NetVigil: Robust and Low-Cost Anomaly Detection for East-West Data Center Security

Kevin Hsieh[*1] Mike Wong[*2,1] Santiago Segarra[1,3] Sathiya Kumaran Mani[1]
Trevor Eberl[1] Anatoliy Panasyuk[1] Ravi Netravali[2] Ranveer Chandra[1] Srikanth Kandula[1]
[1]*Microsoft* [2]*Princeton University* [3]*Rice University*

Abstract– The growing number of breaches in data centers underscores an urgent need for more effective security. Traditional perimeter defense measures and static zero-trust approaches are unable to address the unique challenges that arise from the scale, complexity, and evolving nature of today's data center networks. To tackle these issues, we introduce NetVigil, a robust and cost-efficient anomaly detection system specifically designed for east-west traffic within data center networks. NetVigil adeptly extracts security-focused, graph-based features from network flow logs and employs domain-specific graph neural networks (GNNs) and contrastive learning techniques to strengthen its resilience against normal traffic variations and adversarial evasion strategies. Our evaluation, over various attack scenarios and traces from real-world production clusters, shows that NetVigil delivers significant improvements in accuracy, cost, and detection latency compared to state-of-the-art anomaly detection systems, providing a practical, supplementary security mechanism to protect the east-west traffic within data center networks.

1 Introduction

The modern era of digitalization has brought about unprecedented growth in data center networks, ushering in a period where the need for robust security measures is more critical than ever before. Recent high-profile breaches, such as the Equifax breach [1] and the SolarWinds attack [3, 82] have exposed vulnerabilities in data center network security and demonstrate catastrophic consequences that can arise from a lack of adequate protection. As cyber threats continue to evolve and grow in sophistication, there is an urgent need to develop innovative solutions to safeguard sensitive information and ensure the integrity of data center networks.

One such area that warrants particular attention is the security of east-west traffic within data centers. Traditional security measures have focused primarily on securing north-south traffic at the network perimeter, which has left internal systems susceptible to lateral movements and persistent threats from compromised nodes, SSH keys, or other credentials [73, 75]. Zero-trust architecture [44, 66] aims to mitigate these risks by securing east-west communication and data flows. However, current zero-trust solutions, such as micro-segmentation [40, 59, 70, 77], rely only on static access control rules and are ill-equipped to detect dynamic and unusual behaviors, leaving networks exposed to potential attacks and lateral movement.

Addressing the critical gap in data center network security necessitates the development of an effective and always-on network anomaly detector specifically tailored for east-west traffic. Although existing network anomaly detection solutions have achieved significant progress in handling north-south traffic [32, 47], they struggle to overcome the unique challenges associated with east-west traffic. First, the majority of these solutions require capturing and analyzing network packets, leading to a *cost* that becomes prohibitively high when implemented across all nodes. For example, a recent high-throughput malicious traffic detection system [31] demands at least 17 cores and 10 GB of memory per node to secure a 10 Gb NIC, resulting in an annual cost of six million dollars for an application comprising 1,000 nodes. Second, existing solutions are known for generating *false alarms* [43, 57], an issue further amplified by the dynamic nature of east-west traffic. Even a marginal false alarm rate can significantly escalate operational overhead at scale, causing security teams to inadvertently neglect genuine threats. Third, a multitude of solutions depend on labeled malicious datasets or signatures for training their detectors [22, 46, 51, 53], a strategy that is not only impractical at scale but also renders systems susceptible to novel zero-day attacks [23, 55].

Objectives and Techniques. We introduce NetVigil, a novel anomaly detection system explicitly designed for securing east-west traffic in large-scale networks. In light of the challenges and limitations associated with existing solutions, NetVigil is designed to fulfill three primary objectives: (a) guaranteeing cost-effectiveness when monitoring numerous nodes, (b) precisely identifying anomalous behaviors while emphasizing the reduction of false alarms, and (c) demonstrating robustness to normal traffic changes without depending on prior knowledge of malicious attacks.

NetVigil achieves these objectives with three core ideas:

(1) *Deriving security-focused graph features from flow summaries.* To ensure cost-efficiency, NetVigil leverages low-cost flow summaries, available at both network level (e.g., VPC Flow Logs [21] and NSG Flow Logs [54]) and service level

* Equal contribution.

(e.g., Calico flow logs [6] and Cilium/Hubble [9]). These loggers offer substantial cost savings over packet traces by logging only aggregated statistics. NetVigil adeptly extracts security-oriented graph features from these summaries, effectively compensating for the absence of packet-level information. This approach enables scalable monitoring of large networks without sacrificing anomaly detection accuracy.

(2) *Leveraging graph neural networks (GNNs) and domain-specialized contrastive learning for context-aware robust anomaly detection.* NetVigil employs GNNs to model complex relationships between nodes in the network. The key insight is that different nodes within a network (e.g., microservices) carry diverse contextual information. By integrating contextual information from adjacent nodes, GNNs can detect anomalous behaviors that might be overlooked by traditional solutions focusing on individual flows [31, 56]. To further strengthen the model's resilience against normal traffic fluctuations, NetVigil adopts graph contrastive learning [85] with domain-specialized data augmentation. This approach guides the model toward capturing meaningful representations of standard traffic patterns, enabling it to distinguish between benign and malicious behaviors with greater precision.

(3) *Adapting to temporal dynamics via smoothing and continuous retraining.* NetVigil addresses the evolving nature of network traffic by integrating a temporal loss that encourages similarity between embeddings of temporally-adjacent graphs. Moreover, NetVigil continuously retrains its model using recent clean logs by excluding anomalous flows. This approach keeps the model updated with network behavior, maintaining high detection accuracy over time.

Implementation and Evaluation. We build NetVigil as an end-to-end streaming data pipeline, continuously analyzing network flow summaries and dynamically updating its model. To evaluate its effectiveness, we design a new east-west security benchmark, Yatesbury, using a microservice demo application [11], generating a diverse array of live traces and simulating evasive attack scenarios. Our extensive evaluation, including the benchmark and week-long to month-long traces from production clusters, demonstrates that NetVigil significantly outperforms existing malicious traffic detectors [31,56]. We achieve an average AUC (area under the ROC Curve) improvement of 0.22 (up to 0.62) and reduce operational costs by 2.7 – 16.7× for our 16-VM deployment. We release our benchmark Yatesbury [1] to enable researchers to explore novel attack scenarios in a cloud environment and contribute innovative solutions in this crucial domain.

Contributions. We make the following contributions:

- We introduce a novel network anomaly detection architecture designed to secure east-west traffic within data centers. This architecture utilizes low-cost network flow logs, security-oriented graph features, and graph neural networks (GNNs) to achieve cost-effectiveness and robustness.

[1] https://github.com/microsoft/Yatesbury

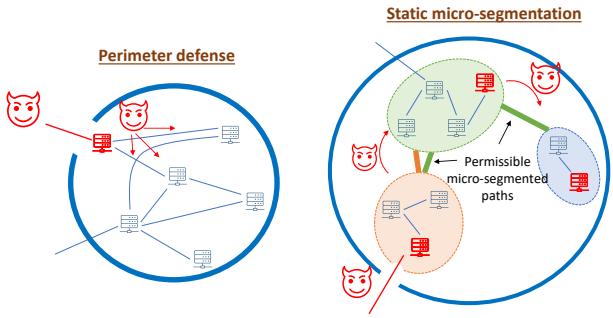


Figure 1: **Difference between perimeter defense and static micro-segmentation**

- We propose an innovative end-to-end training mechanism that combines graph representation learning, graph contrastive learning, and temporal smoothing. This approach enables the learning of east-west traffic dynamics for accurate anomaly detection.
- We build our solution alongside an east-west security benchmark tailored for cloud deployment, validating the performance of our proposed solution through various attack scenarios and long-term traces from two production clusters.

2 Background and Motivation

We briefly discuss key concepts and challenges in securing east-west traffic in data center networks. We focus on the importance of this task and the limitations of traditional security measures, and we provide an overview of network anomaly detection techniques specific to east-west traffic.

2.1 Securing East-West Traffic with Zero-Trust Solutions

Traditional security measures, such as firewalls [69] and intrusion detection systems (IDSes) [43], have primarily focused on securing north-south traffic (interactions between data center nodes and external systems) at the network perimeter, as illustrated in Figure 1. While these measures are essential, they often do not protect east-west traffic (communication between nodes within a data center). Traditional security measures tend to rely on static access control rules or attack signatures, which can be easily bypassed by attackers using compromised credentials or exploiting network vulnerabilities. Once inside the network, an attacker can perform lateral movement with relative ease. The limited visibility into the communication between nodes within the data center renders traditional security measures less effective in detecting and responding to emerging threats.

The zero-trust security model [44, 66] represents a paradigm shift in network security, aiming to address the limitations of traditional security measures and improve the protection of east-west traffic within data center networks.

IDS	Low compute overhead	Low network overhead	Easy management	Zero-day attack protection	Computation cost (CPU cores/100Gbps) or Licensing cost (US dollars)
Zeek (formerly Bro) [61]	✗	✗	✓	✗	400 [2]
Snort [65]	✗	✗	✗	✗	250 [88]
Pigasus [88]	✓	✗	✗	✗	5 + 1 FPGA [88]
Suricata [15]	✗	✗	✗	✗	53 + 1 SmartNIC [14]
Whisper [31]	✗	✓	✓	✓	170
VMware NSX [17]	✗	✓	✓	✗	$4,495/processor/yr [18]
Aviatrix DCF [5]	✗	✗	✓	✗ [4]	~$3,000/gateway/yr

Table 1: **Characteristics of popular IDSes. The characteristics of commercial options are obtained from their websites, which are subject to changes. The licensing costs associated with commercial solutions may not directly correspond to their computation costs.**

The main principles of zero-trust include: (a) least privilege access [68], where users and devices are granted the minimum access rights necessary to perform their tasks; (b) micro-segmentation [70, 77], which divides the network into smaller segments or zones to limit lateral movement (see Figure 1); and (c) continuous monitoring and validation of user and device behavior to ensure compliance with security policies.

Existing solutions [40, 59] primarily focus on micro-segmentation to limit the attack surface and the potential impact of a breach while preventing lateral movement of attackers. However, continuous monitoring and validation of user and device behavior remain a challenge due to the massive scale and dynamic nature of east-west traffic, requiring advanced techniques and tools to effectively detect and respond to anomalies in real time. Without this missing piece, an attack can still find a way to propagate from one node to another through permissible, micro-segmented paths, causing significant damage to critical infrastructure, data loss, or unauthorized access to sensitive information.

