Internship Report: Machine Learning-Based Network Attack Detection

CYBERUS master in Cybersecurity

12 June 2025

Huynh Tuan Khoi Nguyen, Volonterio Luca

Keywords: Graph Neural Networks, Network Anomaly Detection, Graph Attention Networks, Machine Learning, Intrusion Detection

Contents

Introduction

In the contemporary digital ecosystem, local area networks (LANs) have evolved into critical infrastructure components that underpin both small-scale residential environments and large-scale enterprise operations. These networks facilitate seamless connectivity and resource sharing among diverse devices, ranging from traditional computers to Internet of Things (IoT) devices. However, this increasing network complexity, coupled with the growing sophistication of cyber threats, has transformed LANs into attractive targets for malicious actors seeking to establish persistent footholds within protected environments.

Compromised devices within LAN environments present multifaceted security risks, serving as potential entry points for unauthorized data access, command-and-control communications, spam distribution networks, or recruitment into large-scale botnet operations. The challenge of detecting such compromised devices is compounded by several factors: the heterogeneous nature of connected devices, the prevalence of encrypted communication protocols, and the continuous evolution of malware techniques designed to evade traditional detection mechanisms.

Contemporary signature-based detection methods, while effective against known threats, demonstrate significant limitations when confronted with zero-day attacks, polymorphic malware, or sophisticated adversaries employing encryption to obfuscate malicious activities. This reality necessitates the development of advanced, behavior-based detection solutions capable of identifying anomalous patterns without relying on predefined attack signatures.

This project presents a comprehensive, open-source proof-of-concept solution for detecting compromised devices within LAN environments through innovative graph-based traffic analysis combined with machine learning techniques. Our approach leverages a dual-module architecture: a high-performance C++ component responsible for efficient graph construction from network traffic logs, and a sophisticated Python module that employs Graph Attention Networks (GATs) to identify and classify anomalous traffic patterns.

The primary contributions of this work include:

- **Efficient Graph-Based Traffic Modeling**: Development of a scalable, real-time graph representation framework that captures complex network interaction patterns while maintaining computational efficiency suitable for deployment in resource-constrained environments.
- **Hybrid Machine Learning Detection Framework**: Implementation of a novel detection approach that synergistically combines graph structural analysis with attention-based neural networks, enabling accurate identification of subtle anomalous behaviors that traditional methods might overlook.
- **Open-Source Implementation**: Provision of a complete, deployable solution that can be adapted and extended by the cybersecurity community, promoting collaborative advancement in network security research.

This report is structured to provide comprehensive coverage of our methodology and findings: Section 1 examines the foundational research and related work that informed our approach, Section 2 details the system architecture and core methodologies while Section 3 and 4 focuses on the description of the network representation as an oriented graph, Section 5 presents our experimental framework and performance evaluation results. Section 6 demonstrates how we can deploy our system in real life in a portable and scalable manner.

1 Related Work

The detection of compromised devices within local area networks (LANs) represents a critical challenge in modern cybersecurity, particularly given the increasing sophistication of threats and the widespread adoption of encryption. This section examines foundational research that has shaped our approach to graph-based anomaly detection in network security. Three seminal works have significantly influenced the development of our solution: HyperVision's real-time detection of encrypted malicious traffic, a comprehensive survey on dynamic graph anomaly detection, and AddGraph's attention-based temporal approach to anomaly detection in dynamic graphs.

1.1 HyperVision: Real-Time Detection of Encrypted Malicious Traffic

HyperVision [3] proposes an unsupervised machine learning approach to detecting unknown encrypted malicious traffic in real-time. Central to its methodology is the representation of network traffic as a flow interaction graph, where flows are modeled as vertices and interactions between flows are represented as edges. This graph-based approach allows for the analysis of complex relational structures inherent in network traffic. The key innovation of HyperVision lies in its ability to detect encrypted malicious traffic without requiring labeled datasets of known attacks. The system addresses the challenge that encrypted malicious traffic has similar features to benign flows, making it difficult for traditional detection methods to identify threats. By analyzing flow interaction patterns rather than individual flow characteristics, HyperVision can detect attacks that involve multiple steps with different flow interactions between attackers and victims. HyperVision employs a compact in-memory graph construction strategy that handles the challenge of dense graphs by processing short and long flows separately. Short flows are aggregated based on similarity patterns, while long flows undergo distribution fitting to preserve interaction information. The system achieves significant performance improvements, demonstrating at least 0.92 AUC and 0.86 F1 scores while maintaining detection throughput exceeding 80.6 Gb/s with average latency of 0.83 seconds. Our project adopts the graph-based approach from HyperVision, leveraging a C++ graph builder that efficiently constructs an incremental graph from network logs generated by Zeek. By capturing flow interactions and calculating graph attributes, we ensure that our graph retains sufficient information for anomaly detection, aligning with the principles outlined in HyperVision.

