GraphFaaS: Serverless GNN Inference for Burst-Resilient, Real-Time Intrusion Detection

Lingzhi Wang
Northwestern University

Vinod Yegneswaran
SRI International

Xinyi Shi
Northwestern University

Ziyu Li
Northwestern University

Ashish Gehani
SRI International

Yan Chen
Northwestern University

Abstract

Provenance-based intrusion detection is an increasingly popular application of graphical machine learning in cybersecurity, where system activities are modeled as provenance graphs to capture causality and correlations among potentially malicious actions. Graph Neural Networks (GNNs) have demonstrated strong performance in this setting. However, traditional statically-provisioned GNN inference architectures fall short in meeting two crucial demands of intrusion detection: (1) maintaining consistently low detection latency, and (2) handling highly irregular and bursty workloads. To holistically address these challenges, we present GraphFaaS, a serverless architecture tailored for GNN-based intrusion detection. GraphFaaS leverages the elasticity and agility of serverless computing to dynamically scale the GNN inference pipeline. We parallelize and adapt GNN workflows to a serverless environment, ensuring that the system can respond in real time to fluctuating workloads. By decoupling compute resources from static provisioning, GraphFaaS delivers stable inference latency, which is critical for dependable intrusion detection and timely incident response in cybersecurity operations. Preliminary evaluation shows GraphFaaS reduces average detection latency by 85% and coefficient of variation (CV) by 64% compared to the baseline.

1 Introduction

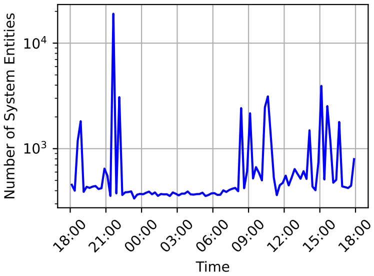

Figure 1: Fluctuations in detection workload over time in the DARPA TC dataset [9]

Graph-based intrusion detection has emerged as a critical application of graph analysis in cybersecurity. In this paradigm, system behaviors recorded in audit logs are transformed into provenance graphs, where nodes represent system entities (e.g., processes, files, network nodes), and edges represent system-level events (e.g., file read/write, process creation). These graphs provide a natural way to model the causal and temporal relationships underlying cyberattacks and anomalous behavior. Recently, Graph Neural Networks (GNNs) have demonstrated strong promise in this domain by learning complex patterns that enable context-aware intrusion detection. However, existing GNN inference architectures struggle to scale in this setting, as they must (1) maintain stable and low detection latency, while also (2) handling inherently irregular and bursty workloads (i.e., workloads with sudden surges in demand). First, in intrusion detection systems, it is particularly crucial for GNN inference to maintain stability and low latency. If the inference time is too long, there is a high risk of missing the window to mitigate real attacks, potentially leading to irreversible consequences. Therefore,

ML For Systems workshop at Neural Information Processing Systems (NeurIPS 2025).

ensuring fast and consistent inference performance is essential for effective threat response. Second, such systems often face unstable and bursty workloads (Figure 1), while malicious activities account for only a small fraction of overall network behavior, leading to highly unpredictable and sporadic patterns in the analysis workload.

Serverless design offers advantages in handling bursty workload and maintaining consistent low detection latency. A serverless system is a cloud model where code runs on-demand without managing servers. It provides a cost-efficient pay-per-use model with no charges for idle resources, while scaling instantly and seamlessly to handle traffic spikes. Additionally, it enables faster development through an event-driven model and ensures finer-grained resource utilization, making it especially well-suited for workloads that are intermittent, unpredictable, or highly event-driven.

Contributions. This paper introduces GraphFaaS, a serverless architecture for low-latency, scalable GNN-based intrusion detection. Our contributions include: (i) A **provenance-aware graph construction pipeline** that avoids redundant computation by leveraging temporal locality in system ls and applying two-stage filtering, based on structural proximity and frequency, to isolate relevant subgraphs. (ii) A **serverless node embedding layer**, which parallelizes node-level feature transformations across dynamic workloads with a feature-length-aware execution strategy to ensure each unit completes within a fixed latency bound while minimizing overhead. (iii) A **scalable GNN inference mechanism**, adapted for serverless execution, which partitions large provenance graphs into balanced subgraphs using a greedy best-fit algorithm and a vertical-scaling fallback to handle extreme cases where graph fragments exceed preset limits, (iv) An **end-to-end implementation** of GraphFaaS on the OpenFaaS platform, evaluated on the DARPA TC dataset [9]. Our system achieves a 6.7x reduction in average detection latency and a 64% reduction in coefficient of variation (CV) compared to a state-of-the-art baseline, while maintaining equivalent detection accuracy.