2.2 Challenges of Network Intrusion Detection Systems on East-West Traffic

A significant body of research [43, 48, 53] has been dedicated to the study of network intrusion detection systems (IDSes), which are tactically positioned at network choke points (e.g., routers or gateways) to safeguard network perimeters. Contemporary IDSes can be classified into signature-based and anomaly-based systems, each offering unique advantages. Signature-based systems efficiently detect known attack patterns using their distinct signatures, while anomaly-based systems focus on recognizing normal patterns to identify novel attacks. Although signature-based systems are effective for known attacks, the rise of zero-day attacks [23, 55] has highlighted the importance of anomaly-based systems. By employing deep learning techniques [32, 47], anomaly-based systems serve as a valuable complement to signature-based systems. Both systems have their merits, and a combination of the two can offer a more robust security solution. Table 1 summarizes key attributes of popular IDSes. Although these solutions work well for north-south traffic, their application to east-west traffic presents fundamental challenges.

Challenge 1: Excessive Compute Overhead in Network Packet Analysis. Past studies [16, 67] showed that only 17% of data center traffic was attributed to north-south traffic. The increasing adoption of micro-service architectures, cloud storage, and software-defined networks has since exacerbated the disparity between north-south and east-west traffic. Consequently, applying existing security solutions to east-west traffic would result in an unsustainable increase of already considerable costs. For example, Pigasus [88] employs Field-Programmable Gate Arrays (FPGAs) to substantially reduce operational expenses for rule-based IDSes. Nevertheless, it still requires five CPU cores and one FPGA to secure a 100 Gbps network. Commercial solutions like VMware NSX [17] tackle this problem by operating IDSes on every hypervisor. Although this method handles east-west traffic better, it results in substantial computational expenses for *all* data center nodes. The situation is no better for anomaly-based IDSes. Whisper [31], an efficient malicious traffic detection system, surpasses its predecessor Kitsune [56] by achieving a 100-fold increase in throughput. Despite this, securing a single 10 Gbps network necessitates 17 processing cores, potentially doubling or even tripling the application's operational costs.

The primary reason for the high overhead in existing solutions is their dependence on capturing and analyzing network packets, which offer rich and fine-grained information on network traffic. Nevertheless, as Table 1 shows, this approach is expensive and not scalable for large data center networks. A cost-effective alternative involves using network or service flow logs [6, 9, 21, 54], which utilize aggregation intervals to condense network packet telemetry into periodic 5-tuples and statistics. While service flow logs offer valuable application-level data, such as service names and request types, they often necessitate an agent in guest VMs or specialized environments like Kubernetes [24]. In this study, we focus on network flow logs due to their widespread availability in a cloud environment and consider integrating service flow logs in future work.

To evaluate this approach, we compare the effectiveness of two contemporary solutions, Whisper [31] and Kitsune [56], on network packet traces and network flow logs for similar set of attacks. As Figures 2 and 3 illustrate, applying existing solutions to network flow logs significantly compromises their accuracy due to the absence of packet-level information.

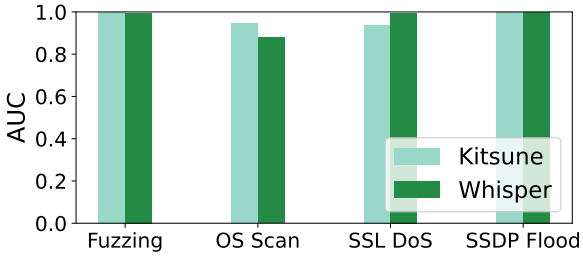


Figure 2: **AUC scores with packet traces**

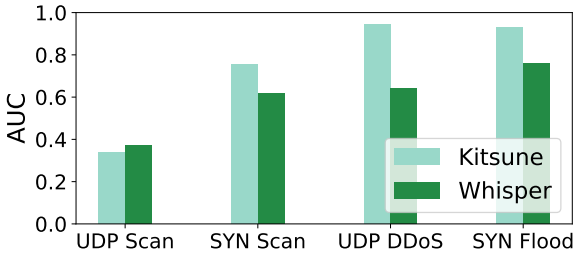


Figure 3: **AUC scores with flow traces**

Challenge 2: Complexity and Congestion in Networking. Numerous popular IDSes [15, 65, 88] are implemented by redirecting (or hair-pinning) traffic to IDS appliances. However, for east-west traffic, this method generates excessive network overheads, as traffic between VMs on the same machine or within the same rack must be rerouted to another cluster, potentially causing significant congestion in the network. Moreover, this approach adds complexity to network management, as hair-pinning must be executed for all communication paths within a data center, making routing substantially more complicated. Thus, commercial offerings [5] that adopt this approach primarily focus on egress traffic rather than encompassing all east-west traffic.

Challenge 3: Elevated False Alarms During Prolonged Deployment. Applying existing solutions to secure east-west traffic presents another challenge: an increased false alarm rate over extended deployment periods. As network traffic patterns evolve, security solutions must adapt to these changes to maintain high detection accuracy. However, many current network security systems struggle to keep up with the dynamic nature of network traffic, such as load variation and workload migration. This results in a high rate of false alarms.

We conduct an evaluation of Whisper [31] using two long-term traces from our first-party production clusters (see Section 6.5 for more detailed information). We find that there is a substantial increase in false alarms over time, while the continuous daily retraining of these solutions yields only marginal improvements (not shown in plot). These false alarms not only utilize significant resources for investigation but also undermine trust in the system, potentially resulting in the disregard of authentic threats.

Summary. The absence of a cost-efficient, robust and effective intrusion detection solution is a crucial impediment in securing east-west traffic. A comprehensive solution must: (a) ensure processing efficiency by eliminating dependence on all network packets, (b) avoid incurring network congestion or routing complexity, (c) proactively adapt to normal traffic fluctuations and (d) demonstrate high efficacy on previously unknown but malicious occurrences. These considerations are the foundation of our design requirements for NetVigil.

3 Overview of NetVigil

We present a novel anomaly-based intrusion detection system, NetVigil, that is explicitly designed to secure east-west data center networks with cost efficiency and robustness against normal traffic fluctuations. NetVigil achieves low operational costs by extracting security-oriented graph features from network flow logs, effectively eliminating the need for fine-grained yet costly network packet traces. The insight of NetVigil lies in the fact that network nodes within a data center typically provide specific functionalities (e.g., microservices, storage, databases, etc.), and by employing graph neural networks (GNNs), we can learn contextual information to enhance anomaly detection accuracy. Furthermore, our system incorporates graph contrastive learning and temporal smoothing techniques to achieve high detection accuracy while maintaining low false alarm rates. Figure 4 provides an overview of the NetVigil architecture.

Inference Time. During the inference phase (depicted on the left side of Figure 4), cloud resources such as virtual machines (VMs) and compute clusters continuously generate network flow logs [6, 9, 21, 54] at intervals ranging from tens of seconds to minutes. These data streams are processed by our *security graph feature extractor* ($\mathbf{I}_1$ in Figure 4, more details in Section 4.1), which groups network flow logs based on their IP addresses, extracts crucial features, and transforms the results into a featurized communication graph. In this graph, each node represents an IP address, and each edge summarizes all flows between respective IP pairs. NetVigil subsequently feeds this featurized communication graph into the continuously trained *GNN autoencoder* ($\mathbf{I}_2$ in Figure 4) to compute anomaly scores for each edge. Edges identified as potentially anomalous, along with the corresponding communication graphs and network flow logs, are then forwarded to the security team for further investigation.

Training Time. At each retraining interval, which typically spans hours or days based on network dynamics, NetVigil gathers clean communication graphs from the inference phase (i.e., excluding anomalous nodes and edges detected by the model) to retrain the GNN autoencoder (depicted on the right side of Figure 4). The *GNN encoder* ($\mathbf{T}_1$ in Figure 4) learns to compress the features of each edge and its incident nodes into an embedding space, allowing the *GNN decoder* ($\mathbf{T}_2$ in Figure 4) to reconstruct these features with minimal reconstruction loss ($\mathbf{L}_1$ in Figure 4). Section 4.2 provides more details.

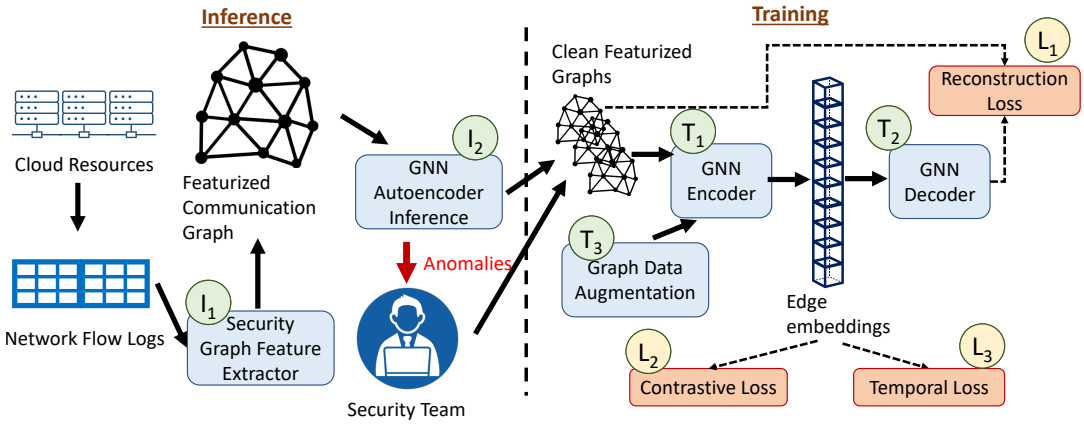


Figure 4: **Overview of NetVigil**

To enhance robustness, we design a *graph data augmentation* module ($\mathbf{T}_3$ in Figure 4) and train the system to encourage similarity between the embeddings of original and augmented graphs by minimizing the contrastive loss ($\mathbf{L}_2$, more details in Section 4.3). Furthermore, we introduce a temporal loss ($\mathbf{L}_3$, more details in Section 4.4) to promote embedding similarity between temporally-adjacent graphs. The entire training process is conducted end-to-end, resulting in an up-to-date GNN autoencoder for the subsequent inference phase.

Feature	Statistics
Number of transmitted packets Number of received packets Total received bytes Total transmitted bytes	min, max, mean, sum, std
Number of TCP flows Number of UDP flows Number of local unseen ports Number of global unseen ports Number of ports	count

Table 2: **Features obtained for each distinct IP pair**

4 Design Details of NetVigil

We provide a comprehensive discussion of the key modules within NetVigil and explain how these components collaborate to fulfill our design objectives. We discuss the security-oriented graph feature extractor (Section 4.1), graph representation learning (Section 4.2), domain-specialized graph contrastive learning (Section 4.3), and temporal smoothing techniques (Section 4.4) employed by NetVigil to ensure a cost-effective, accurate, and robust anomaly detection system for east-west data center networks.

