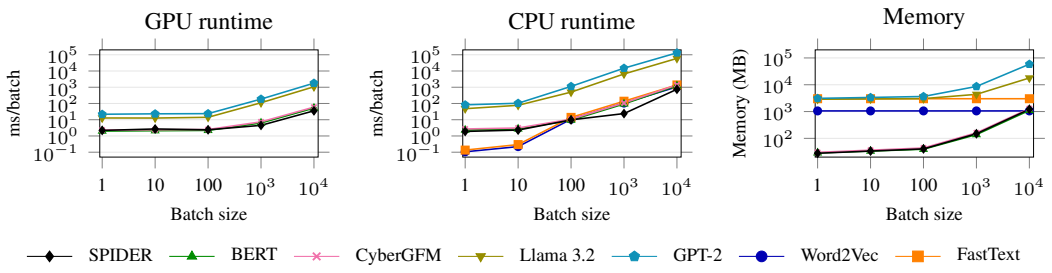

Figure 4: **Inference cost on GPU, CPU, and Memory.** Results are averaged across 100 runs.

opaque subwords. SCE distillation outperforms MSE on average by preserving the teacher's soft output structure. An edge-type-aware R-GCN [Schlichtkrull et al., 2018] teacher matches graph attention (0.82 avg AP), but type-agnostic GraphSAGE [Hamilton et al., 2017] and GIN [Xu et al., 2019] drop to 0.74 and 0.72 (Section L.3).

Value of graph distillation. The teacher-free SupCon row is a controlled baseline for graph distillation, sharing SPIDER's tokenizer, semantic classes, and contrastive loss but no graph teacher: it reaches 0.62 avg AP versus 0.82 (-0.20), and a student predicting behavioral signatures directly only 0.57. Graph-contextualized targets thus cannot be recovered from text and semantic supervision alone, and graph distillation contributes substantially beyond the domain-specific components.

5.4 Inference Cost

We measure the inference runtime and memory usage required to compute SPIDER embeddings across varying batch sizes (Figure 4). We compare against baseline encoders for batch sizes ranging from 1 to 10,000 samples. Graph-based methods are omitted as their cost depends on the full graph size. Word2Vec and FastText are CPU-only baselines and are therefore excluded from the GPU panel. We detail the training cost in section H.

Runtime. On GPU, latency is dominated by kernel dispatch overhead for $b \leq 100$, with all non-LLM methods clustering at 2–3 ms/batch and billion-parameter LLMs roughly an order of magnitude slower. Beyond $b = 100$, runtime scales near-linearly. Overall, SPIDER requires only 35 ms to process a batch of $b{=}10k$ entities on GPU. On CPU, SPIDER processes a single entity in $\sim$2 ms (10 ms for 100 entities), scaling to 750 ms for $10k$ entities. Overall, SPIDER ranks among the fastest methods on both CPU and GPU.

Memory. SPIDER maintains a compact memory footprint in the range $\sim$28–160 MB across batch sizes, within the $\leq$160 MB budget recommended for industrial endpoint deployment [Dong et al., 2023] and two orders of magnitude below billion-parameter LLMs ($\sim$2.9–8.6 GB even in fp16). Word2Vec and FastText appear as flat lines because their footprint scales with the corpus vocabulary rather than the batch: on `CD-E3` ($|V| \approx 1$M, $d{=}256$) they already occupy $\sim$1.1 and $\sim$3.0 GB. In contrast, all other methods reported above have a fixed parameter budget independent of graph or vocabulary size.

SPIDER can run on low-resource CPU-only machines with a small memory footprint, enabling millisecond-level inference and efficient scaling across multiple hosts.

5.5 Generalization and Robustness

We stress-test transfer using VELOX with frozen embeddings (section L).

Unseen operating systems. Pretraining SPIDER without the target OS (*e.g.*no Windows data for `OpTC`) lowers avg AP only from 0.82 to 0.81 (Table 13): most priors transfer via the OS-agnostic event vocabulary, and the gap reflects platform-specific entities (*e.g.*registry paths, Android intents).

Rare and unseen entities. Stratifying test entities by frequency in the pretraining corpus, avg AP degrades gracefully from 0.89 (common) to 0.83 (rare) and 0.80 (unseen) (Table 14), as unseen labels fall back on type and context tokens (Section 3.1).

Spurious correlations. Four design choices limit the risk that the student inherits environment-specific correlations from the teacher: (i) per-epoch neighborhood re-sampling (Section 3.2); (ii) hub and noise edge filtering (section F); (iii) behavioral signatures that abstract neighbors into semantic classes, L2-balanced against the GNN branch (Equation 2); and (iv) anonymized IPs, hashes, and usernames. The transfer results above suggest such artifacts are largely not inherited.

Statistical significance. Paired Wilcoxon signed-rank tests [Wilcoxon, 1945] over 10 seeds show significant gains over Word2Vec ($p \leq 0.004$, large effect sizes) on all five datasets (Table 18).

6 Related Work

Provenance-based intrusion detection. Provenance-based detectors learn benign system behavior and flag deviations at the node [Wang et al., 2021, Li et al., 2024, Rehman et al., 2024, Jiang et al., 2025], subgraph [Cheng et al., 2023], or joint [Jia et al., 2024] level, using GNNs or temporal graph networks. Bilot et al. [2025] showed that an MLP over Word2Vec entity embeddings can match them, suggesting that entity representations matter as much as detection architectures. Yet all learn entity features from scratch per dataset; SPIDER instead provides one pretrained encoder for all of them (Section 5.1).

Graph self-supervised learning and foundation models. Graph self-supervised methods learn transferable representations via mutual-information maximization [Veličković et al., 2018], masked autoencoding [Hou et al., 2022], edge reconstruction [Kipf and Welling, 2016], or contrastive learning [Qiu et al., 2020], and graph foundation models target cross-graph generalization [Liu et al., 2023, Mao et al., 2024]; CyberGFM [King et al., 2026] applies next-token prediction over host-level graph walks to lateral movement detection. These methods require graph access at inference or see structure only through walks; we compare against each family in Section 5.2.

Knowledge distillation on graphs. Knowledge distillation [Hinton et al., 2015] transfers a teacher's knowledge into a cheaper student, including across input modalities [Gupta et al., 2016]. On graphs [Tian et al., 2023a], GLNN [Zhang et al., 2022] and NOSMOG [Tian et al., 2023b] distill a pretrained GNN into a graph-free MLP over node features, and GRAD [Mavromatis et al., 2023] co-trains a GNN teacher with a graph-free language-model student on a text-attributed graph. These methods target supervised node classification on the graph they are trained on. SPIDER differs in its setting rather than its building blocks: (i) no pretrained provenance teacher exists, so ours is co-trained as an EMA of the student with edge-conditioned graph attention; (ii) targets are behavioral priors structured by semantic classes and signatures, not task labels; and (iii) the student must be inductive over open-vocabulary text to transfer across graphs, OSes, and organizations. An adapted GLNN baseline trails SPIDER by 0.14 avg AP, most on Android and Windows (Section L.4).

7 Conclusion

We presented SPIDER, a text-only encoder for operating system entities trained via offline provenance graph distillation. By distilling behavioral context from a graph-aware teacher into a lightweight transformer student, SPIDER produces behavior-aware entity embeddings from labels alone in a single forward pass, with no graph access at inference. Across five benchmarks spanning Linux, FreeBSD, Windows, and Android, one pretrained checkpoint consistently improves four recent provenance-based detectors as a drop-in text backbone and outperforms alternative pretrained encoders. Its small footprint enables fast, memory-efficient inference for real-time detection at scale.