2 Background and Related Work

Provenance-based Intrusion Detection. Intrusion detection is the process of monitoring and analyzing computer systems or networks to identify unauthorized access, malicious activity, or policy violations. In provenance-based intrusion detection systems (PIDS), system behaviors are modeled as directed acyclic graphs (provenance graphs), where nodes represent system entities (e.g., processes, files, sockets, pipes, memory objects) and edges capture interactions between these entities (e.g., reading a file or connecting to a remote host). The provenance graphs evolve dynamically as the system operates, forming a temporal record that encodes the causal relationships among system entities, which is then used to correlate the suspicious traces in the cyberattacks. In recent years, graph machine learning methods, particularly GNNs, have become a prominent approach for detecting advanced persistent threats (APTs) from provenance graphs [1, 2, 10, 3, 8]. Typically, continuous log streams are transformed into fixed-duration graphs, which are then processed by GNN models. Most PIDSs are trained to learn the benign graph patterns. During inference, the GNN model takes a graph snippet as input. The resulting node and graph embeddings are then analyzed to detect deviations from benign graph patterns.

Serverless for GNN. Recent efforts have tackled the challenges of efficient GNN execution from both hardware and system perspectives. GNNAdvisor [14] optimizes GPU utilization by tailoring runtime behaviors based on GNN model and workload characteristics, but focuses mainly on training/inference under static resources. λ Grapher [7] enables efficient GNN serving by exploiting request-level graph locality and fine-grained resource control. Dorylus [11] leverages serverless threads with CPU servers for scalable GNN training, achieving cost-efficiency via computation separation, yet it does not address serving workloads. Fograph [15] targets low-latency GNN inference for IoT by using fog nodes close to data sources, mitigating cloud communication overhead. However, it relies on specialized edge deployments rather than general-purpose serverless infrastructures. While some works explore migrating graph processing to FaaS platforms, they suffer from poor scalability due to high communication overhead and coarse-grained execution [12]. Unlike prior systems that focus on static provisioning, training performance, or specialized edge deployments, GraphFaaS uniquely enables low-latency, scalable GNN inference for intrusion detection by combining fine-grained graph filtering, adaptive partitioning, and dynamic resource scaling within a serverless framework.

3 Typical Detection Workflow of GNN-based PIDS

Log processing and graph construction. The source of intrusion detection is the log stream,i.e., system log data arranged in chronological order. Graph construction extracts system entities and

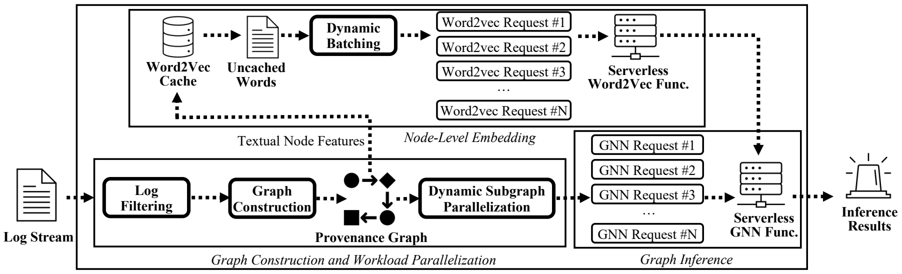

Figure 2: Overview of GraphFaaS Framework with three components: (i) graph construction, (ii) serverless node-level embedding, and (iii) serverless GNN inference.

events from the log stream, representing them as nodes and edges in the provenance graph, and stores them in a graph database.

Node-level embedding. The second step is node-level embedding, also known as featurization [2], which transforms the static attributes of each node into numerical vectors. These vectors act as the initial node representations for the GNN. In provenance graphs, node attributes are typically represented as textual data, for example, process names and command lines for process nodes, file paths for file nodes, and IP addresses and ports for network nodes. To embed these textual features into a vector space, common techniques such as word2vec and doc2vec are often employed.