1.2 Anomaly Detection in Dynamic Graphs: A Comprehensive Survey

The comprehensive survey by Ekle and Eberle [1] provides a detailed taxonomy of anomaly detection methods in dynamic graphs, categorizing approaches into traditional machine learning models, matrix transformations, probabilistic methods, and deep learning techniques. The survey emphasizes the advantages of graph-based approaches for anomaly detection, particularly their ability to adapt to topological structures, integrate multimodal data, and scale to large networks. The survey highlights several key advantages of Graph Neural Networks (GNNs) for anomaly detection: their adaptation to topological structures through dynamic information propagation, integration of multimodal data with varying cardinalities, scalability on large-scale networks, and efficiency in model interpretability. Particularly relevant is the discussion of Graph Attention Networks (GATs), which can capture intricate patterns and complex graph dependencies in dynamic networks. The survey identifies three main types of anomalies: point anomalies (individual data points deviating from normal patterns), contextual anomalies (data points anomalous in specific contexts), and collective anomalies (collections of related data points that are anomalous together). This classification framework provides important guidance for designing detection systems that can identify different types of network security threats. Inspired by this survey, our solution integrates a Python-based module that applies GATs to the incremental graph generated by our C++ module. This approach allows our system to learn traffic patterns and detect anomalies by identifying significant deviations from learned patterns. Additionally, the survey's insights into probabilistic methods encouraged us to consider including a probabilistic model in our detection framework, enhancing the robustness of our anomaly detection approach.

1.3 AddGraph: Attention-based Temporal GCN for Dynamic Graph Anomaly Detection

The AddGraph framework [9] introduces a sophisticated approach to anomaly detection in dynamic graphs using attention-based temporal Graph Convolutional Networks (GCNs). This work addresses critical challenges in detecting anomalous edges within dynamic graph structures, particularly focusing on scenarios where anomalous patterns may be both explicit and implicit.

AddGraph extends traditional GCN models by incorporating temporal information through Gated Recurrent Units (GRUs) with contextual attention mechanisms. This architecture enables the system to capture both long-term behavioral patterns and short-term anomalous activities within dynamic graphs. The framework is particularly relevant to our LAN security application, as it demonstrates how attention mechanisms can effectively combine structural, content, and temporal features for anomaly detection.

A key innovation of AddGraph is its semi-supervised learning approach, which addresses the challenge of insufficient labeled anomaly data through selective negative sampling and margin loss techniques. This approach is particularly valuable in network security contexts where labeled malicious traffic data may be scarce or unrepresentative. The framework's ability to learn from limited labeled data while leveraging the

structural properties of the graph aligns well with real-world deployment scenarios in LAN environments.

The AddGraph methodology demonstrates significant performance improvements over state-of-the-art competitors, particularly in detecting anomalous edges that represent suspicious connections or activities. This edge-focused detection approach is directly applicable to our use case, where compromised devices may exhibit anomalous connection patterns within the LAN topology.

By combining the graph representation approach from HyperVision, the comprehensive anomaly detection taxonomy from the dynamic graph survey, and the attention-based modeling from AddGraph, our solution effectively leverages state-of-the-art methodologies for detecting compromised devices in LAN networks.

2 System Architecture and Methodology

The system architecture consists of two primary components: the C++ graph builder for real-time network traffic graph construction and the Python-based anomaly detection module utilizing Graph Attention Networks (GATs).

2.1 Graph Builder (Log2Graph)

The **Log2Graph** C++ module provides high-performance real-time monitoring, parsing, and structuring of network traffic data into graph format, bridging raw Zeek logs with graph-based anomaly detection.

2.1.1 Core Classes and Their Roles

- `GraphBuilder` **(Singleton):** Orchestrates graph-building, initializes network traffic graphs, processes new connections, and maintains network state representation.
- `TrafficGraph`**:** Represents the complete network traffic graph with nodes (`GraphNode`) and connections (`AggregatedGraphEdge`). Manages node/edge operations and computes graph attributes for anomaly detection.
- `GraphNode` **and** `AggregatedGraphEdge`**:** Fundamental building blocks where:
  - `GraphNode` represents network endpoints (IP addresses) with dynamic attributes like degree, connection counts, and protocol statistics
  - `AggregatedGraphEdge` represents aggregated connections between nodes based on protocol, service, and destination port, capturing bytes/packets transferred, timestamps, and connection counts
- `LogMonitor` **and** `ZeekLogParser`**:** Handle log data ingestion through continuous monitoring of Zeek directories and extraction of connection details.

- `GraphExporter`: Exports traffic graphs to Graphviz DOT format for visualization and Python module processing.

2.1.2 Workflow: From Logs to Graphs

1. **Log Monitoring:** `LogMonitor` continuously monitors Zeek log directories for new entries
2. **Log Parsing:** `ZeekLogParser` extracts connection information from Zeek logs
3. **Graph Update:** `GraphBuilder` updates `TrafficGraph` structure with new nodes/edges
4. **Graph Export:** `GraphExporter` periodically outputs DOT files for Python processing

2.2 Graph Monitor (Python Script)

The **Graph Monitor** serves as the analytical core, consuming graph representations from `Log2Graph` and employing a GAT autoencoder for online learning of normal network patterns and anomaly identification based on reconstruction errors.