Limitations & Future Work. We evaluate SPIDER as an entity encoder for intrusion detection, leaving open whether its behavioral priors transfer to other tasks such as forensic attack reconstruction [Fang et al., 2022, Xu et al., 2022], alert triage [Hassan et al., 2019], or malware classification [Han et al., 2021, Liu et al., 2026]. Our adversarial analysis (section B) shows that *renaming attacks* fail against the intrusion detection pipeline (*i.e.*the detector flags interactions diverging from the impersonated identity's profile) but this robustness may not extend to applications consuming SPIDER embeddings directly (*e.g.*embedding-based triage). Fine-tuning mitigates this, but a complete defense against label-level mimicry across applications remains open, as does, for all provenance-based detectors, perfect behavioral mimicry [Goyal et al., 2023].

Acknowledgment and Disclosure of Funding

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

Competing interests. The authors declare no competing interests.

References

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

Tristan Bilot, Xueyuan Han, and Thomas Pasquier. Learning Transferable Representations from Operating System Entities via Provenance Graph Distillation. *Conference on Neural Information Processing System (NeurIPS'26)*, 2026a.

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

Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jégou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. Emerging Properties in Self-supervised Vision Transformers. In *International Conference on Computer Vision (ICCV'21)*. IEEE/CVF, 2021.

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

DARPA. Transparent Computing Engagement 3 Data Release. https://github.com/darpa-i2o/Transparent-Computing/blob/master/README-E3.md, 2018.

DARPA. Transparent Computing Engagement 5 Data Release. https://github.com/darpa-i2o/Transparent-Computing, 2019.

DARPA. Operationally Transparent Cyber (OpTC) Data Release. https://github.com/FiveDirections/OpTC-data, 2019.

Jesse Davis and Mark Goadrich. The relationship between precision-recall and roc curves. In *International Conference on Machine learning (ICML'06)*, pages 233–240, 2006.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Conference of the North American Chapter of the Association for Computational Linguistics (NAACL'19)*, 2019.

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

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

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

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

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The Llama 3 Herd of Models. *arXiv preprint arXiv:2407.21783*, 2024.

Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Avila Pires, Zhaohan Guo, Mohammad Gheshlaghi Azar, et al. Bootstrap Your Own Latent A New Approach to Self-Supervised Learning. *Advances in Neural Information Processing Systems (NeurIPS'20)*, 2020.

Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'16)*. ACM, 2016.

Saurabh Gupta, Judy Hoffman, and Jitendra Malik. Cross Modal Distillation for Supervision Transfer. In *Conference on Computer Vision and Pattern Recognition (CVPR'16)*. IEEE, 2016.

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Conference on Neural Information Processing Systems (NeurIPS'17)*, 2017.

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

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

Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum Contrast for Unsupervised Visual Representation Learning. In *Conference on Computer Vision and Pattern Recognition (CVPR'20)*. IEEE/CVF, 2020.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the Knowledge in a Neural Network. *arXiv preprint arXiv:1503.02531*, 2015.

Zhenyu Hou, Xiao Liu, Yukuo Cen, Yuxiao Dong, Hongxia Yang, Chunjie Wang, and Jie Tang. GraphMAE: Self-Supervised Masked Graph Autoencoders. In *SIGKDD Conference on Knowledge Discovery and Data Mining (KDD'22)*. ACM, 2022.

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

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

Armand Joulin, Edouard Grave, Piotr Bojanowski, Matthijs Douze, Hérve Jégou, and Tomas Mikolov. Fasttext.zip: Compressing Text Classification Models. *arXiv preprint arXiv:1612.03651*, 2016.

Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomáš Mikolov. Bag of Tricks for Efficient Text Classification. In *Conference of the European Chapter of the Association for Computational Linguistics (EACL'17)*, 2017.

Prannay Khosla, Piotr Teterwak, Chen Wang, Aaron Sarna, Yonglong Tian, Phillip Isola, Aaron Maschinot, Ce Liu, and Dilip Krishnan. Supervised Contrastive Learning. *Advances in Neural Information Processing Systems (NeurIPS'20)*, 2020.

Isaiah J King, Bernardo Trindade, Benjamin Bowman, and H Howie Huang. CyberGFM: Graph Foundation Models for Lateral Movement Detection in Enterprise Networks. *arXiv preprint arXiv:2601.05988*, 2026.

Samuel T King and Peter M Chen. Backtracking Intrusions. *ACM SIGOPS Operating Systems Review*, 2003.

Thomas N Kipf and Max Welling. Variational Graph Auto-encoders. *arXiv preprint arXiv:1611.07308*, 2016.

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

Jiahao Liu, Jun Zeng, Fabio Pierazzi, Ziqi Yang, Lorenzo Cavallaro, and Zhenkai Liang. Unraveling the key of machine learning-based android malware detection. *ACM Transactions on Software Engineering and Methodology*, 2026.

Jiawei Liu, Cheng Yang, Zhiyuan Lu, Junze Chen, Yibo Li, Mengmei Zhang, Ting Bai, Yuan Fang, Lichao Sun, Philip S Yu, et al. Towards Graph Foundation Models: A Survey and Beyond. *arXiv preprint arXiv:2310.11829*, 2023.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

Haitao Mao, Zhikai Chen, Wenzhuo Tang, Jianan Zhao, Yao Ma, Tong Zhao, Neil Shah, Mikhail Galkin, and Jiliang Tang. Position: Graph Foundation Models are Already Here. In *International Conference on Machine Learning (ICML'24)*, 2024.

Costas Mavromatis, Vassilis N. Ioannidis, Shen Wang, Da Zheng, Soji Adeshina, Jun Ma, Han Zhao, Christos Faloutsos, and George Karypis. Train Your Own GNN Teacher: Graph-Aware Distillation on Textual Graphs. In *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD'23)*, 2023.

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

Thomas Pasquier, Xueyuan Han, Mark Goldstein, Thomas Moyer, David M. Eyers, Margo I. Seltzer, and Jean Bacon. Practical Whole-system Provenance Capture. In *Symposium on Cloud Computing (SoCC'17)*. ACM, 2017.

Thomas Pasquier, Xueyuan Han, Thomas Moyer, Adam Bates, Olivier Hermant, David M. Eyers, Jean Bacon, and Margo Seltzer. Runtime Analysis of Whole-System Provenance. In *Conference on Computer and Communications Security (CCS'18)*. ACM, 2018.

Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online Learning of Social Representations. In *SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'14)*. ACM, 2014.

Jiezhong Qiu, Qibin Chen, Yuxiao Dong, Jing Zhang, Hongxia Yang, Ming Ding, Kuansan Wang, and Jie Tang. GCC: Graph Contrastive Coding for Graph Neural Network Pre-Training. In *SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'20)*, pages 1150–1160. ACM, 2020.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 2019.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67, 2020.

Mati Ur Rehman, Hadi Ahmadi, and Wajih Ul Hassan. Flash: A Comprehensive Approach to Intrusion Detection via Provenance Graph Representation Learning. In *Symposium on Security and Privacy (S&P'24)*. IEEE, 2024.

Michael Schlichtkrull, Thomas N. Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. Modeling Relational Data with Graph Convolutional Networks. In *European Semantic Web Conference (ESWC'18)*, 2018.

Yunsheng Shi, Zhengjie Huang, Shikun Feng, Hui Zhong, Wenjin Wang, and Yu Sun. Masked Label Prediction: Unified Message Passing Model for Semi-Supervised Classification. In *International Joint Conference on Artificial Intelligence (IJCAI'21)*, 2021.

Yijun Tian, Shichao Pei, Xiangliang Zhang, Chuxu Zhang, and Nitesh V. Chawla. Knowledge Distillation on Graphs: A Survey. *arXiv preprint arXiv:2302.00219*, 2023a.