4.1 Security Graph Feature Extractor

Our security graph feature extractor aims to gather security-oriented features from network flow logs, prioritizing cost efficiency while maintaining essential information for downstream anomaly detection. Within each aggregation interval (e.g., one minute), network flow logs typically contain the following information: (a) 5-tuple data, encompassing protocol, source and destination IP addresses, and source and destination ports, (b) the number of transmitted and received packets, and (c) the volume of transmitted and received bytes.

Our graph feature extractor operates at the IP address level instead of the network flow (IP and port) level for two primary reasons. First, operating at the network flow level leads to much larger graphs and increases the burden on both inference and training processes. For instance, we observe orders of magnitude increases in the number of nodes and edges in our production traces when constructing communication graphs at the network flow level (139k nodes and 115k edges) compared to the IP level (300 nodes and 10-20k edges). Furthermore, aggregating at the IP address level facilitates the identification of correlations between flows associated with the same IP address (e.g., a notable increase in the number of flows or usage of different ports). This, in turn, simplifies the detection of anomalous attacks, such as port scanning.

Our graph feature extractor offers additional operational cost reductions through a tunable *detection window* (e.g., two or three minutes). By generating a single communication graph per detection window, the feature extractor effectively balances detection latency with cost efficiency. Although a larger detection window might marginally impact detection accuracy for evasive attacks, this approach enables network operators to harmonize the requirement for prompt detection with limited resource constraints. Consequently, it becomes a suitable solution for large-scale network monitoring and security applications.

Table 2 summarizes the features we obtain for each distinct IP pair. We exclude ephemeral ports from the port-related features, as they do not provide learnable information. In addition to common features like the number of packets and bytes, we monitor critical security-oriented features such as the number of flows and unseen ports, which strongly indicate unusual occurrences. We maintain a record of globally observed ports

across all flows, as well as the locally observed ports for each IP pair within the training dataset. By maintaining port information as statistics, our feature extractor can function at the IP address level without sacrificing essential information. This approach allows us to capture crucial correlations among distinct flows associated with the same IP address while maintaining scalability and facilitating the detection of anomalous attacks involving multiple flows.

4.2 Graph Representation Learning

In order to discover relationships within the communication graph, NetVigil employs Graph Neural Networks (GNNs). These advanced machine learning models are specifically designed to analyze and extract patterns from complex graph-structured data. The key insight is that nodes within our communication graph correspond to various roles in an application and, as a result, each node and its neighbors exhibit a particular pattern over time. This contextual information enables NetVigil to identify anomalous behaviors that may seem normal if analyzed individually (e.g., C&C communication patterns or DNS amplification) without requiring the costly processing of granular packet-level information such as packet sizes, arrival times, and payloads. The ability to forego detailed packet-level data stems from the sufficiency of flow-level information for communication graph construction, capturing many key characteristics that distinguish a malicious flow from a normal one, including traffic volume, flow count, and interactions with various ports and IPs. Similar to prior work, E-GraphSage [51], we first aggregate edge features on each node as contextual information, and then concatenate the original edge features with this contextual information as the input to our edge encoder. Since E-GraphSage relies on supervised training, which is not practical at scale, we build our solution using an autoencoder by mapping each concatenated edge feature into a compressed embedding space.

Algorithm 1 presents the pseudocode for our GNN autoencoder. Lines 1–3 aggregate edge features using an AGG function, which can be mean, median, or element-wise pooling. Line 5 concatenates the aggregated contextual information with the original edge features, and Lines 6–8 encode the concatenated edge features into edge embeddings (Line 9). Lines 11-19 decode the embeddings back to the original edge features, and the reconstruction loss between e_{uv} and $\tilde{e}_{uv}$ corresponds to the $\mathbf{L}_1$ in Figure 4. Formally, for a (mini)batch $\mathcal{B}(\mathcal{G})$ of graphs, we have that

$$\mathbf{L}_1 = \frac{1}{\sum_{i|\mathcal{G}_i\in\mathcal{B}(\mathcal{G})}|\mathbb{E}_i|}\sum_{i|\mathcal{G}_i\in\mathcal{B}(\mathcal{G})}\sum_{(u,v)\in\mathbb{E}_i}\|e_{uv}-\tilde{e}_{uv}\|^2, \quad (1)$$

where $\mathbb{E}_i$ corresponds to the edge set of $\mathcal{G}_i$.

It is worth noting that although we employ simple graph convolutional networks (GCNs) [45] in Algorithm 1, the GNN architecture can be interchangeable, provided that the encoder takes into account both original edge features and aggregated neighboring features. While our experiments do not show a significant accuracy gain from using additional convolutional layers or alternate GNN architectures, some domain-specific GNN architectures may still perform better. We leave the exploration of domain-specific architectures for future work.

Algorithm 1 GNN Autoencoder for Edge Embedding

Input: Graph $\mathcal{G}(\mathbb{V},\mathbb{E})$
Input: Edge features $e_{uv}, \forall uv \in \mathbb{E}$
Input: Number of autoencoder layers L
Input: Encoder/decoder weights $W_E^l, W_D^l, \forall l \in 1,...,L$
1: **for** $v \in \mathbb{V}$ **do** ▷ Aggregate neighboring edge features
2: $h_v \leftarrow \text{AGG}(e_{uv}, \forall u \in N(v), (u,v) \in \mathbb{E})$
3: **end for**
4: **for** $(u,v) \in \mathbb{E}$ **do**
5: $h_{uv}^0 \leftarrow \text{CONCAT}(h_u, e_{uv})$
6: **for** $l \in 1,...,L$ **do** ▷ Edge Encoder
7: $h_{uv}^l \leftarrow \sigma(W_E^l \cdot h_{uv}^{l-1})$
8: **end for**
9: $z_{uv} = h_{uv}^L$ ▷ Edge embedding
10: **end for**
11: **for** $v \in \mathbb{V}$ **do** ▷ Broadcast edge embedding
12: $h_v \leftarrow \text{AGG}(z_{uv}, \forall u \in N(v), (u,v) \in \mathbb{E})$
13: **end for**
14: **for** $(u,v) \in \mathbb{E}$ **do**
15: $h_{e_{uv}}^0 \leftarrow \text{CONCAT}(h_u, z_{uv})$
16: **for** $l \in 1,...,L$ **do** ▷ Edge Decoder
17: $h_{uv}^l \leftarrow \sigma(W_D^l \cdot h_{uv}^{l-1})$
18: **end for**
19: $\tilde{e}_{uv} = h_{uv}^L$ ▷ Reconstructed edge features
20: **end for**

4.3 Domain-Specific Contrastive Learning

One of the primary challenges faced by existing network anomaly detectors is the generation of numerous false alarms for normal changes that were not encountered during the training process. A common approach to address this issue involves curating an extensive long-term dataset for training, with the expectation that all normal behaviors will be encompassed within this dataset. However, this approach is not scalable for east-west traffic, as they frequently experience normal changes, such as configuration updates, load variations, and node failures. To tackle this challenge, we employ graph contrastive learning [50], which augments the training data with general and domain-specific perturbations to enhance the model's generality. This approach allows the model to better accommodate and adapt to the dynamic nature of network traffic, thereby reducing the incidence of false alarms while accurately detecting genuine anomalies.

We find that most normal traffic fluctuations arise from (1) not all edges appearing consistently, (2) traffic volume varying over time, and (3) noisy behavior from non-application edges. Based on these observations, we employ the following data

augmentation strategies to enhance the model performance:

- Randomly removing edges and nodes: By presenting subgraphs to the model, the system gains the ability to more effectively analyze network communication patterns, attributable to the simplified structure and reduced noise.
- Adding noise to edge features: Edge features, such as the number of packets and the volume of transmitted/received bytes, are perturbed to test the model's robustness against variations in feature values.
- Removing non-application edges: It has been observed that application traffic within the network exhibits greater predictability compared to inter-service communications. Consequently, by removing nodes and edges unrelated to the application running within the network, the model can better learn and recognize application-level communication patterns, thereby enhancing its robustness and reducing the occurrence of false positives.

Formally, during training, a minibatch $\mathcal{B}(\mathcal{G})$ of graphs is randomly sampled. For every graph $\mathcal{G} \in \mathcal{B}(\mathcal{G})$, we generate two augmented versions $\mathcal{G}^{(1)}$ and $\mathcal{G}^{(2)}$ by randomly selecting two of the data augmentation strategies mentioned above. We denote the corresponding embeddings of an edge $uv \in \mathbb{E}$ by $z_{uv}^{(1)}$ and $z_{uv}^{(2)}$; see Line 9 in Algorithm 1. Recalling that the cosine similarity between two vectors x and y is given by $\cos(x,y) = x^\top y/(\|x\|\|y\|)$, we define the contrastive loss of a given edge uv as [85]

$$\ell_{uv} = -\log\left(\frac{\exp(2\cos(z_{uv}^{(1)}, z_{uv}^{(2)}))}{\sum_{u'v'}\exp(2\cos(z_{uv}^{(1)}, z_{u'v'}^{(2)}))}\right), \qquad (2)$$

where the negative edges $u'v'$ are randomly selected from augmented versions of other graphs in the minibatch. Notice that minimizing ℓ_{uv} promotes $z_{uv}^{(1)}$ and $z_{uv}^{(2)}$ to be similar, i.e., the embeddings corresponding to the same edge for two different augmented versions should be close to each other. Moreover, minimizing ℓ_{uv} also promotes $z_{uv}^{(1)}$ and $z_{u'v'}^{(2)}$ (the embeddings of different edges in augmented versions of different graphs) to be different from each other. In this way, we avoid the collapse of different embeddings into a common representation and encourage the full utilization of the embedding space. Our contrastive loss ($\mathbf{L}_2$ in Figure 4) is given by the average value of ℓ_{uv} over all edges in the minibatch

$$\mathbf{L}_2 = \frac{1}{\sum_{i|\mathcal{G}_i\in\mathcal{B}(\mathcal{G})}|\mathbb{E}_i|}\sum_{i|\mathcal{G}_i\in\mathcal{B}(\mathcal{G})}\sum_{uv\in\mathbb{E}_i}\ell_{uv}, \qquad (3)$$

where $\mathbb{E}_i$ corresponds to the edge set of $\mathcal{G}_i$. As defined, the loss $\mathbf{L}_2$ depends on the randomly selected augmentation strategies to compute the contrastive pairs of every graph. During training, we randomly choose new augmentation strategies for each minibatch, ensuring every gradient step is based on new contrastive pairs, thereby promoting generalization.