2.2.1 Core Modules and Their Roles

- `main.py`: Primary entry point handling command-line arguments, logging setup, and continuous monitoring loop initialization.
- `workflow.py`: Encapsulates high-level operational logic, coordinating graph loading, GNN processing, anomaly detection, and exports while managing `train` vs `detect` mode operations.
- `neural_net.py`: Defines the GAT-based autoencoder with:
  - **Encoder:** Learns compressed node representations using attention mechanisms
  - **Decoder:** Reconstructs original features from embeddings
  - **Loss Function:** Minimizes reconstruction error (high error indicates anomalies)
- `evaluate.py`: Assesses GNN performance and feature quality, rating features based on anomaly detection utility for continuous improvement.

2.2.2 Workflow: From Graphs to Anomalies

1. **Graph Input:** Monitor directory for new `.dot` files from `Log2Graph`
2. **Graph Processing:** Convert DOT files to PyTorch Geometric objects with feature normalization
3. **GNN Processing (Mode-Dependent):**
   - **Train mode:** Update model parameters through backpropagation for online learning

- **Detect mode:** Run inference to generate reconstructions without parameter updates

4. **Anomaly Detection:** Calculate reconstruction errors and flag anomalies exceeding thresholds
5. **Evaluation:** Assess model performance and feature contributions
6. **Visualization/Checkpointing:** Export visualizations and save model states periodically

2.2.3 Train vs. Detect Mode

Train Mode (`--mode train`):

- **Purpose:** Teach GNN what constitutes normal network behavior
- **Initial Training:** Extensive parameter adjustment (e.g., 50 epochs) to minimize reconstruction loss
- **Online Updates:** Continuous adaptation with limited optimization steps (e.g., 25 steps) to handle evolving patterns
- **Statistical Adaptation:** Update running statistics for consistent feature normalization
- **Outcome:** Robust autoencoder capable of accurately reconstructing normal network graphs

Detect Mode (`--mode detect`):

- **Purpose:** Identify deviations from learned normal patterns in real-time
- **Model Loading:** Load pre-trained checkpoint and statistical parameters
- **Inference:** Process graphs without backpropagation, calculating reconstruction errors
- **Anomaly Scoring:** High reconstruction error indicates anomalies; scores compared against thresholds
- **Outcome:** Real-time anomaly identification and logging based on deviation from normal patterns

Critical Dependency: Detect mode effectiveness depends on comprehensive training with representative normal traffic data. Insufficient training leads to high false positive/negative rates, requiring dedicated training periods before detection deployment.

3 Graph Attention Network Architecture

The cornerstone of our anomaly detection system is a sophisticated Graph Attention Network (GAT) based autoencoder designed to process and learn intricate patterns within dynamic network graphs. This section provides a comprehensive exposition of the architecture, beginning with fundamental concepts for readers less familiar with graph neural networks and deep learning methodologies, while maintaining rigorous mathematical formulations throughout. The overall architecture is visually summarized in Figure 5.

3.1 Foundational Concepts and Motivation

Graph Neural Networks in Context Traditional neural networks operate on Euclidean data structures with fixed topology, such as images represented as regular grids or sequences with linear ordering. However, many real-world systems exhibit irregular, non-Euclidean structures best represented as graphs $G = (V, E)$, where V denotes the vertex set and E represents the edge set. Graph Neural Networks (GNNs) extend deep learning to these irregular domains by designing message-passing mechanisms that aggregate information from local neighborhoods while preserving permutation invariance—a critical property ensuring consistent results regardless of node ordering. Graph Convolutional Networks (GCNs) [5] are a prominent example of such networks.

Autoencoder Architecture for Anomaly Detection An autoencoder implements a dimensionality reduction and reconstruction paradigm through two primary components: an encoder function $f_{enc} : X \to Z$ that maps input features $X \in \mathbb{R}^{N \times F}$ to compressed embeddings $Z \in \mathbb{R}^{N \times d}$ (where $d \ll F$), and a decoder function $f_{dec} : Z \to \hat{X}$ that reconstructs the original features. The reconstruction error $||X - \hat{X}||$, when properly characterized, serves as an anomaly indicator under the assumption that normal patterns exhibit consistent reconstructability while anomalous patterns deviate significantly from learned representations.

Attention Mechanisms in Graph Contexts Attention mechanisms address the fundamental challenge of selective information processing by learning adaptive weights α_{ij} that determine the relative importance of neighboring nodes. In graph contexts, this translates to the mathematical formulation:

$$h_i^{(l+1)} = \sigma\left(\sum_{j \in \mathcal{N}_i \cup \{i\}} \alpha_{ij}^{(l)} W^{(l)} h_j^{(l)}\right)$$

where $h_i^{(l)}$ represents the hidden state of node i at layer l, and $\alpha_{ij}^{(l)}$ denotes the attention coefficient between nodes i and j. Graph Attention Networks (GATs) [8] are a prime example of GNNs leveraging this mechanism.