Yijun Tian, Chuxu Zhang, Zhichun Guo, Xiangliang Zhang, and Nitesh V. Chawla. Learning MLPs on Graphs: A Unified View of Effectiveness, Robustness, and Efficiency. In *International Conference on Learning Representations (ICLR'23)*, 2023b.

UBC Provenance. PIDSMaker: A Framework for Provenance-Based Intrusion Detection Systems. https://github.com/ubc-provenance/PIDSMaker, 2025.

Petar Veličković, William Fedus, William L Hamilton, Pietro Liò, Yoshua Bengio, and R Devon Hjelm. Deep Graph Infomax. *arXiv preprint arXiv:1809.10341*, 2018.

Su Wang, Zhiliang Wang, Tao Zhou, Hongbin Sun, Xia Yin, Dongqi Han, Han Zhang, Xingang Shi, and Jiahai Yang. THREATRACE: Detecting and Tracing Host-Based Threats in Node Level Through Provenance Graph Learning. *IEEE Transactions on Information Forensics and Security*, 2021.

Benjamin Warner, Antoine Chaffin, Benjamin Clavié, Orion Weller, Oskar Hallström, Said Taghadouini, Alexis Gallagher, Raja Biswas, Faisal Ladhak, Tom Aarsen, et al. Smarter, Better, Faster, Longer: A Modern Bidirectional Encoder for Fast, Memory Efficient, and Long Context Finetuning and Inference. In *Annual Meeting of the Association for Computational Linguistics (ACL'25)*, 2025.

Frank Wilcoxon. Individual Comparisons by Ranking Methods. *Biometrics Bulletin*, 1(6):80–83, 1945.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How Powerful are Graph Neural Networks? In *International Conference on Learning Representations (ICLR'19)*, 2019.

Zhiqiang Xu, Pengcheng Fang, Changlin Liu, Xusheng Xiao, Yu Wen, and Dan Meng. DEPCOMM: Graph Summarization on System Audit Logs for Attack Investigation. In *Symposium on Security and Privacy (S&P'22)*. IEEE, 2022.

Shichang Zhang, Yozen Liu, Yizhou Sun, and Neil Shah. Graph-less Neural Networks: Teaching Old MLPs New Tricks Via Distillation. In *International Conference on Learning Representations (ICLR'22)*, 2022.

Michael Zipperle, Florian Gottwalt, Elizabeth Chang, and Tharam Dillon. Provenance-based intrusion detection systems: A survey. *ACM Computing Surveys*, 55(7):1–36, 2022.

A Qualitative Embedding Analysis

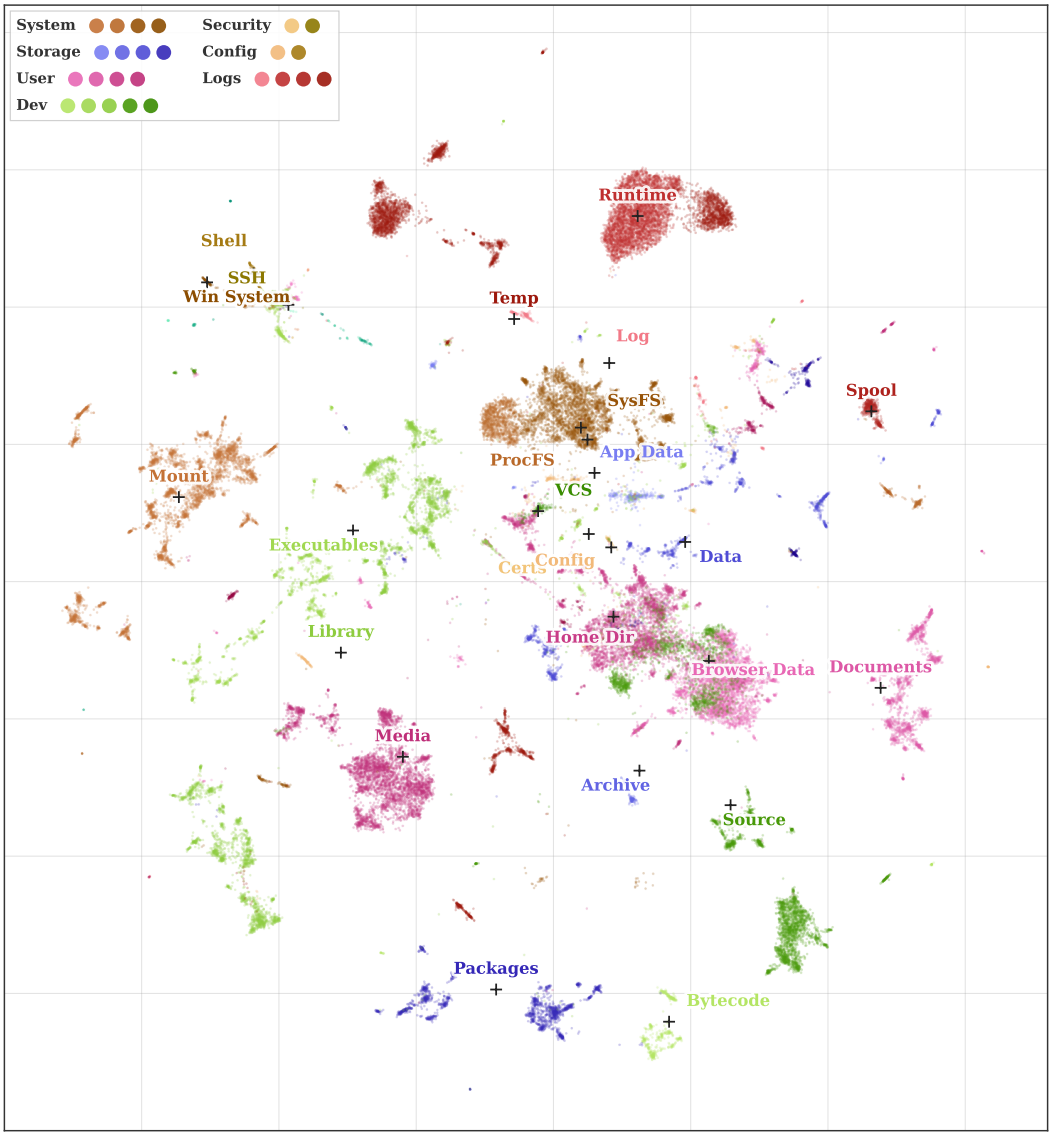

Figure 5: Two-dimensional UMAP projection of the cross-OS embedding space ($n \approx 500$k entities across Linux, Windows, Android, and FreeBSD). Points are colored by semantic class and grouped into eight super-categories (System, Port, Storage, User, Dev, Security, Config, Logs) sharing similar hues. The most prevalent classes are labeled with centroids (+) marking cluster centers.

A.1 Embedding space

Figure 5 shows a projection of the learned embedding space for files, processes and sockets sampled from the test datasets. Clusters that share similar roles in the provenance graph occupy neighboring regions: SSH and Shell sit adjacent to the Port cluster, reflecting shared involvement in network-facing activity, while Config and Certs border both System and Security, consistent with their dual role in configuration and access control. Notably, the value of the embedding does not lie in perfectly separating entities by their semantic class, a straightforward supervised classifier could achieve that. Rather, its strength is that entities naturally position themselves *between* multiple clusters when they participate in overlapping functional contexts. A browser cache entry, for example, may sit between User and Storage because it is both a user-generated artifact and a data store; a VCS object may

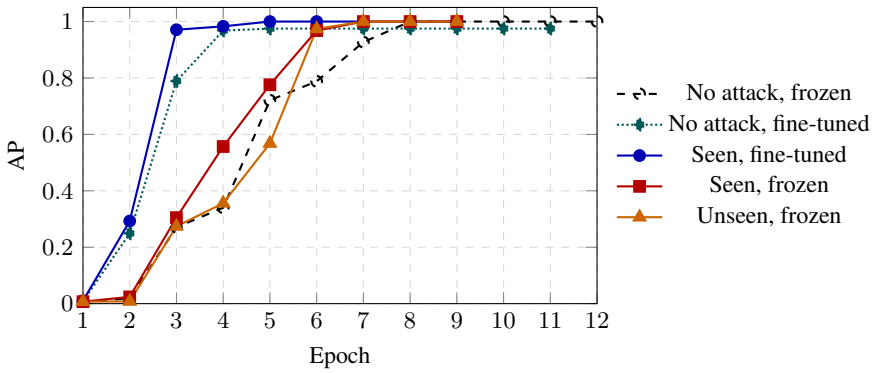

Figure 6: **Impact of renaming attacks on SPIDER.** AP across detector training epochs under renaming attacks on CADETS-E3 with VELOX as the downstream detector.

bridge Dev and Storage. These inter-cluster gradients capture the polymorphic nature of OS artifacts in provenance graphs, encoding not a single label but the blend of behavioral roles each entity plays across system activity.