4.4 Temporal Smoothing and Continuous Retraining

Another crucial aspect that our model seeks to capture is temporal dynamics. Through our analysis of several traces from production clusters, we observe that network traffic within a short time window (e.g., minutes) tends to exhibit similarity, while patterns can undergo significant changes over longer periods (e.g., hours or days). This observation aligns with the understanding that major network traffic changes are typically driven by rare events (e.g., failures), periodicity (e.g., time of day), or application changes (e.g., code updates), which do not generally occur within short time frames.

We incorporate these temporal dynamics with a two-fold strategy. First, we define a temporal loss ($\mathbf{L}_3$ in Figure 4) during training to encourage embedding similarity between temporal adjacent graphs. For every pair of temporally adjacent graphs $\mathcal{G}^t$ and $\mathcal{G}^{t+1}$, we minimize the norm of the difference between consecutive embeddings of the same edge. More precisely, we have that

$$\mathbf{L}_3 = \frac{1}{\sum_t|\mathbb{E}_t\cap\mathbb{E}_{t+1}|}\sum_t\sum_{uv|uv\in\mathbb{E}_t\cap\mathbb{E}_{t+1}}\|z_{uv}^t - z_{uv}^{t+1}\|^2, \qquad (4)$$

where $\mathbb{E}_t$ denotes the edge set of $\mathcal{G}^t$. By minimizing $\mathbf{L}_3$, we promote the embeddings of the same edge in two consecutive time steps to be close to each other. Notice that, as a result of the dynamic nature of our graph, it might be that an edge uv that exists at time t is no longer present at time $t+1$. Hence, in (4), we account for this by only considering edges that belong to the intersection of two consecutive edge sets.

Second, we employ a periodic training procedure (e.g., hours or days) to update the model with the most recent traffic patterns. In each retraining window, we compile clean communication graphs (i.e., nodes and edges without any potential anomalies) from the inference phase (left side of Figure 4), as well as the false alarms cleared by the security team, to form the training set. Subsequently, we retrain the model by minimizing the composite loss, $\mathcal{L}$, which combines the reconstruction loss $\mathbf{L}_1$, contrastive loss $\mathbf{L}_2$, and temporal loss $\mathbf{L}_3$. The composite loss is defined as $\mathcal{L} = \mathbf{L}_1 + \alpha\mathbf{L}_2 + \beta\mathbf{L}_3$, where α and β are hyperparameters that trade off the relative importance of the different losses. This approach helps the model stay up-to-date and can effectively detect genuine anomalies and adapt to fluctuations in network traffic patterns.

5 Benchmarks and datasets

Datasets and network traces for intrusion and anomaly detection are available, but they primarily consist of packet traces. As stated in Section 2.2, conducting inference for each packet leads to considerable computational burdens, particularly in east-west traffic where there is a larger volume of network traffic. In light of this, we collect flow-level logs that summarize

network traffic. We deploy a 16-VM scale set on Microsoft Azure, activate Azure Network Watcher, and enable network security group (NSG) flow logging [54]. NSG logs record essential flow-level information such as 5-tuple, timestamp, number of bytes and packets transmitted, similar to other commercial flow logging offerings (e.g., [21]). For our 16-VM scale set, we deploy a web-based e-commerce app [11] that allows users to browse clothing items, add them to the cart, and complete purchases. Our application consists of 11 microservices for components such as ad generation, product catalog, payment, and product cart, as well as a load generator to send GET and POST requests, simulating user behavior like viewing items, setting currency, adding items to the cart, and submitting payment information.

On top of this setup, we develop and introduce a new benchmark, Yatesbury [1], designed to evaluate the performance and efficiency of anomaly detectors in east-west network traffic. We evaluate NetVigil with 13 distinct attacks that accurately represent a variety of malicious behaviors. Table 3 enumerates these attacks, each of which involves one or more compromised malicious nodes. Each trace lasts for 1-2 hours, and we label each trace for each 2-minute window and mark a connection as anomalous if any malicious traffic is sent or received between the two nodes. Malicious nodes communicate with each other and send malicious network traffic to the benign nodes. We utilize traditional network attacks that encompass various port scan methods, including traditional exhaustive port scanning, distributed scanning of multiple targets, stealth scanning using SYN packets to bypass firewalls, and accelerated scanning using UDP packets. Other traditional attacks we implement are DoS attacks, such as SYN flooding, which inundate victims with SYN requests, DDoS attacks involving multiple attackers, and DDoS attacks using UDP packets.

Furthermore, we incorporate attacks featuring more intricate communication patterns, which better represent malicious activity in east-west traffic. For these attacks, analyzing each flow in isolation (as done in traditional IDSes) is less effective, as it does not provide a holistic and comprehensive view of the network. We first employ Infection Monkey [8], an open-source breach and attack simulator for evaluating data center resiliency to perimeter breaches and internal host infection. It supports a wide range of different features such as port scanning, credential exploitation, and lateral movement to infect hosts. We also incorporate C&C communication patterns, which consist of a C&C server sending file updates, periodic heartbeat messages, and commands to control compromised hosts. These patterns are indicative of the communication observed in high-profile data breaches [1,3]. Further, we employ DNS amplification attacks, where multiple attackers send DNS requests to a DNS server and direct the responses to the victims. The DNS requests are crafted so that the responses are much larger in size to overwhelm the target machine.

6 Evaluation

6.1 Methodology

Benchmarks and datasets. We evaluate NetVigil on Yatesbury with our demo microservice application along with live production traces from our two first-party compute clusters.

Implementation. We implement NetVigil as an end-to-end data streaming pipeline using 1,400 lines of Python code. The inference pipeline extracts featurized communication graphs utilizing NetworkX [10] and pandas [12] libraries, while the training pipeline is built on PyTorch [13] and the Deep Graph Library (DGL) [7].

Baselines. We compare NetVigil against two state-of-the-art anomaly-based IDSes: Kitsune [56] and Whisper [31]. To evaluate Kitsune on our datasets, we modify its autoencoders to ingest flow-level features. Due to Kitsune's slow runtime, we implement several optimizations to reduce unnecessary computation that improves inference time by 5–10×; we refer to this optimized version of Kitsune as Kitsune+. Whisper's frequency domain analysis necessitates packet-level traces. Modifying it to utilize connection-level traces is challenging, as it performs frequency domain analysis on individual packets within each flow. Using only a single data point (aggregated flow-level statistics) would render it ineffective. To make Whisper compatible with flow logs, we utilize aggregated flow-level statistics to convert flow logs into packet traces. To understand the difference between flow and packet traces, we also carry out additional evaluations using packet traces for both Kitsune+ and Whisper.

Metrics. To evaluate NetVigil, we use the area under the ROC curve (AUC) as our primary metric, along with the true positive rate (TPR) and false positive rate (FPR). Importantly, AUC provides a measure of how well the detector can distinguish between the positive and negative classes, across all possible threshold settings. TPR and FPR are also crucial because the goal is to detect as many anomalies as possible while minimizing false alarms, which, as mentioned in Section 2, can significantly lower trust in an anomaly detector and is a fundamental challenge due to the dynamism of network traffic. To get these numbers, we select the threshold that maximizes $(TPR - FPR)$. Additionally, we compare latencies in running each anomaly detector. All latency experiments were run on a single 36-core, 72-hyperthread, 256-GB RAM machine (Intel(R) Xeon(R) Gold 5220).

6.2 Overall Results

We first compare the detection accuracy of NetVigil with our baselines, Kitsune+ and Whisper. For all attacks except one, NetVigil yields significantly higher performance over the baselines with AUC scores ranging from 0.6400 to 1.000, resulting in AUC improvements of up to 0.6591 over Kitsune+ and up

Attack	Description	# flows	Ratio malicious
Vertical Port Scan	Run an exhaustive scan of open ports	1429	0.0265
SYN Flood	DoS attack where connections are rapidly initialized but not completed	2817	0.0184
SYN Flood DDoS	DoS attack where connections are rapidly initialized but not completed (multiple attackers)	2437	0.0439
UDP DDoS	DoS attack with UDP packets (multiple attackers)	1473	0.0081
Distributed Stealth Port Scan	Run a targeted stealth scan of several key ports across many nodes with SYN packets	4069	0.0058
Distributed Port Scan	Run a targeted scan of several key ports across many nodes	4054	0.0051
Distributed UDP Port Scan	Run a targeted stealth scan of several key across many nodes with UDP packets	4319	0.0050
Infection Monkey 1	Scans key ports and launches network exploits	2768	0.0122
Infection Monkey 2	Scans key ports and launches network exploits (target limited number of hosts)	1490	0.0107
Infection Monkey 3	Scans key ports and launches network exploits (mount limited number of exploits)	4677	0.0027
C&C communication	Compromised nodes receive commands, heartbeats, and file updates from C&C server	2163	0.0254
DNS amplification	Attackers send DNS requests and direct responses to victim	4410	0.0825

Table 3: **Attack datasets**

	Kitsune+			Whisper			NetVigil		
	AUC	**TPR**	**FPR**	**AUC**	**TPR**	**FPR**	**AUC**	**TPR**	**FPR**
Moderate									
Vertical Port Scan	0.9300	0.8684	0.0057	0.9049	**0.9736**	0.1315	**0.9843**	0.9473	**0.0000**
SYN Flood	0.9322	0.8653	0.0014	0.7609	0.7307	0.1414	**1.0000**	**1.0000**	**0.0000**
Medium									
SYN Flood DDoS	0.9455	0.8971	0.0141	0.9148	0.9719	0.1283	**1.0000**	**1.0000**	**0.0000**
UDP DDoS	0.8676	0.7500	0.0199	0.6403	0.4166	0.1457	**0.9998**	**1.0000**	**0.0000**
Distributed Port Scan	0.4059	0.0952	0.0300	0.3961	0.0476	0.0329	**0.9968**	**0.9523**	**0.0000**
Distributed Stealth Port Scan	0.7542	0.6666	0.0758	0.6186	0.4166	0.1070	**0.9892**	**0.8333**	**0.0000**
Distributed UDP Port Scan	0.3367	0.0000	0.0281	0.3732	0.0000	0.4449	**0.9958**	**0.9545**	**0.0183**
Difficult									
Infection Monkey 1	0.5586	0.1176	**0.0003**	0.4395	0.0588	0.0190	**0.9997**	**1.0000**	0.0029
Infection Monkey 2	0.7497	0.5000	**0.0006**	0.4396	-	-	**0.9997**	**1.0000**	0.0033
Infection Monkey 3	0.5000	-	-	0.5000	-	-	**0.9998**	**1.0000**	**0.0006**
C&C communication	0.6347	0.4727	0.1480	0.5000	-	-	**0.9301**	**0.7636**	**0.0896**
DNS amplification	0.3962	0.0000	**0.0244**	0.8149	**0.8928**	0.2913	**0.8915**	0.3736	0.0692
SQL injection	**0.8531**	**1.0000**	**0.2237**	0.0900	-	-	0.6400	0.6428	0.2648
Unauthorized DB access	0.5976	**0.7500**	0.4720	0.7214	0.5833	0.0275	**0.8000**	0.7083	**0.1732**