3.2 Overall Architecture Design

The NodeGNNAnomalyDetector implements a sophisticated dual-pathway anomaly detection framework that combines reconstruction-based and embedding-based approaches. This architectural choice addresses the inherent limitations of single-pathway systems: reconstruction-based methods may fail to detect anomalies that maintain feature consistency while exhibiting unusual relational patterns, while embedding-based approaches may lack interpretability in explaining anomaly sources. The system architecture, as illustrated in Figure 5, comprises four interconnected components, operating on graph data that can be efficiently handled using libraries like PyTorch Geometric [2]:

1. **GAT-based Encoder: Topological Feature Learning** The encoder $E : G \to Z$ maps input graphs to compressed embeddings while preserving both nodal attributes and topological relationships. The mathematical formulation ensures that:

$$Z = E(X, A) = \text{GAT}_L(\text{GAT}_{L-1}(...\text{GAT}_1(X, A)))$$

where A represents the adjacency matrix and GAT_l denotes the l-th GAT layer.

2. **Reconstruction Decoder: Feature Space Recovery** The decoder $D : Z \to \hat{X}$ implements the inverse mapping, attempting to recover original features from compressed representations:

$$\hat{X} = D(Z) = \text{MLP}_{\text{decode}}(Z)$$

Reconstruction fidelity serves as a primary anomaly indicator through the error metric:

$$\mathcal{L}_{\text{recon}} = \frac{1}{N} \sum_{i=1}^{N} \ell(X_i, \hat{X}_i)$$

3. **Anomaly Scoring MLP: Direct Pattern Recognition** The scoring function $S : Z \to [0, 1]^N$ provides explicit anomaly probabilities:

$$p_{\text{anomaly}} = S(Z) = \sigma(\text{MLP}_{\text{score}}(Z))$$

where σ denotes the sigmoid activation function.

4. **Online Learning Framework: Adaptive Model Evolution** The framework implements continuous learning through:

$$\theta_{t+1} = \theta_t - \eta_t \nabla_\theta \mathcal{L}(\mathcal{D}_{\text{replay}} \cup \mathcal{D}_{\text{current}})$$

where θ represents model parameters and η_t denotes the adaptive learning rate.

3.3 GAT Encoder Architecture

The encoder employs a multi-layer GAT architecture with sophisticated attention mechanisms designed to capture both local neighborhood patterns and global graph structure. The detailed architecture of the GAT encoder layers is presented in Figure 6. The mathematical foundation extends the standard GAT formulation to incorporate edge features, addressing the limitation of purely node-centric attention mechanisms.

Enhanced Attention Computation The attention mechanism computes coefficients α_{ij} through a three-component integration:

$$\alpha_{ij} = \frac{\exp(\text{LeakyReLU}(\mathbf{a}^T[\mathbf{W}\mathbf{h}_i||\mathbf{W}\mathbf{h}_j||\mathbf{W}_e\mathbf{e}_{ij}]))}{\sum_{k\in\mathcal{N}_i}\exp(\text{LeakyReLU}(\mathbf{a}^T[\mathbf{W}\mathbf{h}_i||\mathbf{W}\mathbf{h}_k||\mathbf{W}_e\mathbf{e}_{ik}]))}$$

where:

- $\mathbf{h}_i, \mathbf{h}_j$ represent node feature vectors
- $\mathbf{e}_{ij}$ denotes edge features between nodes i and j
- $\mathbf{W}, \mathbf{W}_e$ are learnable transformation matrices
- $\mathbf{a}$ is the attention parameter vector
- $||$ indicates vector concatenation

This formulation enables the attention mechanism to consider not only node similarities but also the characteristics of connecting edges, providing richer contextual information for anomaly detection.

Multi-Head Attention Implementation The system employs K attention heads to capture diverse relationship patterns:

$$\mathbf{h}_i^{(l+1)} = \sigma\left(\frac{1}{K}\sum_{k=1}^{K}\sum_{j\in\mathcal{N}_i}\alpha_{ij}^{(k)}\mathbf{W}^{(k)}\mathbf{h}_j^{(l)}\right)$$

where each head k learns distinct attention patterns, and the final representation aggregates information across all heads.

Layer-wise Transformations The encoder processes features through successive transformations: Layer 1: $\mathbf{H}^{(1)} = \text{GAT}(\mathbf{X}, \mathbf{A}; \mathbf{W}^{(1)}) \in \mathbb{R}^{N\times 64}$ Layer 2: $\mathbf{Z} = \text{GAT}(\mathbf{H}^{(1)}, \mathbf{A}; \mathbf{W}^{(2)}) \in \mathbb{R}^{N\times 32}$ where the dimensional reduction from 64 to 32 creates a compressed representation suitable for downstream processing.