B Adversarial Robustness

A natural concern with any pretrained encoder is whether an adversary aware of the embedding space can craft attacks that evade detection. We evaluate SPIDER against two threat models that target distinct components of the detection pipeline: *renaming attacks* (Section B.1), which target the entity encoder by relabeling malicious binaries to inherit benign embeddings, and *mimicry attacks* (Section B.2), which target the downstream detector by inserting spurious edges to make the attack region resemble benign neighborhoods.

B.1 Renaming Attack

Because SPIDER maps text to embeddings, an adversary could rename a malicious binary to a benign-looking path (e.g., `payload.exe` → `firefox.exe`) to obtain a benign embedding from SPIDER. This vulnerability is shared by all text-based featurizers used in current PIDS (Word2Vec, FastText, HFH, Doc2Vec): any encoder that maps text to a representation can be fooled by adversarial relabeling. Two factors mitigate the impact in practice. First, detection is performed by a downstream model that observes the entity's interactions in the provenance graph: a renamed payload that connects to suspicious sockets or writes to unusual paths will still produce anomalous edge scores even if its node embedding is benign. Moreover, the downstream detector is trained from scratch on the target environment and learns each entity's expected activity: if it has seen genuine `firefox.exe` behavior in training, an attacker renaming a payload to `firefox.exe` at inference produces an even larger distribution gap and is *more* likely to be flagged. Second, continued pretraining on deployment data progressively re-aligns embeddings with observed behavior, so a persistent renamed binary exhibiting non-Firefox interactions will drift away from the Firefox cluster over time. We empirically validate both factors below.

Setup. We replace, in the test split of CADETS-E3, the embedding of every attack entity successfully detected by SPIDER (e.g., `main`, `pEja72mA`, `XIM`, `tmux-1002`, `font`, `sendmail`) with the embedding of a benign target. Bare names and path-style variants are both matched, with prefix preservation (e.g., `/tmp/main` → `/tmp/<benign>`). Train and validation embeddings are left untouched, matching a threat model in which the adversary cannot influence defender training data. We evaluate three scenarios.

(i) Seen, frozen. Each malicious entity is assigned a distinct benign target drawn from the top-100 most frequent processes in the training set (e.g., `firefox`, `sshd`). This is a worst case scenario in which the attacker mimics processes SPIDER has sharp embeddings for, with SPIDER frozen at deployment.

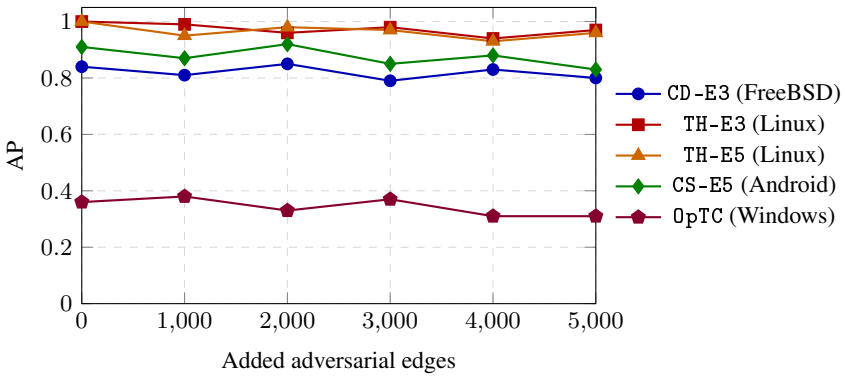

Figure 7: **Impact of mimicry attacks on SPIDER.** Benign edges are injected into the attack region to force misclassification.

(ii) Seen, fine-tuned. Same attack as (i), but SPIDER is fine-tuned on the train set before training the downstream detector, sharpening benign embeddings around their actual deployment behavior.

(iii) Unseen, frozen. The attacker mimics a benign-looking name absent from the pretraining corpus, with SPIDER frozen.

Results. Figure 6 reports the AP of VELOX across training epochs for the three scenarios, alongside the frozen and fine-tuned no-attack baselines from Table 2. All three attacked variants converge to AP $= 1.0$, and *Seen, continued pretraining* recovers the no-attack convergence speed almost exactly. Despite renamed processes inheriting a known-benign embedding, the detector flags them because their provenance-graph interactions diverge from the impersonated identity's behavioral profile. This supports the claim that text-only mimicry is insufficient against a structure-aware detector built on SPIDER embeddings, and that continued pretraining is a complementary defense that hardens benign embeddings against impersonation rather than relaxing them.

B.2 Mimicry Attack

A second kind of attack likely in a deployed setting is mimicry attacks, in which the attacker inserts spurious graph elements into the attack region to make its distribution resemble benign activity. Following the protocol used by our four detector baselines [Cheng et al., 2023, Rehman et al., 2024, Jiang et al., 2025, Bilot et al., 2025], we evaluate SPIDER under the mimicry attack proposed by Goyal et al. [2023], varying the number of inserted edges from 0 to 5,000 across all five datasets with VELOX as the downstream detector and frozen SPIDER embeddings. Figure 7 shows that AP remains stable across the full perturbation range on every dataset, indicating that the attack fails to suppress the anomaly score of genuine attack nodes. This is consistent with our design: because SPIDER embeds entities from text alone, the encoder output is unaffected by inserted edges, and the behavioral priors learned during pretraining give the detector a stable starting point that is harder to dilute through neighborhood-level mimicry than featurizers trained jointly with the detector.

C Hyperparameter Sensitivity

To validate the robustness of SPIDER to hyperparameter choices, we perform a sensitivity analysis around the best configuration reported in Table 6. For each of three key hyperparameters (learning rate, neighborhood size, and pretraining epochs), we vary the value while holding all others fixed and measure downstream AP averaged across the five evaluation datasets with VELOX as the detector. Results are reported in Figure 8. Performance is stable across all three hyperparameters, with only minor variation around the best configuration. The largest drop occurs with the smallest neighborhood size $[3, 10]$, which limits the contextual information available to the teacher; values in the typical $[5, 20]$ to $[10, 40]$ range yield better performance.

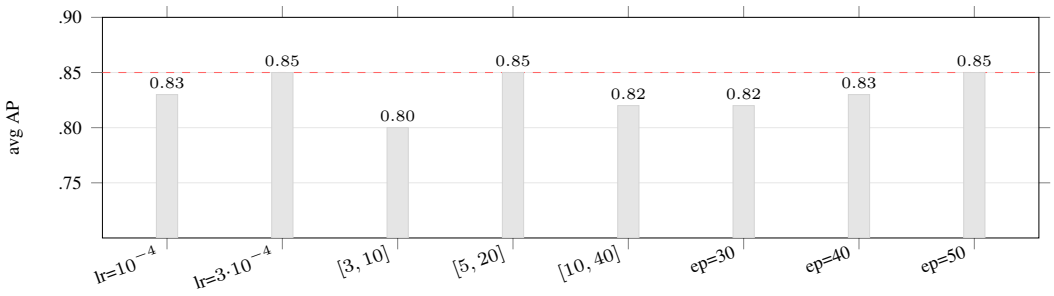

Figure 8: **Sensitivity of SPIDER to key hyperparameters.** Avg AP across the five evaluation datasets when varying learning rate, neighborhood size $[n_{\min}, n_{\max}]$, and pretraining epochs. The dashed red line at 0.85 marks the AP of the best configuration.