Table 4: **Comparison of Kitsune+, Whisper, and NetVigil for various attacks.**

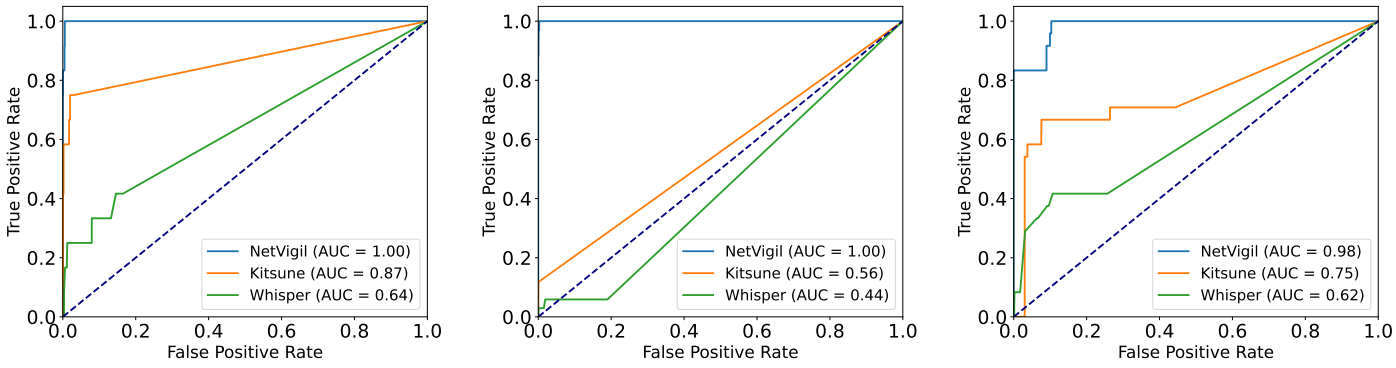
Attack	Kitsune+	Whisper	NetVigil
Vertical Port Scan	0.9817	0.9876	**0.9843**
UDP DDoS	0.9974	0.6414	**0.9998**
Dist. Stealth Port Scan	0.7267	0.6487	**0.9892**
Infection Monkey 1	0.8100	0.6188	**0.9997**
DNS Amplification	0.6759	0.8247	**0.8915**

Table 5: **AUC scores of Kitsune+ and Whisper using packet-level traces and NetVigil with flow-level logs for various attacks.**

to 0.6226 over Whisper. Table 4 presents the overall results for AUC, TPR, and FPR. Crucially, we observe that NetVigil outperforms the baselines because of two factors: (1) our novel security-centric feature extractor that effectively identifies lower-level malicious traffic characteristics in each connection that adversaries employ to fly under the radar, and (2) our use of graphs and a GNN architecture to obtain a holistic and comprehensive view of network behavioral patterns across many nodes.

Illustrating the efficacy of our feature selection approach, NetVigil exhibits strong performance in identifying DDoS attacks and vertical port scanning, achieving an AUC greater than 0.98 and an FPR approaching 0.0 for SYN Flood, SYN Flood DDoS, UDP DDoS, and Vertical Port Scanning. The extraction of packet- and connection-level statistics facilitates the detection of abnormal communications, such as substantial quantity of initiated connections in the SYN Flood scenario and the packet volume in other DDoS attacks.

Our GNN architecture excels in detecting reconnaissance patterns that span multiple nodes, an area where Kitsune+ and Whisper baselines demonstrate subpar performance. For Distributed Port Scan, Distributed Stealth Port Scan, and Distributed UDP Port Scan, Kitsune+ and Whisper yield AUC scores of 0.4059, 0.7542, 0.3367 and 0.3961, 0.6186, 0.3732



(a) UDP DDoS (b) Infection Monkey 1 (c) Distributed Stealth Port Scan

Figure 5: **Area under the ROC curve for 3 sample attack traces.**

respectively. These scans involve an adversary probing a selected number of ports across various victim machines. The low traffic volume and variation from these network patterns enable them to evade detection. Similarly, our baselines exhibit inadequate performance in detecting infection monkey attacks due to similar reasons.

In contrast to these efforts, NetVigil uses its feature extractor in tandem with a graphical view of the network to successfully identify these scans and attacks. The features include previously unseen ports, a key characteristic in many scans and attacks. NetVigil also analyzes the number of ports as well as statistics on the number of bytes and packets that are sent/received. A large number of different ports with a comparatively small amount of traffic volume can be indicative of port scanning or of an adversary attempting multiple different exploits that target different ports/services. Furthermore, our GNN architecture detects higher-level behavioral patterns and relationships, rather than just relying on detecting each connection in isolation, as traditional host-based IDSes do. This is useful for detecting distributed port scanning and infection monkey attacks since, contrary to vertical port scanning and DoS attacks, each individual connection exhibits little abnormality in volume and variation, but each malicious actor makes connections to many different hosts, deviating from their typical communication patterns.

Although NetVigil performs well on detecting C&C communication and DNS amplification, it struggles to achieve the same performance as the other attacks. In addition to these scenarios encompassing behavioral patterns of many different nodes, each communication is more similar to traditional network traffic due to the file transfers and DNS queries, making them more difficult for our feature extractor to pick up.

Table 5 shows the results on several selected attacks in which we evaluated Kitsune+ and Whisper in their intended environment using packet-level traces. Overall, their AUC scores are significantly higher when using packet-level traces compared to flow-level traces, with Kitsune+ achieving near-perfect AUC scores for vertical port scanning and UDP DDoS. However, these approaches still fall short in matching NetVigil for other attacks due to the limitations of their extracted features and host-based detection models.

6.3 Efficiency Results

Figure 6 illustrate the wall clock and CPU times of NetVigil compared to Kitsune+ [56] and Whisper [31] for five different attacks. Each experiment involved feature extraction, training, and inference. NetVigil achieves key performance improvements through the following features:

1. Using flow-based features instead of packet-level data, reducing the amount of data significantly since only the aggregated statistics for each flow need to be processed.
2. Using a graph representation that aggregates features across multiple instances of the same connection.
3. An efficient GNN architecture with an autoencoder of two fully-connected layers.

Due to these components, the majority of time is spent during feature extraction. GNN inference takes only 2-3 seconds on average for a network trace with 16 VMs. Across 5 different attack traces, NetVigil achieves significantly lower execution times, yielding speedups of >= 37.59× and 2.87× – 7.38× over Kitsune+ and Whisper, respectively, for wall clock time, and speedups of >= 29.67× and 2.04× – 6.93× for CPU time. Further latency experiments where we varied the number of VMs and cores can be found in Appendix A.1.

6.4 System cost

When evaluating the system costs for Kitsune+ and Whisper, it is important to note that both of these tools necessitate packet traces, which can result in significant CPU and storage overheads to acquire. In addition, both baselines also incur considerable compute overheads during inference, requiring

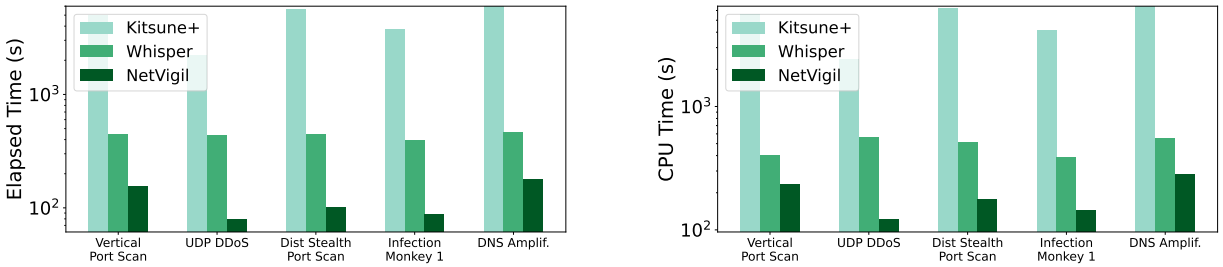


Figure 6: **Comparing detection latencies of Kitsune+, Whisper, and NetVigil in elapsed seconds and CPU time across several attacks. The y-axes use a logarithmic scale.**

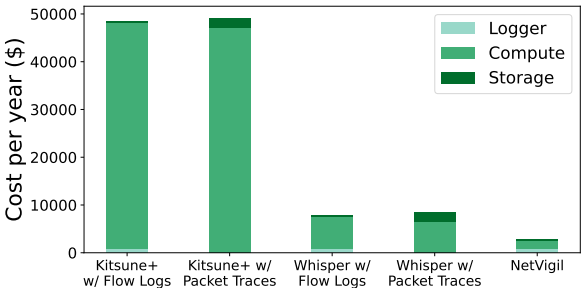


Figure 7: **System cost breakdown for Kitsune+, Whisper, and NetVigil for our 16-VM deployment**

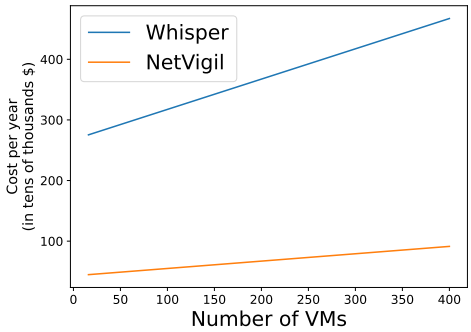


Figure 8: **Estimated monetary system cost for Whisper and NetVigil for production cluster of 400 VMs.**

an 8 vCPU VM for Whisper and a 56 vCPU VM for Kitsune+ to match the performance of NetVigil. Thus, we estimate, for our 16-VM deployment, a total system cost of $49159/year with packet traces and $48428/year with flow traces for Kitsune+ and a cost of $8602/year with packet traces and $7871 with flow traces for Whisper. To put these costs into perspective, we also analyze NetVigil. Because of the low cost of NetVigil, using a 2 vCPU VM is sufficient for performing all inference with flow logs, resulting in a total system cost of $2939/year. Figure 7 shows the system cost breakdown.

We perform a cost assessment of the system at larger scales by analyzing the trace data from our production cluster (see Section 6.5), which consists of 400 virtual machines (VMs). The outcomes are depicted in Figure 8. In comparison with Figure 7, it is clear that NetVigil exhibits superior cost-efficiency than Whisper under this scenario. This observation is primarily attributable to two factors. First, the network throughput in this setting exceeds that of our 16-VM deployment. As a result, Whisper's processing overhead, which is directly proportional to the number of packets, is substantially larger, while NetVigil's overhead remains independent of network throughput. Second, the production cluster primarily utilizes TCP connections, leading to a significantly reduced quantity of network flow records in contrast to the predominantly employed UDP connections in our 16-VM deployment.