Regularization Mechanisms The architecture incorporates multiple regularization strategies with mathematical formulations:

- **Batch Normalization:** Applied to each layer output $\mathbf{H}^{(l)}$:

$$\text{BN}(\mathbf{H}^{(l)}) = \gamma\frac{\mathbf{H}^{(l)} - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} + \beta$$

- **Dropout:** Implemented as multiplicative noise [7]:

$$\mathbf{H}_{\text{drop}}^{(l)} = \mathbf{H}^{(l)} \odot \mathbf{m}$$

where $\mathbf{m} \sim \text{Bernoulli}(1-p)$ with dropout probability $p = 0.2$.

- **Residual Connections:** When dimensional compatibility permits:

$$\mathbf{H}^{(l+1)} = \mathbf{H}^{(l)} + \text{GAT}(\mathbf{H}^{(l)}, \mathbf{A})$$

3.4 Decoder Architecture

The decoder implements a progressive feature reconstruction strategy through a series of fully connected layers with increasing dimensionality, as part of the reconstruction pathway shown in Figure 7. The mathematical formulation follows:

$$\hat{\mathbf{X}} = \text{Linear}_{64 \to F_N}(\text{ReLU}(\text{BN}(\text{Linear}_{128 \to 64}(\text{ReLU}(\text{BN}(\text{Linear}_{32 \to 128}(\mathbf{Z}))))))))$$

This architecture enables progressive detail recovery from compressed embeddings, with each layer contributing to the reconstruction fidelity. The batch normalization and ReLU activations ensure stable training dynamics and non-linear transformation capabilities.

Reconstruction Loss Functions The system supports multiple loss formulations to accommodate different data characteristics:

- **Mean Squared Error:** $L_{\text{MSE}} = \frac{1}{N \cdot F_N} \sum_{i=1}^{N} \sum_{j=1}^{F_N} (X_{ij} - \hat{X}_{ij})^2$
- **Mean Absolute Error:** $L_{\text{MAE}} = \frac{1}{N \cdot F_N} \sum_{i=1}^{N} \sum_{j=1}^{F_N} |X_{ij} - \hat{X}_{ij}|$
- **Huber Loss:** $L_{\text{Huber}} = \frac{1}{N} \sum_{i=1}^{N} \begin{cases} \frac{1}{2}(X_i - \hat{X}_i)^2 & \text{if } ||X_i - \hat{X}_i|| \leq \delta \\ \delta(||X_i - \hat{X}_i|| - \frac{1}{2}\delta) & \text{otherwise} \end{cases}$
- **Log-Cosh Loss:** $L_{\text{LogCosh}} = \frac{1}{N} \sum_{i=1}^{N} \log(\cosh(X_i - \hat{X}_i))$

The choice of loss function significantly impacts anomaly sensitivity, with MSE providing high sensitivity to large deviations and MAE offering robustness to outliers.

3.5 Anomaly Scoring MLP

The anomaly scoring component implements a dedicated neural network $S : \mathbb{R}^{32} \to [0, 1]$ that learns to map embeddings directly to anomaly probabilities. This approach complements reconstruction-based detection by learning explicit anomaly patterns from the embedding space, as part of the direct scoring pathway detailed in Figure 7.

Network Architecture The MLP follows a funnel-like structure:

$$S(\mathbf{z}) = \sigma(\mathbf{W}_3\text{ReLU}(\mathbf{W}_2\text{ReLU}(\text{BN}(\mathbf{W}_1\mathbf{z} + \mathbf{b}_1)) + \mathbf{b}_2) + \mathbf{b}_3)$$

where:

- $\mathbf{W}_1 \in \mathbb{R}^{64\times32}$: Input transformation
- $\mathbf{W}_2 \in \mathbb{R}^{32\times64}$: Hidden transformation
- $\mathbf{W}_3 \in \mathbb{R}^{1\times32}$: Output transformation
- σ denotes the sigmoid activation function

Supervised Learning Objective The MLP training employs **Focal Loss** to address class imbalance. Focal Loss is a specialized loss function designed to mitigate the challenge of **class imbalance** in classification tasks, particularly when one class significantly outnumbers the others. Traditional loss functions, such as cross-entropy, can be dominated by the contribution of numerous "easy" negative examples (i.e., non-anomalous instances that are correctly classified with high confidence). This can lead to inefficient learning, as the model primarily optimizes for these abundant, well-understood examples, potentially neglecting the minority class (anomalies). Focal Loss addresses this by adaptively re-weighting the loss contribution of each example. It down-weights the contribution of well-classified examples, thereby **focusing learning on misclassified or hard-to-classify examples**. This is achieved through a **modulating factor** $(1 - p_t)^\gamma$, where p_t is the model's predicted probability for the true class and $\gamma \geq 0$ is a configurable focusing parameter. A higher γ increases the down-weighting for easy examples. Additionally, an **alpha term** α_t balances the importance of positive and negative examples. This formulation ensures that the model pays greater attention to difficult-to-classify instances while down-weighting easy examples, which is crucial for effective anomaly detection where anomalies are inherently rare.

$$L_{\text{focal}} = -\alpha_t(1 - p_t)^\gamma \log(p_t)$$

where:

- $p_t = p$ if $y = 1$, otherwise $p_t = 1 - p$
- α_t balances positive/negative examples
- γ focuses learning on hard examples

3.6 Online Learning and Adaptation

The online learning framework implements sophisticated mechanisms for continuous adaptation while mitigating catastrophic forgetting through experience replay and statistical monitoring. The Adam optimizer

[4] is used for parameter updates during online learning, as illustrated in the overall architecture in Figure 5.