D Hyperparameter Selection

D.1 Encoder Pretraining

Table 5: Model and training hyperparameters.

Parameter	Value
Training	
Optimizer	AdamW (β_1=0.9, β_2=0.95, ϵ=10^{-8})
Weight decay	0.1
LR schedule	Cosine decay (10% linear warmup)
Gradient clip (max norm)	5.0
$P\times K$ batch	P=512 classes, K=4 entities/class

Encoder hyperparameter tuning. For SPIDER and all encoder baselines (Table 3), we tune hyperparameters once on our pretraining corpus and use the same configuration across all evaluation datasets. This protocol matches our transferability claim: a single checkpoint should work across operating systems without per-dataset adaptation. We select the configuration that maximizes k-NN accuracy ($k = 5$) over the entity semantic classes, which measures how well the embedding space separates entities by functional role. This is a self-consistency criterion that verifies SupCon training converges, not a proxy for downstream detection performance, which is never observed during selection. We allow a maximum runtime of 24 hours per run. For all encoder baselines, we tune embedding dimension over $[128, 256, 384]$, learning rate over $[10^{-4}, 3 \cdot 10^{-4}]$, and batch size over $[1024, 2048]$. The selected best learning rate is $3 \cdot 10^{-4}$ and the best batch size is 2048 across all encoders. For GPT-2 / Llama, batch size is constrained to $[256, 512]$ due to memory, and we tune the fine-tuning learning rate over $[10^{-5}, 3 \cdot 10^{-5}]$, selecting 10^{-5} and a batch size of 512; BERT / ModernBERT also use a best batch size of 512. Method-specific ranges and best embedding dimensions are reported in Table 6.

When fine-tuning SPIDER on each dataset (🔥 setting), we use a reduced learning rate of $3 \cdot 10^{-5}$ and 5 epochs over the training set, while keeping all hyperparameters fixed at the values selected above.

Training Hyperparameters. Table 5 reports the full training configuration used across experiments.

D.2 Downstream Detectors

For each detector (VELOX, ORTHRUS, FLASH, KAIROS), we start from the per-dataset tuned hyperparameters reported by Bilot et al. [2025] and perform an additional grid search over the shared ranges in Table 7.

E Tokenization Examples

We provide examples of tokenized entities in Table 8.

Table 6: Method-specific hyperparameter ranges and selected values for encoder pretraining.

Method	Hyperparameter	Range	Best
SPIDER	Embedding dimension	$[128, 256, 384]$	256
	BPE vocabulary size	[8000, 10000]	8000
	Neighborhood size $[n_{\min}, n_{\max}]$	$\{[5, 20], [10, 40]\}$	$[5, 20]$
Word2Vec	Embedding dimension	$[128, 256, 384]$	128
	Window size	$[3, 5]$	5
	Negative samples	$[3, 5]$	5
FastText	Embedding dimension	$[128, 256, 384]$	256
	Window size	$[3, 5]$	3
	Negative samples	$[3, 5]$	3
DeepWalk	Embedding dimension	$[128, 256, 384]$	256
	Walk length	$[5, 10]$	10
	Window size	$[5, 10]$	5
Node2Vec	Embedding dimension	$[128, 256, 384]$	256
	Walk length	$[5, 10]$	10
	p / q	$[0.5, 1]$	1 / 1
BERT / ModernBERT	Model size	[tiny, mini, med]	mini
	Walk length	$[5, 10]$	10
CyberGFM	Embedding dimension	$[128, 256, 384]$	256
	Walk length	$[5, 10]$	10
GAE	Embedding dimension	$[128, 256, 384]$	256
	Neighborhood size $[n_{\min}, n_{\max}]$	$\{[5, 20], [10, 40]\}$	$[5, 20]$
DGI	Embedding dimension	$[128, 256, 384]$	256
	Neighborhood size $[n_{\min}, n_{\max}]$	$\{[5, 20], [10, 40]\}$	$[5, 20]$
GraphMAE	Embedding dimension	$[128, 256, 384]$	256
	Mask rate	$[0.5, 0.75]$	0.5
	Replace rate	$[0.0, 0.1]$	0.1
	Neighborhood size $[n_{\min}, n_{\max}]$	$\{[5, 20], [10, 40]\}$	$[5, 20]$

Table 7: Additional grid search ranges for downstream detectors.

Scope	Hyperparameter	Range
All Detectors	Hidden dimension	$[64, 128, 256]$
	Learning rate	$[10^{-4}, 10^{-3}]$
	Batch size (snapshot duration)	[5min, 10min, 15min]

F Pretraining Details

Pretraining corpus. We built the pretraining corpus from provenance data spanning four operating systems (Linux, FreeBSD, Windows, and Android) by executing thousands of commands across activity categories typical of servers and PCs (e.g., web browsing, daemons, web servers, package management) instrumenting the systems to collect concise and clean provenance logs. Aggregate corpus statistics are reported in Table 9.

Redundancy reduction. Raw provenance data is highly redundant; we apply three levels of reduction: (i) we filter edges involving shared libraries, pseudo-filesystems (`/proc`, `/sys`, `/dev`), common configuration files, and other hub entities that carry no discriminative signal; (ii) when sampling each entity's 1-hop neighborhood, we deduplicate neighbors by (edge type, node type, label) and select via round-robin across edge types to maximize action diversity; and (iii) entities whose labels produce identical token sequences after tokenization are merged, unioning their behavioral signatures so that events differing only by hashes or random identifiers collapse into a single training identity. Overall, the pretraining set comprises 11.5M input entities after perfect deduplication, and 1.3M high-quality, diverse examples after post-tokenization deduplication. From the system activity, 532 unique behavioral signature triplets were extracted and used for training.

Table 8: Examples of domain-aware tokenization. Phase 1 applies domain-specific normalization (well-known directories, port categories, hash collapsing, flag preservation), and Phase 2 decomposes remaining rare words via BPE subword merges.

Raw label	Tokenized output
Process	
`/usr/sbin/sshd -D`	`[PROC] [/USR] [/SBIN] /sshd [-D]`
`/usr/bin/python3 /opt/app/server.py --port 8080`	`[PROC] [/USR] [/BIN] /python3 [/OPT] /app /server [EXT_py] [FLAG] port [NUM]`
File	
`/usr/lib/x86_64-linux-gnu/libssl.so.1.1`	`[FILE] [/USR] [/LIB] /x86 [SEP_U] 64 [SEP_D] linux [SEP_D] gnu /libssl [EXT_so] 1 1`
`/var/spool/postfix/incoming/D1F3A2B4C01`	`[FILE] [/VAR] /spool /postfix /incoming [HASH]`
Socket	
`203.0.113.45:49152` → `10.0.0.5:22`	`[SOCK] 203 0 113 45 [PORT_EPH] 10 0 0 5 [PORT_SSH]`
`192.168.5.100:48721` → `192.168.5.10:53`	`[SOCK] 192 168 5 100 [PORT_REG] 192 168 5 10 [PORT_DNS]`

`[SEP_D]` = `[SEP_DASH]`, `[SEP_U]` = `[SEP_UNDERSCORE]`. Domain tokens (bracketed) are fixed vocabulary; unbracketed words go through BPE subword decomposition. Hex strings $\geq$7 characters are collapsed to `[HASH]`; integers $>$255 to `[NUM]`; well-known ports to named tokens (e.g. `[PORT_SSH]`, `[PORT_DNS]`).