GNN. The node-level embeddings provide the initial representations for GNN. During message passing, the GNN propagates and aggregates node features across its neighbors. The scope of this propagation, i.e., the receptive field, is determined by the number of GNN layers. After propagation, each node is assigned an updated vector representation. To train the GNN parameters, typical optimization objectives include node classification, edge type prediction, or graph reconstruction.

Result Aggregation. GNN generates an embedding for each node. For an intrusion detection system, these embeddings are further analyzed to determine whether an intrusion has occurred. Common approaches in existing work include applying clustering methods, fixed thresholds, or outlier detection techniques to identify anomalous nodes.

4 Serverless GNN-based Intrusion Detection Architecture

As shown in Figure 2, the GraphFaaS framework is composed of three main components: (i) graph construction, (ii) serverless node-level embedding, and (iii) serverless GNN inference.

Graph Construction. We start by constructing the provenance graph from the log stream. Although the provenance graph can be very large, we find that most of its structure stays the same between two detection intervals. To take advantage of this, we use log filtering: instead of reprocessing the entire graph, we focus only on the parts that have changed. Specifically, we keep nodes that are within a 2K-hop distance of the active nodes (where the GNN has K layers), which avoids redundant computation. We can also apply frequency-based filtering [4, 5]: we keep only the edges, nodes, and their 2K-hop neighborhoods that occur infrequently in the training data, since rare patterns are usually more important for detection. Finally, the filtered subgraphs are split into subtasks and run in parallel, allowing us to fully leverage serverless auto-scaling.

Serverless Node-level Embedding. Our goal is to ensure consistent and low inference latency under bursty workloads. The core idea is to divide the workload into small, parallel execution units, each requiring only a short processing time below a predefined threshold. As the workload changes, the number of execution units adjusts accordingly. Leveraging the automatic scaling capability of the serverless architecture, computing resources are allocated adaptively to match demand. This design guarantees that all execution units complete within the time threshold, regardless of their number. For instance, when a large workload arrives, it is split into more execution units, and the serverless platform automatically scales up by launching additional instances to handle them. Conversely, when the workload is low, the platform scales down to release resources and avoid waste. GNN inference typically consists of two stages: node-level embedding and message passing with aggregation. We present how each of these stages can be adapted to a serverless architecture.

As introduced in §3, node-level embedding transforms node attributes into vectors that serve as the initial representations for GNNs. Because static node attributes are independent of one another, the embedding process is naturally parallelizable across nodes. We implement a serverless function that

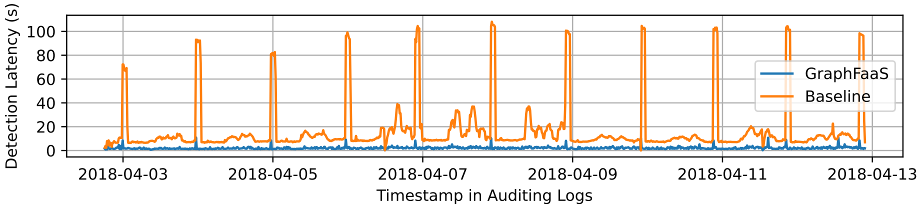

Figure 3: Detection latency over time. GraphFaaS: mean = 2.10, STD (standard deviation) = 1.09, CV (coefficient of variation) = 0.52; Baseline: mean = 14.16, STD = 4498.92, CV = 1.46.

takes a node attribute as input and returns its vector embedding. Since node attributes are typically represented as textual strings, and execution time for most embedding methods depends on string length, we partition execution units according to feature length. Shorter strings are grouped into a single execution unit, while longer strings are processed separately. This strategy ensures that each execution unit completes within the designated time threshold while also reducing the overhead of excessive parallelization, such as network transmission and packet processing costs.