Replay Buffer Dynamics The replay buffer maintains a representative sample of past experiences through reservoir sampling:

$$P(\text{replace}) = \frac{|\mathcal{B}|}{|\mathcal{B}| + t}$$

where t represents the number of new samples encountered since buffer initialization.

Statistical Monitoring The system maintains running statistics using Welford's algorithm for numerical stability: Mean Update: $\mu_t = \mu_{t-1} + \frac{x_t - \mu_{t-1}}{t}$ Variance Update: $\sigma_t^2 = \sigma_{t-1}^2 + \frac{(x_t - \mu_{t-1})(x_t - \mu_t) - \sigma_{t-1}^2}{t}$

Adaptive Learning Rate Scheduling The system employs Cosine Annealing with Warm Restarts:

$$\eta_t = \eta_{\min} + \frac{1}{2}(\eta_{\max} - \eta_{\min})(1 + \cos(\frac{T_{\text{cur}}}{T_i}\pi))$$

where T_{cur} denotes the current epoch within restart period T_i, enabling periodic exploration of the parameter space.

3.7 Anomaly Detection Methodology

The final anomaly detection employs a dual-threshold approach combining statistical and learned thresholds, as a part of the overall anomaly detection decision process shown in Figure 5 and detailed within the dual pathways in Figure 7.

Statistical Threshold (Reconstruction-based) Nodes exceeding the statistical threshold are flagged as anomalous:

$$\text{Anomaly}_{\text{recon}} = \{i : \ell(X_i, \hat{X}_i) > \mu_{\text{recon}} + k \cdot \sigma_{\text{recon}}\}$$

where $k = 2.0$ provides approximately 95% coverage under Gaussian assumptions.

Learned Threshold (MLP-based) The MLP-based detection employs a probability threshold:

$$\text{Anomaly}_{\text{MLP}} = \{i : S(z_i) > \tau\}$$

where $\tau = 0.8$ represents the decision boundary.

Combined Decision Rule The final anomaly set combines both approaches:

$$\text{Anomaly}_{\text{final}} = \text{Anomaly}_{\text{recon}} \cup \text{Anomaly}_{\text{MLP}}$$

This union-based approach maximizes sensitivity while maintaining interpretability through the dual-pathway framework. The comprehensive architecture enables robust, adaptive anomaly detection suitable for dynamic network monitoring scenarios while preserving mathematical rigor and interpretability through the dual detection pathways. The integration of sophisticated attention mechanisms, regularization techniques, and online learning capabilities ensures reliable performance across diverse network conditions and evolving anomaly patterns.

4 Network Graph Features and Representation

The effectiveness of graph-based anomaly detection depends on rich, relevant features describing network nodes and edges. The `Log2Graph` module extracts raw connection-level features from Zeek logs, while the Python-based `Graph Monitor` performs **feature aggregation and enhancement** at the node level. This two-stage process creates comprehensive node attributes capturing behavioral patterns over time for GNN-based anomaly detection.

4.1 Node Features (Enhanced Aggregations)

Nodes represent network entities (IP addresses). The `Graph Monitor` functions (`aggregate_node_data` and `extract_node_features_improved`) process raw connection data to create aggregated attributes that capture behavioral profiles:

Behavioral Role and Connection Dynamics:

- `outgoing_ratio/incoming_ratio`: Proportion of outgoing/incoming connections indicating client/server behavior
- `server_score/client_score`: Derived scores for predominant role classification
- `total_connections`, `outgoing_connections`, `incoming_connections`: Raw connection counts
- `most_freq_conn_state`: Most frequent connection state (established, denied, incomplete)

Protocol and Service Diversity:

- `unique_protocols`: Number of distinct protocols used
- `protocol_entropy`: Shannon entropy of protocol distribution
- `most_freq_proto_ratio`: Dominance of most frequent protocol

Port Usage Patterns:

- `unique_remote_ports/unique_local_ports`: Count of distinct ports accessed/offered

- `remote_port_diversity/local_port_diversity`: Port diversity ratios
- `privileged_remote_ratio/privileged_local_ratio`: Proportion of privileged port usage

Temporal Patterns:

- `first_seen_hour_minute_sin/cos`: Cyclical features for daily rhythm analysis
- `last_seen_hour_minute_sin/cos`: Last activity timing patterns

Application Layer Statistics:

- `http_ratio/ssl_ratio`: Proportion of HTTP/SSL connections

Traffic Volume Metrics:

- `total_outgoing_bytes/packets`: Bytes/packets sent
- `total_incoming_bytes/packets`: Bytes/packets received

4.2 Edge Features

Edges represent communications between nodes, capturing connection details directly from Zeek logs. As explained in the previous section, connections i.e. edges are aggregated based on source and dest ip, protocol, service and port.