Table 9: Pretraining corpus statistics.

Nodes	Unique Nodes	Edges	Tokens	Avg Tok/Entity	% File	% Proc	% Sock
42.3M	11.5M	287.3M	353.3M	8.36	25.0	36.4	38.6

G Pretraining Evaluation

We monitor four embedding-quality metrics during pretraining (Figure 9), each computed on a held-out portion of the pretraining corpus using the entity semantic classes as ground-truth labels. **k-NN accuracy** ($k = 5$) measures how often an entity's k nearest neighbors in the embedding space share its class, capturing local cluster purity. **ARI** (Adjusted Rand Index) and **NMI** (Normalized Mutual Information) compare a k-means clustering of the embeddings against the ground-truth class assignments, measuring respectively pairwise agreement (corrected for chance) and mutual information (normalized to $[0, 1]$); both capture global clustering quality rather than local neighborhood structure. **mAP** (mean Average Precision) treats each class as a retrieval query and ranks all other entities by embedding similarity, rewarding embeddings that place same-class entities consistently above other-class ones across all ranks. All four metrics improve monotonically through training and plateau by the end of the schedule. The fact that all four agree on the trajectory (local (k-NN), global (ARI, NMI), and rank-based (mAP)) indicates that the embedding space improves uniformly across scales rather than overfitting to any single notion of cluster quality.

H Computational Cost

Training Time. Figure 10 reports the training times of the different encoders. SPIDER trains in 1h52, on par with the lightest word-embedding baselines (Word2Vec, FastText) and substantially faster than heavier graph-level methods and large language models. SPIDER parallelizes all its operations efficiently on GPU: the transformer, the graph attention operation and the MLP.

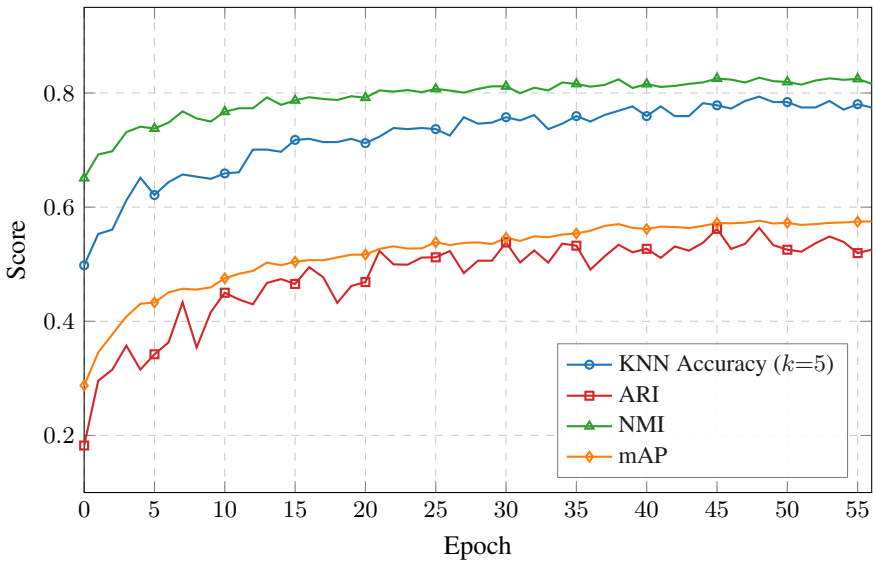

Figure 9: **Evaluation metrics during pretraining.**

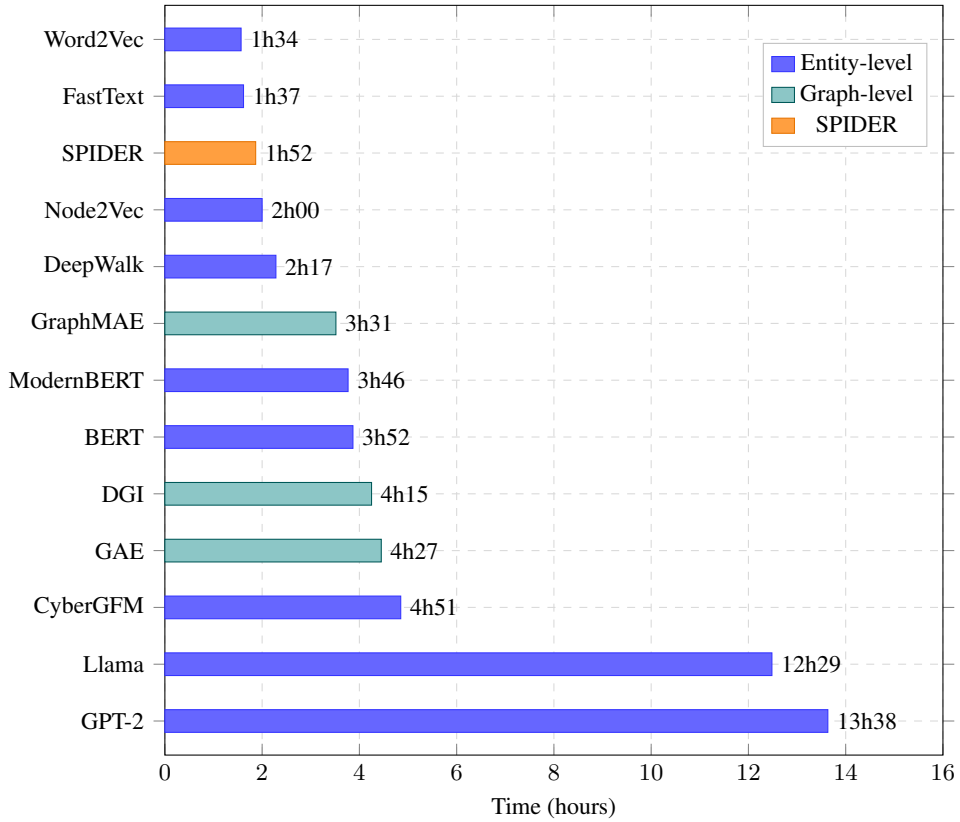

Figure 10: Training (or processing) time comparison across embedding and language models.

I Baseline Details

I.1 Detector Baselines

We provide additional details on the four detectors benchmarked in Section 5.1. All four systems share a similar architecture. SPIDER replaces only the text encoding stage, leaving the rest of each system's pipeline untouched. Table 10 summarizes the components.

Table 10: Pipeline components of the four detector baselines.

System	Text Encoding	Encoder	Decoder	Objective
FLASH	Word2Vec + pos. enc.	GraphSAGE	XGBoost	Node-type prediction
KAIROS	HFH	TGN + attention	4-layer MLP	Edge-type prediction
ORTHRUS	Word2Vec	TGN (no memory) + attention	3-layer MLP	Edge-type prediction
VELOX	Word2Vec	Linear	2-layer MLP	Edge-type prediction

FLASH [Rehman et al., 2024]. FLASH performs intrusion detection by computing node anomaly scores from text embeddings. It uses Word2Vec with positional encoding to featurize node attributes, encodes them via a GraphSAGE-based GNN, and decodes via XGBoost trained to predict node types. OOV tokens at inference are replaced with zero-vectors.

KAIROS [Cheng et al., 2023]. KAIROS captures temporal dynamics in dynamic provenance graphs and detects attacks at the time window level. Entity attributes are featurized via Hierarchical Feature Hashing (HFH), an inductive scheme that handles unseen tokens at inference. The encoder is a Temporal Graph Network (TGN) with graph attention, decoded by a 4-layer MLP trained to predict edge types; anomalies are flagged using a fixed threshold.