GNN Inference. We implement a serverless function that takes the initial node embeddings and edges as input and outputs the updated node vectors. Similar to the embedding stage, the workload is divided into execution units so that the serverless platform can scale automatically to handle bursty workloads. Since the model remains fixed during detection, inference latency is primarily determined by the graph size. To efficiently process large graphs, we partition them into subgraphs whose sizes are close to a predefined threshold. This partitioning strategy needs to address the key challenge of workload balancing, specifically, how to partition or pack the graph such that each subgraph stays within the desired size limit while minimizing the total number of subgraphs, thereby avoiding excessive waste of computational resources. We design a greedy best-fit algorithm to partition the graph (See AppendixA for more details). Due to the dependency explosion problem in provenance graphs, even the smallest subgraph (a central node and its k-hop neighbors) may exceed the preset size limit. In such cases, vertical scaling is triggered, allocating more compute resources (e.g., CPU cores, memory) to each serverless function instance, instead of spawning more instances.

5 Preliminary Results

We implemented GraphFaas in Python based on OpenFaaS, an open-source serverless platform. We compared GraphFaas with Flash [10], a state-of-the-art PIDS. To ensure a fair comparison, we reimplemented Flash to adapt it to a server-client detection model, where data is sent from the monitored machine to a detection server for analysis. The only difference is that we used a Docker container to simulate a statically provisioned detection environment for Flash, whereas GraphFaas was deployed using a serverless architecture based on OpenFaaS. All other experimental conditions were kept identical between the two systems. We used a widely adopted dataset from DARPA TC Engagement 3 [9], which contains 11 days of audit logs and includes four attack campaigns.

We verified that GraphFaaS achieves the same detection accuracy as Flash, which is expected since the underlying detection models of Flash remain unchanged. Therefore, the detection results are identical. However, GraphFaaS significantly outperforms Flash in detection latency. As shown in Figure 3, on a 11-day dataset, it achieves an average latency of 2.1 seconds, compared to 14.16 seconds for the baseline. In addition, GraphFaaS is more resilient to bursty traffic. It maintains low latency even during sudden surges in workload, as evidenced by its lower standard deviation and coefficient of variation. While occasional latency spikes occur, the delay never exceeds 10 seconds.

6 Conclusion and Future Work

GraphFaaS leverages the elasticity and agility of a serverless design to ensure consistently low intrusion detection latency. Several challenges remain for future work. First, provenance graphs are particularly vulnerable to the dependency explosion problem [6, 13], where certain super-nodes dominate the graph. Even with workload parallelization, these super-nodes remain a major latency bottleneck. Second, during cyberattacks, the scale of the provenance graph can fluctuate dramatically. To effectively capture long-range dependencies while preserving low latency, it is necessary to dynamically adjust the number of GNN layers and invoke the appropriate serverless functions on demand. We believe GraphFaaS can serve as the basis to address these issues with serverless design. Finally, we plan to evaluate GraphFaaS against a broader class of PIDs and across more datasets.

Acknowledgments and Disclosure of Funding

This work was supported in part by the National Science Foundation (NSF) under Grant No. CNS-2229455. The authors gratefully acknowledge the support of the NSF, which made this research possible.

References

[1] Talha Abrar, Ahmad Shamail, Jaffer Iqbal, Amaan Ahmed, Muhammad Abdullah, Muhammad Shayan, Fareed Zaffar, Thomas Pasquier, David Eyers, and Ashish Gehani. On the Reproducibility of Provenance-based Intrusion Detection that uses Deep Learning. *3rd ACM Conference on Reproducibility and Replicability (REP)*, 2025.

[2] 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 *USENIX Security Symposium*, 2025.

[3] 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 *2024 IEEE Symposium on Security and Privacy (SP)*, pages 3533–3551. IEEE, 2024.

[4] Feng Dong, Liu Wang, Xu Nie, Fei Shao, Haoyu Wang, Ding Li, Xiapu Luo, and Xusheng Xiao. {DISTDET}: A {Cost-Effective} distributed cyber threat detection system. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 6575–6592, 2023.

[5] 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 systems security symposium*, 2019.

[6] Md Nahid Hossain, Sanaz Sheikhi, and R Sekar. Combating dependence explosion in forensic analysis using alternative tag propagation semantics. In *2020 IEEE symposium on security and privacy (SP)*, pages 1139–1155. IEEE, 2020.

[7] Haichuan Hu, Fangming Liu, Qiangyu Pei, Yongjie Yuan, Zichen Xu, and Lin Wang. λgrapher: A resource-efficient serverless system for gnn serving through graph sharing. In *Proceedings of the ACM Web Conference 2024*, pages 2826–2835, 2024.