6.5 Production Traces

We collect network flow records from two first-party compute clusters. The first cluster, `Service-Cluster`, contains approximately 400 VMs, and we gather traces for a week. The second cluster, `Compute-Cluster`, consists of around 200 VMs, and we acquire traces for two months. We confirm that no known attacks are present in these traces and use them to evaluate the false alarm rate of NetVigil.

We assess the number of false alarms without model retraining. For `Service-Cluster`, there are 4,356 false alarms on the last day of the week if the model isn't retrained, while model retraining reduces false alarms to 10. For `Compute-Cluster`, there are 1,231 false alarms without model retraining at the end of the week, and the number increases to 2,315 on the last day of the month. This cluster has less activity than `Service-Cluster`, explaining the lower dynamics. Model retraining reduces false alarms to fewer than 5 per day. The results from both product traces validate the importance of continuous retraining (Section 4.4).

Additionally, we inject attack traces into these production records to examine the performance of NetVigil. We incorporate Infection Monkey 1, 2, and 3, and replace the IP addresses and timestamps to blend the injected attack traces with normal ones. We observe that the detection accuracy of NetVigil remains consistent (similar to Table 4).

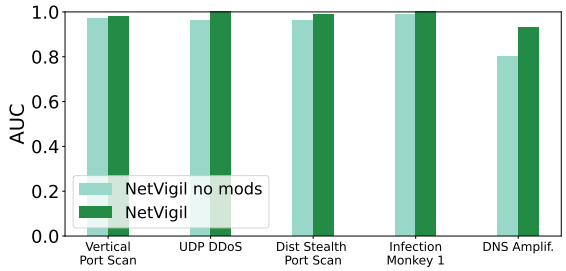


Figure 9: **AUC on several attack traces with and without temporal smoothing and data augmentation**

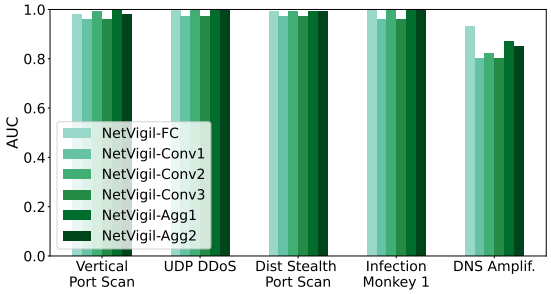


Figure 10: **AUC on several attack traces with different model modifications**

6.6 Ablation Study

We perform experiments to determine the importance of temporal smoothing and data augmentation. Figure 9 shows the AUC for NetVigil both with and without data augmentation and temporal smoothing. Using both techniques yields 1-2% AUC improvement for most baselines and 10% improvement for DNS amplification. Crucially, data augmentations add more heterogeneity to the dataset allowing it to become more robust to the dynamism in network traffic patterns. Temporal smoothing helps by ensuring that temporally similar graphs should be similar in structure and composition. Further, `NetVigil no mods` still performs highly, yielding over 0.95 AUC for all but 1 attack trace showing the efficacy of our approach on new network patterns even without any modifications.

We experiment further with different architectural modifications. Figure 10 shows the AUC results across 5 different attack traces with different model modifications. First we use a simple GNN architecture with fully-connected layers that operate on the graph's node and edge features (denoted by `NetVigil-FC`). We also used 3 different graph convolutional architectures (denoted by `NetVigil-Conv1`, `NetVigil-Conv2`, `NetVigil-Conv3`) that use graph attention layers [78], GraphSAGE layers [34], and EdgeConv layers [81]. We also try different aggregations for our message passing function using `min()` and `max()` instead of `mean()`, denoted by `NetVigil-Agg1` and `NetVigil-Agg2`, respectively. In summary, while `NetVigil-FC` performs the best overall due to its simpler architecture (and lower likelihood of overfitting) compared to the other architectures that use convolutional layers, all model variants perform similarly. We highlight that our approach is not tied to a particular model or architecture and that NetVigil is still able to reap significant performance gains on many different model variants.

7 Related Work

Related work on data center network security is discussed in Sections 1 and 2, and the advantages of NetVigil compared with the most relevant baselines are demonstrated and discussed in detail in Section 6. In this section, we focus on work related to GNNs, graph contrastive learning, and anomaly detection in graphs.

Graph Neural Networks. GNNs have gained significant attention in recent years as powerful tools for analyzing and modeling structured data represented as graphs. A considerable amount of research has been conducted in this field, and a variety of GNN architectures have been proposed. The most common ones include graph convolutional networks [45], graph attention networks [78], and GraphSAGE [34]. We refer the reader to relevant surveys [83, 87, 91] for further details on these and other related architectures. GNNs have achieved state-of-the-art performance in a series of problems in (wired and wireless) communication networks [28, 36, 41]. The NetVigil framework is agnostic to the specific GNN chosen for the encoder and the decoder. Thus, practitioners can seamlessly experiment with different architectures that might better accommodate their data.

Graph Contrastive Learning. Graph contrastive learning has emerged as a potential solution to several challenges faced by GNNs, such as heavy label reliance and weak robustness [50]. The core idea behind graph contrastive learning is to embed augmented versions of the same sample (node, edge, or graph) close to each other while trying to push away embeddings from different samples. Generating these augmented versions of a given sample can be challenging in a graph setting. Unlike images for which different augmented versions (contrastive pairs) can be generated by imposing different color filters or rotation operations, designing contrastive pairs can be challenging in graph settings. Some works use different parts of a graph to build these contrastive pairs [62, 71, 79] by comparing, e.g., nodes with subgraphs [38, 42]. Other works adopt graph data augmentations to generate contrastive pairs [35, 63, 72, 84, 86, 92]. Instead of relying on a generic set of augmentations, in NetVigil we leverage domain-specific knowledge to determine graph transformations that constitute valid contrastive pairs.

Anomaly Detection in Graphs. Anomaly detection is the data mining process that aims to identify patterns in data that do not conform to expected behavior [26]. If we focus on

the graph context, the objective is to find the graph objects (nodes, edges, or substructures) that are rare and that differ significantly from the majority of the reference objects in the given graph [20]. Graph anomaly detection has been applied to myriad settings, including telecom fraud [29], opinion spam [30], and malware detection [64]. Graphs bring specific challenges to the anomaly detection problem related to the inter-dependency of objects (a node being anomalous is not only a function of itself but also of its neighborhood) and the size of the search space (the search space of complex anomalies such as graph substructures is huge). Methods have been derived for the unsupervised [33] and (semi-)supervised [27] settings, for static [25] and dynamic [39] graphs, and for attributed [58] and plain (no attributes) [19] graphs. Notice that our problem falls in the most challenging category of unsupervised learning for dynamic and attributed graphs. Over the last five years, there has been an increasing interest in applying deep learning techniques to graph anomaly detection [52]. The idea is to depart from non-deep learning techniques with limited (linear) representation power [60] and use deep graph representation learning and GNNs to extract expressive representations such that graph anomalies and normal objects can be easily separated. Several methods have been developed for the simpler cases of static or plain graphs [37, 49, 80]. For the dynamic *and* attributed case, the existing techniques are limited [74, 89, 90]. Moreover, real-world networks (including our application) usually exhibit changes in *both* the network structure and node attributes. However, most existing works only consider changes in one of these aspects [74, 90]. To the best of our knowledge, we are the first to consider an autoencoder architecture enhanced by contrastive learning and temporal smoothing to tackle the challenging dynamic setting where both the network structure and attributes are changing.

8 Discussion

Privacy Consideration. Network flow logs may contain personally identifiable information (PII), such as user IP addresses, which are subject to data privacy compliance requirements [22, 76]. To address these privacy concerns, NetVigil employs two strategies. First, our model can be deployed using the Software as a Service (SaaS) model, where users continuously stream anonymized network flow logs to a server running our system. Anonymization can be achieved through encrypted IP addresses, as our model does not require plaintext IP addresses for anomaly detection, and users can interpret the encrypted results accordingly. Second, our model can be deployed within a user's cloud subscription as a standalone service, ensuring that all network flow logs remain entirely under the user's control. By implementing these strategies, we maintain a high level of privacy while still providing effective anomaly detection in network traffic patterns.

Initial Clean Training Set. NetVigil requires at least one clean dataset to train the initial model, with subsequent models obtained as discussed in Section 4.4. As with all anomaly-based intrusion detection systems, if a cloud deployment is already compromised from the outset, some anomalous behaviors might contaminate the model. Therefore, it is much safer to obtain the initial training set in a secure environment (e.g., a sandbox). This precautionary measure helps ensure that the model's foundation is built upon clean and reliable data, allowing it to effectively detect and adapt to genuine anomalies and fluctuations in network traffic patterns.

Applying to North-South Traffic. Although our primary focus in this study is on east-west traffic, the principles of NetVigil, such as employing GNNs for intrusion detection using flow-level logs, can potentially be extended to north-south traffic. This method could further decrease the significant computational costs of existing IDSes or create a comprehensive security solution for both east-west and north-south traffic simultaneously. North-south traffic typically exhibits increased node and contextual information variability, which may necessitate specialized learning techniques. Exploring this design would be an interesting future direction.

9 Conclusion

We present NetVigil, a novel network anomaly detection system specifically designed for securing east-west traffic in large-scale data center networks. Addressing the limitations of existing solutions, our approach focuses on three key objectives: (a) ensuring cost-effectiveness in monitoring numerous nodes, (b) accurately identifying anomalous behaviors while minimizing false alarms, and (c) exhibiting robustness against normal traffic fluctuations without reliance on prior knowledge of malicious attacks. By employing low-cost network flow logs, security-oriented graph features, graph neural networks, and a novel end-to-end training mechanism, our system achieves substantial improvements over existing malicious traffic detectors. We hope that the insights gained from our solution, along with the new east-west security benchmark, Yatesbury, will facilitate the validation of our proposed architecture and foster future research and innovation in this vital area of study.

Acknowledgments

We extend our gratitude to our shepherd, Bruce Davie, and the anonymous reviewers for providing invaluable and constructive feedback. Our thanks also go to our engineering and product collaborators, including Narayan Annamalai, Umair Aftab, Eliran Azulai, Wyman Chong, Jamie Lee, Kiran Muthabatulla, Mariana Alanis Tamez, and Roger Wong, for their contributions to production traces, data processing pipelines, and system requirements.

References

[1] 2017 Equifax data breach. https://en.wikipedia.org/wiki/2017_Equifax_data_breach, Retrieved on 2023-04.