ORTHRUS [Jiang et al., 2025]. ORTHRUS targets node-level detection with high attribution quality. It featurizes entity types, paths, command lines, IPs and ports using Word2Vec, encodes with a memory-free TGN variant combined with graph attention, and decodes via a 3-layer MLP trained on edge type prediction. Detection combines a validation-based threshold with K-Means clustering on anomaly scores.

VELOX [Bilot et al., 2025]. VELOX shows that, after addressing common shortcomings in detector evaluation (data snooping, hyperparameter tuning, instability), a simple MLP over Word2Vec embeddings matches or exceeds the detection performance of more elaborate GNN-based detectors on eight of nine DARPA datasets, while being orders of magnitude lighter and enabling real-time inference. Because VELOX isolates the encoder to a single linear layer, the quality of its entity featurization propagates almost directly to detection performance, making it a particularly clean stress test for embedding methods. We use VELOX as our default downstream detector in Table 3 for this reason.

I.2 Encoder Baselines

We provide additional details on the encoder baselines compared in Section 5.2. All methods are pretrained on the same corpus (section 4) and evaluated as backbone encoders for VELOX [Bilot et al., 2025]. We compare their input granularity in training/inference in Table 11.

I.2.1 Graph-level Methods

These methods exploit the provenance graph at both training and inference, requiring the full graph to be stored and traversed at deployment time. For fair comparison, we use as text encoder the same transformer as the one in SPIDER. The resulting node embeddings are used in the downstream GNN.

GAE [Kipf and Welling, 2016] (Graph Autoencoder) learns node embeddings by reconstructing the adjacency matrix by predicting edges through negative sampling. We use graph attention layers [Shi et al., 2021] as downstream encoder to learn inductive features.

DGI [Veličković et al., 2018] (Deep Graph Infomax) maximizes mutual information between local node patches and a global graph summary by contrasting against corrupted versions of the graph. We use the same graph attention layers [Shi et al., 2021] as encoder.

Table 11: Input granularity of encoder baselines.

Method	Training	Inference
GAE, DGI, GraphMAE	graph	graph
Word2Vec, FastText, GPT-2, Llama 3.2	entity	entity
DeepWalk, Node2Vec, BERT, ModernBERT, CyberGFM	walk	entity
SPIDER	graph	entity

GraphMAE [Hou et al., 2022] is a masked graph autoencoder that randomly masks node features and reconstructs them via a GAT-based encoder-decoder. We use the original scaled cosine error loss with a re-mask decoding strategy.

I.2.2 Entity-level Text Methods

These methods compute embeddings from entity text alone, both at training and inference, and ignore graph structure entirely. They scale trivially to enterprise deployments but cannot capture neighborhood semantics.

Word2Vec [Mikolov et al., 2013] learns embeddings via skip-gram with negative sampling, treating each entity label as a sequence of tokens. It is the default featurizer for several recent detectors (VELOX, ORTHRUS, FLASH).

FastText [Joulin et al., 2016, 2017] extends Word2Vec by representing each token as a bag of character n-grams, providing partial robustness to out-of-vocabulary tokens through subword composition.

GPT-2 1.5B [Radford et al., 2019] is a 1.5B-parameter autoregressive transformer pretrained on web text. We fine-tune it on our pretraining corpus, then extract entity embeddings as the mean-pooled hidden state of the final layer.

Llama 3.2 1B [Grattafiori et al., 2024] is a 1B-parameter decoder-only transformer pretrained on a recent multilingual corpus. We follow the same fine-tuning and mean-pooling protocol as for GPT-2.

I.2.3 Entity-level Methods with Walks

These methods incorporate graph structure during training via random walks while producing embeddings that depend only on entity text at inference, matching SPIDER's deployment profile. The standard recipe treats each walk as a sentence in which every node is a token, and trains a text encoder on the resulting corpus. We adapt this recipe to provenance graphs by representing each walk as the concatenation of the tokenized labels of its nodes; the text encoder then learns co-occurrence patterns that reflect the causal structure of the graph. We finally use mean-pooling over the final layer to obtain entity embeddings. Walks impose a deterministic linear order on a causal path, in contrast to an unordered graph neighborhood: the latter can be fed to a GNN, which is permutation-invariant by construction, but not to a text encoder, which expects a sequence.

DeepWalk [Perozzi et al., 2014] samples uniform temporal random walks over the provenance graph and trains skip-gram on the resulting node sequences, producing node embeddings.

Node2Vec [Grover and Leskovec, 2016] extends DeepWalk with biased random walks controlled by return (p) and in-out (q) parameters, allowing the embedding to interpolate between BFS-like (structural roles) and DFS-like (community membership) neighborhoods.

BERT [Devlin et al., 2019] and **ModernBERT** [Warner et al., 2025] are bidirectional transformer encoders trained from scratch on the walks via masked language modeling. ModernBERT incorporates recent architectural improvements (rotary position embeddings, GeGLU activations, longer context).

CyberGFM [King et al., 2026] is a recent graph foundation model designed for lateral movement detection in enterprise networks, framing detection as next-token prediction over host-level graphs. We adapt it to the provenance setting by training on temporal walks over our pretraining corpus.

J Semantic Classes

We enumerate the semantic classes that structure the teacher's contrastive objective (section 3). We derive classes from the mapping $(\texttt{type}(v), \ell(v))$: process names, file-path prefixes, file extensions, and well-known port numbers are each mapped to a functional category. This information is readily available from package managers and well-defined standards (*e.g.*port assignments). We curate the mapping once at the prefix level (a few hundred entries) and apply it automatically to all entities in the pretraining corpus. As a result, no per-entity labeling is required, and the same mapping transfers across all four operating systems we evaluate.

We organize the classes into six groups.

Network services. `webserver`, `database`, `mail`, `ssh`, `proxy`, `vpn`, `fileshare`, `msgqueue`.

System and infrastructure processes. `init`, `cron`, `sysadmin`, `monitoring`, `pkgmgr`, `svchost`, `win_system`, `android_sys`.

User-facing and developer tools. `shell`, `browser`, `editor`, `office`, `runtime`, `compiler`, `build`, `cirunner`, `vcs`, `transfer`, `netscan`, `crypto`, `db_client`, `scripting`, `script`.

Files by role. `config`, `config_web`, `config_mail`, `config_vpn`, `config_security`, `config_monitoring`, `config_container`, `config_fs`, `log`, `log_web`, `log_db`, `log_audit`, `data`, `data_pkg`, `data_mail`, `data_container`, `data_monitoring`, `data_ntp`, `executable`, `binary`, `library`, `source`, `object`, `bytecode`, `package`, `archive`, `database_file`, `certificate`, `key_file`, `pubkey`.

User and system artifacts. `homedir`, `shellrc`, `user_cache`, `user_downloads`, `user_desktop`, `user_media`, `document`, `media`, `browser_data`, `tmp`, `spool`, `cache`, `procfs`, `sysfs`, `pipe`, `mount`, `app_data`, `progdir`, `progdata`, `prefetch`, `recycle`, `crashdump`, `unknown`.

Sockets. `port_http`, `port_https`, `port_ssh`, `port_dns`, `port_mail`, `port_ftp`, `port_dhcp`, `port_ldap`, `port_smb`, `port_netbios`, `port_winrm`, `port_mysql`, `port_postgres`, `port_mongodb`, `port_redis`, `port_elasticsearch`, `port_database`, `port_low`, `port_reg`, `port_eph`, `socket_file`, `unix_socket`.