Connection Metadata:

- `protocol`: Transport layer protocol (e.g., "tcp")
- `service`: Application service (e.g., "ssl", "http", "dns")
- `dst_port`: Destination port number
- `count`: Connection frequency between nodes

Traffic Volume & Packet Counts:

- `total_orig_bytes/total_resp_bytes`: Bytes sent by originator/responder
- `total_orig_pkts/total_resp_pkts`: Packets sent by originator/responder
- `total_orig_ip_bytes/total_resp_ip_bytes`: IP layer bytes including headers

4.3 Feature Extraction Process

The feature population involves two stages:

1. **Raw Feature Extraction (Log2Graph C++):** Creates basic nodes/edges with raw log information and performs initial node updates
2. **Feature Aggregation (Python Graph Monitor):** Aggregates connection-level data and derives behavioral metrics through ratio calculations, diversity measures, and scoring functions

This dual-stage approach efficiently handles raw data parsing in C++ and leverages Python for sophisticated feature engineering.

5 Evaluation

5.1 Dataset Introduction

We define the following criteria for the suitable dataset for our evaluation:

1. The dataset must comprise of both benign and anomalous traffic, since our system requires the benign one for base-lining.
2. The dataset must include a variety of network attacks, covering both unencrypted and encrypted traffic.
3. The dataset must be captured from a network that simulates realistic Small Office/Home Office environment

After selection, we determined to use the Intrusion detection evaluation dataset (CIC-IDS2017) [6]. The dataset covers a range of attacks, including FTP Brute Forcing, SSH Brute Forcing, DoS, Heartbleed, Web Attacks, Infiltration, Botnet and DDoS. Such extensive coverage will be valuable in testing our system's capabilities. Another strength of CIC-IDS2017 is it realistically simulated network.