[2] 2019 Zeek specs needed for 10gbps. http://mailman.icsi.berkeley.edu/pipermail/zeek/2019-September/014574.html, Retrieved on 2023-09.

[3] 2020 United States federal government data breach. https://en.wikipedia.org/wiki/2020_United_States_federal_government_data_breach, Retrieved on 2023-04.

[4] A Deeper Look at the Distributed Cloud Firewall: A Firewall for the Cloud Era. https://aviatrix.com/blog/a-deeper-look-at-the-distributed-cloud-firewall-a-firewall-for-the-cloud-era/, Retrieved on 2023-09.

[5] Aviatrix Distributed Cloud Firewall. https://aviatrix.com/distributed-cloud-firewall/, Retrieved on 2023-09.

[6] Calico flow logs. https://docs.tigera.io/calico-cloud/visibility/elastic/flow/, Retrieved on 2023-10.

[7] Deep Graph Library. https://www.dgl.ai/, Retrieved on 2022-11.

[8] Infection monkey. https://www.akamai.com/infectionmonkey, Retrieved on 2023-03.

[9] Introduction to Cilium & Hubble. https://docs.cilium.io/en/stable/overview/intro/, Retrieved on 2023-10.

[10] NetworkX: network analysis in python. https://networkx.org/, Retrieved on 2023-01.

[11] Online boutique. https://github.com/GoogleCloudPlatform/microservices-demo, Retrieved on 2022-07.

[12] pandas - python data analysis library. https://pandas.pydata.org/, Retrieved on 2023-01.

[13] PyTorch. https://pytorch.org/, Retrieved on 2022-11.

[14] Scaling Suricata performance to 100 Gbps with Napatech SmartNICs. https://www.napatech.com/support/resources/solution-descriptions/scaling-suricata-performance-to-100-gbps-with-napatech-smartnics/, Retrieved on 2023-09.

[15] Suricata. https://suricata.io/, Retrieved on 2023-09.

[16] Trends in data center security: Part 1 – traffic trends. https://blogs.cisco.com/security/trends-in-data-center-security-part-1-traffic-trends, Retrieved on 2023-09.

[17] VMware NSX. https://www.vmware.com/products/nsx.html, Retrieved on 2023-09.

[18] VMware NSX: The platform for network virtualization. https://www.virtualizationworks.com/NSX.asp, Retrieved on 2023-09.

[19] Leman Akoglu, Mary McGlohon, and Christos Faloutsos. Oddball: Spotting anomalies in weighted graphs. In *Advances in Knowledge Discovery and Data Mining: 14th Pacific-Asia Conference, PAKDD 2010, Hyderabad, India, June 21-24, 2010. Proceedings. Part II 14*, pages 410–421. Springer, 2010.

[20] Leman Akoglu, Hanghang Tong, and Danai Koutra. Graph based anomaly detection and description: a survey. *Data mining and knowledge discovery*, 29:626–688, 2015.

[21] Amazon. Logging IP traffic using VPC Flow Logs. https://docs.aws.amazon.com/vpc/latest/userguide/flow-logs.html, Retrieved on 2023-04.

[22] Behnaz Arzani, Selim Ciraci, Stefan Saroiu, Alec Wolman, Jack W Stokes, Geoff Outhred, and Lechao Diwu. PrivateEye: Scalable and privacy-preserving compromise detection in the cloud. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2020.

[23] Leyla Bilge and Tudor Dumitraş. Before we knew it: an empirical study of zero-day attacks in the real world. In *Proceedings of the ACM conference on Computer and communications security (CCS)*, 2012.

[24] Brendan Burns, Brian Grant, David Oppenheimer, Eric Brewer, and John Wilkes. Borg, Omega, and Kubernetes. *Communications of the ACM*, 59(5), 2016.

[25] Deepayan Chakrabarti. Autopart: Parameter-free graph partitioning and outlier detection. In *Knowledge Discovery in Databases: PKDD 2004: 8th European Conference on Principles and Practice of Knowledge Discovery in Databases, Pisa, Italy, September 20-24, 2004. Proceedings 8*, pages 112–124. Springer, 2004.

[26] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM Comput. Surv.*, 41(3), jul 2009.

[27] Duen Horng Chau, Shashank Pandit, and Christos Faloutsos. Detecting fraudulent personalities in networks of online auctioneers. In *Knowledge Discovery*

in Databases: PKDD 2006: 10th European Conference on Principles and Practice of Knowledge Discovery in Databases Berlin, Germany, September 18-22, 2006 Proceedings 10, pages 103–114. Springer, 2006.

[28] Arindam Chowdhury, Gunjan Verma, Chirag Rao, Anantham Swami, and Santiago Segarra. Unfolding WMMSE using graph neural networks for efficient power allocation. *IEEE Transactions on Wireless Communications*, 20(9):6004–6017, 2021.

[29] Corinna Cortes, Daryl Pregibon, and Chris Volinsky. Communities of interest. In *Advances in Intelligent Data Analysis: 4th International Conference, IDA 2001 Cascais, Portugal, September 13–15, 2001 Proceedings 4*, pages 105–114. Springer, 2001.

[30] Hanbo Dai, Feida Zhu, Ee-Peng Lim, and HweeHwa Pang. Detecting anomalies in bipartite graphs with mutual dependency principles. In *2012 IEEE 12th International Conference on Data Mining*, pages 171–180, 2012.

[31] Chuanpu Fu, Qi Li, Meng Shen, and Ke Xu. Realtime robust malicious traffic detection via frequency domain analysis. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2021.

[32] Sunanda Gamage and Jagath Samarabandu. Deep learning methods in network intrusion detection: A survey and an objective comparison. *Journal of Network and Computer Applications*, 169, 2020.

[33] Jing Gao, Feng Liang, Wei Fan, Chi Wang, Yizhou Sun, and Jiawei Han. On community outliers and their efficient detection in information networks. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 813–822, 2010.

[34] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[35] Kaveh Hassani and Amir Hosein Khasahmadi. Contrastive multi-view representation learning on graphs. In Hal Daumé III and Aarti Singh, editors, *37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 4116–4126. PMLR, 13–18 Jul 2020.

[36] Shiwen He, Shaowen Xiong, Yeyu Ou, Jian Zhang, Jiaheng Wang, Yongming Huang, and Yaoxue Zhang. An overview on the application of graph neural networks in wireless networks. *IEEE Open Journal of the Communications Society*, 2:2547–2565, 2021.

[37] Renjun Hu, Charu C. Aggarwal, Shuai Ma, and Jinpeng Huai. An embedding approach to anomaly detection. In *IEEE 32nd International Conference on Data Engineering (ICDE)*, pages 385–396, 2016.

[38] Weihua Hu, Bowen Liu, Joseph Gomes, Marinka Zitnik, Percy Liang, Vijay Pande, and Jure Leskovec. Strategies for pre-training graph neural networks. In *International Conference on Learning Representations*, 2020.

[39] Tsuyoshi Idé and Hisashi Kashima. Eigenspace-based anomaly detection in computer systems. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 440–449, 2004.

[40] Illumio. Zero trust: the security paradigm for the modern organization. https://www.illumio.com/solutions/zero-trust, Retrieved on 2023-04.

[41] Weiwei Jiang. Graph-based deep learning for communication networks: A survey. *Computer Communications*, 185:40–54, 2022.

[42] Yizhu Jiao, Yun Xiong, Jiawei Zhang, Yao Zhang, Tianqi Zhang, and Yangyong Zhu. Sub-graph contrast for scalable self-supervised graph representation learning. In *IEEE International Conference on Data Mining (ICDM)*, pages 222–231, 2020.

[43] Ansam Khraisat, Iqbal Gondal, Peter Vamplew, and Joarder Kamruzzaman. Survey of intrusion detection systems: techniques, datasets and challenges. *Cybersecurity*, 2(1), 2019.

[44] John Kindervag, Stephanie Balaouras, and Lindsey Coit. Build security into your network's DNA: The zero trust network architecture. *Forrester Research Inc*, 27, 2010.

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

[46] Vinod Kumar and Om Prakash Sangwan. Signature based intrusion detection system using SNORT. *International Journal of Computer Applications & Information Technology*, 1(3), 2012.

[47] Donghwoon Kwon, Hyunjoo Kim, Jinoh Kim, Sang C Suh, Ikkyun Kim, and Kuinam J Kim. A survey of deep learning-based network anomaly detection. *Cluster Computing*, 22, 2019.

[48] Hung-Jen Liao, Chun-Hung Richard Lin, Ying-Chih Lin, and Kuang-Yuan Tung. Intrusion detection system: A comprehensive review. *Journal of Network and Computer Applications*, 36(1), 2013.

[49] Ninghao Liu, Xiao Huang, and Xia Hu. Accelerated local anomaly detection via resolving attributed networks. In *26th International Joint Conference on Artificial Intelligence*, IJCAI'17, page 2337–2343. AAAI Press, 2017.

[50] Yixin Liu, Ming Jin, Shirui Pan, Chuan Zhou, Yu Zheng, Feng Xia, and Philip S. Yu. Graph self-supervised learning: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 35(6):5879–5900, 2023.

[51] Wai Weng Lo, Siamak Layeghy, Mohanad Sarhan, Marcus Gallagher, and Marius Portmann. E-graphsage: A graph neural network based intrusion detection system for IOT. In *IEEE/IFIP Network Operations and Management Symposium (NOMS)*, 2022.

[52] Xiaoxiao Ma, Jia Wu, Shan Xue, Jian Yang, Chuan Zhou, Quan Z. Sheng, Hui Xiong, and Leman Akoglu. A comprehensive survey on graph anomaly detection with deep learning. *IEEE Transactions on Knowledge and Data Engineering*, pages 1–1, 2021.

[53] Mohammad Masdari and Hemn Khezri. A survey and taxonomy of the fuzzy signature-based intrusion detection systems. *Applied Soft Computing*, 92, 2020.

[54] Microsoft. Flow logs for network security groups. https://learn.microsoft.com/en-us/azure/network-watcher/network-watcher-nsg-flow-logging-overview, Retrieved on 2023-04.

[55] Microsoft. Microsoft digital defense report 2022. https://www.microsoft.com/en-us/security/business/microsoft-digital-defense-report-2022, Retrieved on 2023-04.

[56] Yisroel Mirsky, Tomer Doitshman, Yuval Elovici, and Asaf Shabtai. Kitsune: An ensemble of autoencoders for online network intrusion detection. In *25th Annual Network and Distributed System Security Symposium (NDSS)*, 2018.

[57] Chirag Modi, Dhiren Patel, Bhavesh Borisaniya, Hiren Patel, Avi Patel, and Muttukrishnan Rajarajan. A survey of intrusion detection techniques in cloud. *Journal of network and computer applications*, 36(1), 2013.