We emphasize that this taxonomy serves a structural role in pretraining only: SPIDER's student encoder produces embeddings from entity text alone, with no class lookup at inference time. Empirically, removing class supervision substantially degrades downstream detection (Section 5.3), confirming that this structure carries useful signal even where boundaries between adjacent classes are imperfect.

Inference & Fine-tuning. At inference, only the student is used: given an entity, we tokenize its text as "`{type} {attribute}`" (e.g., `[PROC] /usr/bin/nginx -c`) and run a single forward pass. No semantic class, behavioral signature, or graph access is required, so the deployment pipeline is independent of the curated mapping and the pretraining vocabulary $\mathcal{B}$. For fine-tuning (continued pretraining on a target dataset), we reuse the same training objective and the behavioral signature vocabulary $\mathcal{B}$ fixed during initial pretraining.

K Valid Event Type Triplets

Table 12 enumerates the valid $(\texttt{src\_type}, \texttt{event\_type}, \texttt{dst\_type})$ triplets determined by the DARPA Transparent Computing data model.

L Additional Experiments

This appendix reports additional experiments supporting Section 5.3, section 6, and Section 5.5. Unless stated otherwise, all experiments follow the protocol of Table 3: VELOX as the downstream detector with frozen (❄) embeddings.

Table 12: Valid edge triplets used by SPIDER. Each row gives an allowed (source node type, event type, destination node type) relation.

Source Type	Event Type	Destination Type
PROC	EVENT_CLONE	PROC
PROC	EVENT_EXECUTE	FILE
PROC	EVENT_CONNECT	SOCK
PROC	EVENT_OPEN	FILE
PROC	EVENT_WRITE	FILE
PROC	EVENT_READ	FILE
PROC	EVENT_SENDTO	SOCK
PROC	EVENT_SENDMSG	SOCK
SOCK	EVENT_RECVFROM	PROC
SOCK	EVENT_RECVMSG	PROC

L.1 Leave-One-OS-Out Pretraining

To test transfer to operating systems never seen during pretraining, we pretrain SPIDER with all data from the target OS removed from the corpus and evaluate on that OS's datasets (*e.g.*no Windows data when evaluating on OpTC). Table 13 shows a modest degradation: most behavioral priors transfer across OSes thanks to the OS-agnostic event vocabulary, and the remaining gap reflects platform-specific entities (*e.g.*Windows registry paths, Android intents) that benefit from in-corpus exposure.

Table 13: **Leave-one-OS-out pretraining.** AP when the target OS is excluded from the pretraining corpus.

Pretraining corpus	CD-E3	TH-E3	TH-E5	CS-E5	OpTC	Avg
Full corpus	0.84	1.00	1.00	0.91	0.36	0.82
Target OS excluded	0.95	0.97	0.98	0.85	0.30	0.81

L.2 Frequency-Stratified Evaluation

To assess whether SPIDER works only for common, stable entities, we partition test entities by the number of occurrences of their label in the pretraining corpus, as common (> 100), rare (1–100), or unseen (0), and compute AP within each stratum. Table 14 shows that performance degrades gently with decreasing frequency rather than collapsing on unseen entities, confirming that SPIDER embeddings generalize inductively to rare and unseen labels. The "unseen, frozen" scenario of Section B.1 complements this analysis in the adversarial setting.

Table 14: **Frequency-stratified evaluation.** AP per stratum of entity frequency in the pretraining corpus.

Frequency	CD-E3	TH-E3	TH-E5	CS-E5	OpTC	Avg
Common (> 100)	0.99	1.00	1.00	0.93	0.42	0.89
Rare (1–100)	0.96	1.00	1.00	0.88	0.31	0.83
Unseen (0)	0.92	0.98	0.97	0.80	0.32	0.80

L.3 Teacher GNN Architecture

We use graph attention [Shi et al., 2021] in the teacher for two reasons. First, it natively conditions attention on learnable edge-type embeddings, and event types carry the core semantics of provenance: a WRITE from a process to a file means something entirely different from a READ between the same pair. Uniform aggregators such as GraphSAGE ignore edge types unless extended with relational machinery. Second, attention learns to down-weight residual high-degree hub neighbors that survive filtering. To validate this choice empirically, we replace the graph attention layer with R-GCN [Schlichtkrull et al., 2018], GraphSAGE [Hamilton et al., 2017], and GIN [Xu et al., 2019],

keeping everything else fixed. As shown in Table 15, graph attention and R-GCN, which both condition on edge types, outperform the type-agnostic alternatives.

Table 15: **Teacher GNN architecture.** AP when replacing the teacher's graph attention layer.

Teacher GNN	CD-E3	TH-E3	TH-E5	CS-E5	OpTC	Avg
Graph attention (ours)	0.84	1.00	1.00	0.91	0.36	0.82
R-GCN	0.94	1.00	1.00	0.87	0.30	0.82
GraphSAGE	0.88	0.96	0.98	0.78	0.12	0.74
GIN	0.85	0.94	0.96	0.75	0.10	0.72

L.4 GNN-to-MLP Distillation Baseline

We compare SPIDER against the closest existing distillation recipe, Graph-Less Neural Networks (GLNN) [Zhang et al., 2022], which distills a pretrained GNN teacher into a graph-free MLP student by matching the teacher's outputs on the nodes of the training graph. We adapt GLNN to our pretraining corpus and evaluate it under the same protocol as SPIDER (Table 16). GLNN trails SPIDER by 0.17 avg AP, with the largest gaps on CS-E5 (Android) and OpTC (Windows), where deployment-specific entities dominate and its node-bound distillation targets do not cover them.

Table 16: **Comparison with a GNN-to-MLP distillation baseline.**

Method	CD-E3	TH-E3	TH-E5	CS-E5	OpTC	Avg
GLNN (adapted)	0.82	0.93	0.88	0.55	0.22	0.68
SPIDER	0.84	1.00	1.00	0.91	0.36	0.82

L.5 Entity-Level Baselines with Domain Tokenization

Graph-level baselines in Table 3 already use the same text encoder and tokenizer as SPIDER (Section I.2), so they differ from SPIDER only in their training objective. To equalize domain knowledge for entity-level baselines as well, we retrain Word2Vec and FastText on the pretraining corpus tokenized with SPIDER's tokenizer (Section 3.1). Table 17 shows that the domain tokenizer contributes meaningful signal, but both methods still trail SPIDER by a wide margin, isolating the value of graph distillation and contrastive supervision beyond tokenization.

Table 17: **Entity-level baselines with SPIDER's tokenizer.**

Method	Tokenizer	CD-E3	TH-E3	TH-E5	CS-E5	OpTC	Avg
Word2Vec	native	0.70	0.77	0.96	0.14	0.08	0.53
Word2Vec	SPIDER's	0.78	0.82	1.00	0.42	0.10	0.62
FastText	native	0.60	0.80	0.12	0.89	0.26	0.53
FastText	SPIDER's	0.68	0.84	0.40	0.90	0.20	0.60
SPIDER	SPIDER's	0.84	1.00	1.00	0.91	0.36	0.82

L.6 Statistical Significance

Some datasets contain few attack nodes, so AP can be sensitive to the ranking of a handful of entities. We therefore run paired Wilcoxon signed-rank tests [Wilcoxon, 1945] over the 10 seeds comparing SPIDER against Word2Vec (native), both frozen with VELOX as the detector, and report the rank-biserial correlation as effect size (Table 18). Differences are significant with large effect sizes on CD-E3, TH-E3, CS-E5, and OpTC.

Table 18: **Paired Wilcoxon signed-rank tests** (10 seeds): SPIDER vs. Word2Vec (native), both frozen, with VELOX.

	CD-E3	TH-E3	TH-E5	CS-E5	OpTC
Wilcoxon p	0.002	0.002	0.002	0.002	0.004
Effect size (rank-biserial r)	1.00	1.00	1.00	1.00	0.90