[8] 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.

[9] DARPA TC program. Transparent Computing Engagement 3 Data Release, 2020.

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

[11] John Thorpe, Yifan Qiao, Jonathan Eyolfson, Shen Teng, Guanzhou Hu, Zhihao Jia, Jinliang Wei, Keval Vora, Ravi Netravali, Miryung Kim, et al. Dorylus: Affordable, scalable, and accurate {GNN} training with distributed {CPU} servers and serverless threads. In *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21)*, pages 495–514, 2021.

[12] Lucian Toader, Alexandru Uta, Ahmed Musaafir, and Alexandru Iosup. Graphless: Toward serverless graph processing. In *2019 18th International Symposium on Parallel and Distributed Computing (ISPDC)*, pages 66–73. IEEE, 2019.

[13] Lingzhi Wang, Xiangmin Shen, Weijian Li, Zhenyuan Li, R Sekar, Han Liu, and Yan Chen. Incorporating gradients to rules: Towards lightweight, adaptive provenance-based intrusion detection. *arXiv preprint arXiv:2404.14720*, 2024.

[14] Yuke Wang, Boyuan Feng, Gushu Li, Shuangchen Li, Lei Deng, Yuan Xie, and Yufei Ding. {GNNAdvisor}: An adaptive and efficient runtime system for {GNN} acceleration on {GPUs}. In *15th USENIX symposium on operating systems design and implementation (OSDI 21)*, pages 515–531, 2021.

[15] Liekang Zeng, Peng Huang, Ke Luo, Xiaoxi Zhang, Zhi Zhou, and Xu Chen. Fograph: Enabling real-time deep graph inference with fog computing. In *Proceedings of the ACM Web Conference 2022*, pages 1774–1784, 2022.

A Algorithm for Subgraph Partition & Packing

The goal of subgraph partitioning & packing is to divide a large graph into several smaller subgraphs such that the size of the K-hop neighborhood of each subgraph remains within a predefined threshold. At the same time, we aim to avoid excessive fragmentation by merging K-hop neighborhoods that share overlapping regions. In other words, we seek a partitioning of the original graph where each subgraph's K-hop neighborhood is bounded in size, while also minimizing the total number of subgraphs. For each cluster k, we maintain its current node set $U_k = N_2(X_k)$, represented as a bitset, and its edge count $f_k = |E_G[U_k]|$. When a new vertex v is considered for insertion, we compute the increase in edge count as:

$$\Delta_k = |E_G[U_k \cup B_2(v)]| - f_k.$$

If Δ_k does not exceed the remaining capacity of cluster k, the vertex is inserted into that cluster. Otherwise, a new cluster is created.

To minimize the total number of clusters used, we adopt the *Best-Fit* strategy, which attempts to pack the vertex into the cluster where it fits most tightly. A simpler alternative is the *First-Fit* strategy.

Algorithm 1 BinPack_FFD (First-Fit Decreasing Bin Packing) for Graph Partitioning

```
Require: List of neighborhoods neighbors, capacity capacity, strict flag strict
Ensure: List of bins and remaining capacities
Sort neighbors by number of edges in descending order
Initialize bins as empty list
Initialize remain as empty list
Initialize subgraph as empty list of edge sets
for each (idx, neighborhood) in sorted items do
    placed $\leftarrow$ False
    edge_set_to_add $\leftarrow$ set of edges in neighborhood
    for $b = 0$ to $|$bins$| - 1$ do
        margin $\leftarrow$ size of edge_set_to_add minus overlapping edges with subgraph[b]
        if strict is True then
            if margin $<$ remain[b] then
                Add idx to bins[b]
                Update subgraph[b] and remain[b]
                placed $\leftarrow$ True, break
            end if
        else
            if margin $\leq$ remain[b] then
                Add idx to bins[b]
                Update subgraph[b] and remain[b]
                placed $\leftarrow$ True, break
            end if
        end if
    end for
    if placed is False then
        Create new bin with idx
        Add edge_set_to_add to subgraph
        Append capacity - |edge_set_to_add| to remain
    end if
end for
return bins, remain
```