[58] Caleb C Noble and Diane J Cook. Graph-based anomaly detection. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 631–636, 2003.

[59] Palo Alto Networks. Prisma cloud: Cloud network security. https://www.paloaltonetworks.com/prisma/cloud/cloud-network-security, Retrieved on 2023-04.

[60] Guansong Pang, Chunhua Shen, Longbing Cao, and Anton Van Den Hengel. Deep learning for anomaly detection: A review. *ACM Comput. Surv.*, 54(2), mar 2021.

[61] Vern Paxson. Bro: A system for detecting network intruders in real-time. *Comput. Netw.*, 31(23–24):2435–2463, dec 1999.

[62] Zhen Peng, Wenbing Huang, Minnan Luo, Qinghua Zheng, Yu Rong, Tingyang Xu, and Junzhou Huang. Graph representation learning via graphical mutual information maximization. In *Proceedings of The Web Conference 2020*, WWW '20, page 259–270. Association for Computing Machinery, 2020.

[63] 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 *26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '20, page 1150–1160. Association for Computing Machinery, 2020.

[64] Md Sazzadur Rahman, Ting-Kai Huang, Harsha V Madhyastha, and Michalis Faloutsos. Efficient and scalable socware detection in online social networks. In *USENIX Security Symposium*, pages 663–678, 2012.

[65] Martin Roesch. Snort - lightweight intrusion detection for networks. In *Proceedings of the 13th USENIX Conference on System Administration*, LISA '99, page 229–238, USA, 1999. USENIX Association.

[66] Scott Rose, Oliver Borchert, Stu Mitchell, and Sean Connelly. Zero trust architecture. Technical report, National Institute of Standards and Technology, 2020.

[67] Arjun Roy, Hongyi Zeng, Jasmeet Bagga, George Porter, and Alex C. Snoeren. Inside the social network's (datacenter) network. In *Proceedings of the ACM Conference on Special Interest Group on Data Communication (SIGCOMM)*, 2015.

[68] Fred B Schneider. Least privilege and more computer security. *IEEE Security & Privacy*, 1(5), 2003.

[69] Rupam Kumar Sharma, Hemanta Kumar Kalita, and Biju Issac. Different firewall techniques: A survey. In *Fifth International Conference on Computing, Communications and Networking Technologies (ICCCNT)*, 2014.

[70] Nitin Singh Sikarwar and Dinesh Verma. Micro segmentation: today's success formulae. *International Journal of Operations Management and Services*, 2(1), 2012.

[71] Fan-Yun Sun, Jordan Hoffman, Vikas Verma, and Jian Tang. InfoGraph: Unsupervised and semi-supervised graph-level representation learning via mutual information maximization. In *International Conference on Learning Representations*, 2020.

[72] Susheel Suresh, Pan Li, Cong Hao, and Jennifer Neville. Adversarial graph augmentation to improve graph contrastive learning. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 15920–15933, 2021.

[73] Colin Tankard. Advanced persistent threats and how to monitor and deter them. *Network security*, 2011(8), 2011.

[74] Xian Teng, Yu-Ru Lin, and Xidao Wen. Anomaly detection in dynamic networks using multi-view time-series hypersphere learning. In *2017 ACM on Conference on Information and Knowledge Management*, pages 827–836, 2017.

[75] Zhihong Tian, Wei Shi, Yuhang Wang, Chunsheng Zhu, Xiaojiang Du, Shen Su, Yanbin Sun, and Nadra Guizani. Real-time lateral movement detection based on evidence reasoning network for edge computing environment. *IEEE Transactions on Industrial Informatics*, 15(7), 2019.

[76] European Union. General data protection regulation (GDPR). https://commission.europa.eu/law/law-topic/data-protection_en, Retrieved on 2023-04.

[77] Romans Vanickis, Paul Jacob, Sohelia Dehghanzadeh, and Brian Lee. Access control policy enforcement for zero-trust-networking. In *29th Irish Signals and Systems Conference (ISSC)*, 2018.

[78] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations*, 2018.

[79] Petar Velickovic, William Fedus, William L. Hamilton, Pietro Liò, Yoshua Bengio, and R Devon Hjelm. Deep graph infomax. In *International Conference on Learning Representations*, 2019.

[80] Yanling Wang, Jing Zhang, Shasha Guo, Hongzhi Yin, Cuiping Li, and Hong Chen. Decoupling representation learning and classification for GNN-based anomaly detection. In *44th international ACM SIGIR conference on research and development in information retrieval*, pages 1239–1248, 2021.

[81] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E. Sarma, Michael M. Bronstein, and Justin M. Solomon. Dynamic graph cnn for learning on point clouds. *ACM Trans. Graph.*, 38(5), oct 2019.

[82] Marcus Willett. Lessons of the SolarWinds hack. *Survival*, 63(2), 2021.

[83] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 32(1):4–24, 2021.

[84] Yuning You, Tianlong Chen, Yang Shen, and Zhangyang Wang. Graph contrastive learning automated. In Marina Meila and Tong Zhang, editors, *38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 12121–12132. PMLR, 18–24 Jul 2021.

[85] Yuning You, Tianlong Chen, Yongduo Sui, Ting Chen, Zhangyang Wang, and Yang Shen. Graph contrastive learning with augmentations. *Advances in neural information processing systems (NeurIPS)*, 33, 2020.

[86] Yuning You, Tianlong Chen, Yongduo Sui, Ting Chen, Zhangyang Wang, and Yang Shen. Graph contrastive learning with augmentations. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 5812–5823, 2020.

[87] Ziwei Zhang, Peng Cui, and Wenwu Zhu. Deep learning on graphs: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 34(1):249–270, 2022.

[88] Zhipeng Zhao, Hugo Sadok, Nirav Atre, James C Hoe, Vyas Sekar, and Justine Sherry. Achieving 100gbps intrusion prevention on a single server. In *Proceedings of the 14th USENIX Conference on Operating Systems Design and Implementation (OSDI)*, 2020.

[89] Li Zheng, Zhenpeng Li, Jian Li, Zhao Li, and Jun Gao. Addgraph: Anomaly detection in dynamic graph using attention-based temporal gcn. In *28th International Joint Conference on Artificial Intelligence*, IJCAI'19, page 4419–4425, 2019.

[90] Panpan Zheng, Shuhan Yuan, Xintao Wu, Jun Li, and Aidong Lu. One-class adversarial nets for fraud detection. In *AAAI Conference on Artificial Intelligence*, volume 33, pages 1286–1293, 2019.

[91] Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *AI Open*, 1:57–81, 2020.

[92] Yanqiao Zhu, Yichen Xu, Feng Yu, Qiang Liu, Shu Wu, and Liang Wang. Graph contrastive learning with adaptive augmentation. In *Proceedings of the Web Conference*, WWW '21, page 2069–2080, 2021.

A Appendix

A.1 Additional efficiency results

We demonstrate the scalability of our approach to larger network log sizes. With a trace of 4 VMs, it takes 123 seconds for Whisper and 96 seconds for NetVigil. As the trace size increases, the execution time of Whisper also increases, resulting in 373 seconds to process a trace with 16 VMs. Meanwhile, the execution time of NetVigil only increases slightly to 140 seconds. Furthermore, as Whisper is allocated more cores, the CPU time increases from 508 seconds for 4 cores to 1520 seconds for 16 cores, while the wall clock time decreases marginally from 173 seconds with 4 cores to 164 seconds with 8 cores. In contrast, the runtime of NetVigil remains relatively stable when changing the number of allocated cores since it does not rely on parallelism for efficiency.

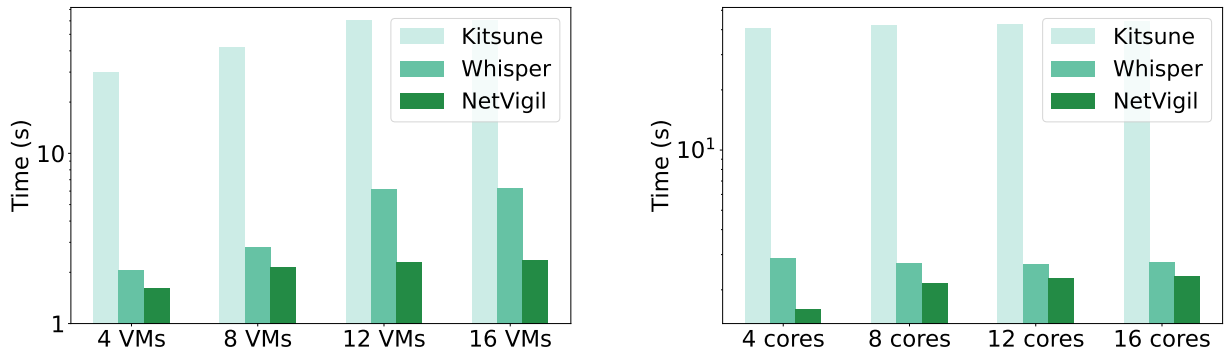


Figure 11: **Comparing detection latencies of Kitsune+, Whisper, and NetVigil in elapsed seconds while varying number of VMs in the network trace (left), number of cores used for processing each trace (middle), and attack traces (right).**

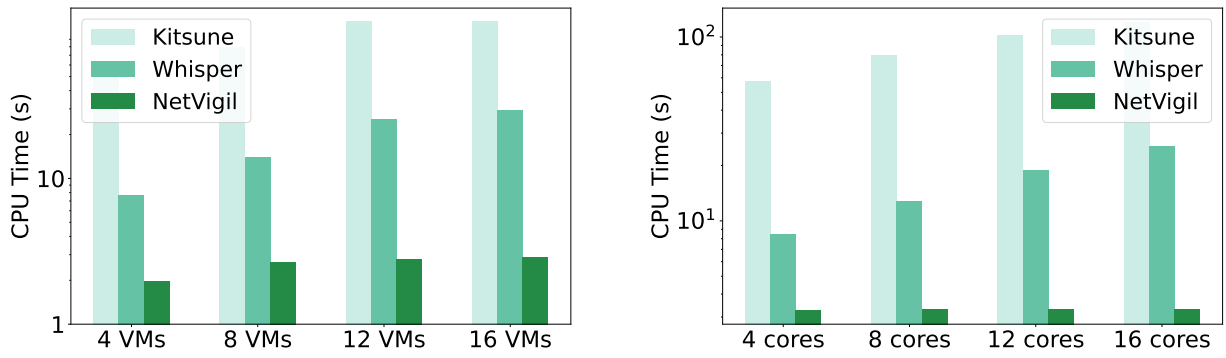


Figure 12: **Comparing detection latencies of Kitsune+, Whisper, and NetVigil in CPU seconds while varying number of VMs in the network trace (left), number of cores used for processing each trace (middle), and attack traces (right).**