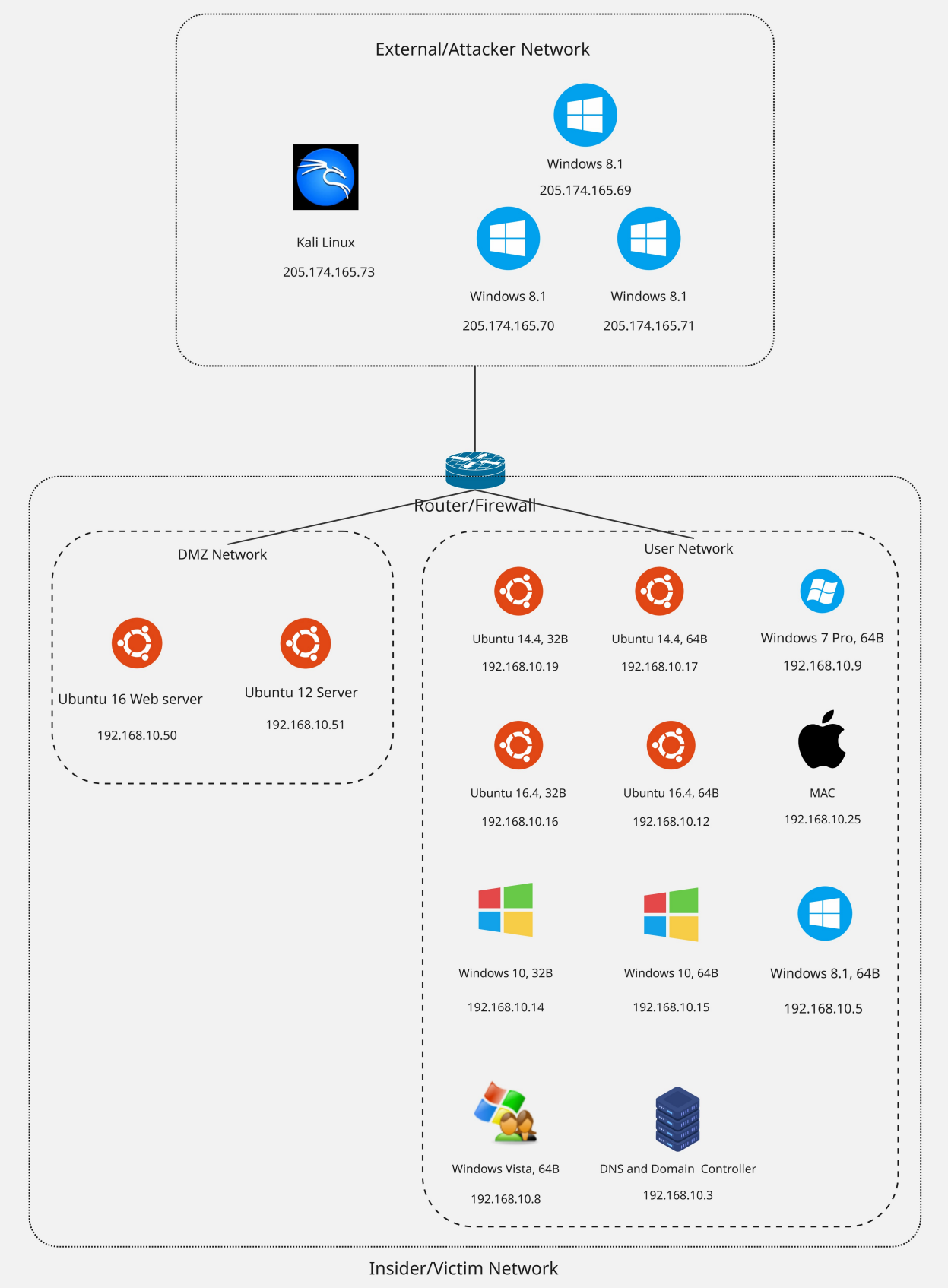

Figure 1: CIC-IDS2017 Dataset Network Setup

The network, with 18 devices being based on Windows, Linux and MacOS, is complex enough to mimic a real life small office network. Additionally, the authors have simulated 25 users' activities that involve various protocols like HTTP, HTTPS, FTP, SSH, and email protocols. The variety of devices and protocols will ensure a diverse set of captured traffic, allowing our AI detection system to observe and analyze network characteristics comprehensively.

The dataset includes the captured traffic within 5 days as PCAP format. While there is only benign traffic in the first day. there will malicious traffic made by different attacks in the remaining 4 days.

5.2 Dataset Processing

Our plan to process the dataset is as follows:

1. Train our AI detection system with benign traffic from the first captured day.
2. Extracted individual attack periods from the daily PCAPs into separate files to evaluate detection performance on each specific attack type.

After extraction, we obtained 13 individual attack captures which will be fed into our detection system to evaluate whether it can successfully identify each attack type.

5.3 Evaluation Result

To evaluate our system, we deploy it on a Digital Ocean VPS. The specification for this VPS is: 32 GB Memory, 8 AMD vCPUs, and 400 GB Disk.

After installing our system, we use the 'tcpreplay' tool to send CIC-IDS2017 dataset's pcap files to a monitored network interface.

Considering our system's current limitation to node-level (IP address) anomaly detection, we define detection success as the flagging of either the attacking or victim host as anomalous. This limitation will be discussed more in the upcoming sections.

The detection results for the attacks can be summarized as follows:

Timestamp	Attack Classification	Source IP Address	Target IP Address(es)	Status
Tuesday 09:20 – 10:20	FTP Brute Force Attack (Patator)	205.174.165.73 (NAT: 172.16.0.1)	192.168.10.50	Detected
Tuesday 14:00 – 15:00	SSH Brute Force Attack (Patator)	205.174.165.73 (NAT: 172.16.0.1)	192.168.10.50	Detected
Wednesday 09:47 – 10:10	Denial of Service (Slowloris)	205.174.165.73 (NAT: 172.16.0.1)	192.168.10.50	Detected
Wednesday 10:14 – 10:35	Denial of Service (SlowHTTPTest)	205.174.165.73 (NAT: 172.16.0.1)	192.168.10.50	Detected
Wednesday 10:43 – 11:00	Denial of Service (HULK)	205.174.165.73 (NAT: 172.16.0.1)	192.168.10.50	Detected
Wednesday 11:10 – 11:23	Denial of Service (GoldenEye)	205.174.165.73 (NAT: 172.16.0.1)	192.168.10.50	Detected
Wednesday 15:12 – 15:32	SSL/TLS Heartbleed	205.174.165.73 (NAT: 172.16.0.1)	192.168.10.51	Detected
Thursday 09:20 – 10:00	Web Application Brute Force	205.174.165.73 (NAT: 172.16.0.1)	192.168.10.50	Detected
Thursday 10:15 – 10:35	Cross-Site Scripting (XSS)	205.174.165.73 (NAT: 172.16.0.1)	192.168.10.50	Detected
Thursday 10:40 – 10:42	SQL Injection Attack	205.174.165.73 (NAT: 172.16.0.1)	192.168.10.50	Detected
Thursday 14:19 – 14:35	Metasploit Framework Exploitation	205.174.165.73 (NAT: 172.16.0.1)	192.168.10.8	Detected
Thursday 14:53 – 15:00	System Infiltration (Cool Disk MAC)	205.174.165.73 (NAT: 172.16.0.1)	192.168.10.25	Detected
Thursday 15:04 – 15:45	Data Exfiltration (Dropbox)	205.174.165.73 (NAT: 172.16.0.1)	192.168.10.8	Detected
Friday 10:02 – 11:02	Botnet Command & Control (ARES)	205.174.165.73 (NAT: 172.16.0.1)	192.168.10.15 (Win 10), 192.168.10.9 (Win 7), 192.168.10.14 (Win 10), 192.168.10.5 (Win 8), 192.168.10.8 (Vista)	Detected
Friday 13:55 – 14:35	Network Port Scanning	205.174.165.73 (NAT: 172.16.0.1)	192.168.10.50	Detected
Friday 15:56 – 16:16	Distributed Denial of Service (LOIC)	205.174.165.69 - 71 (NAT: 172.16.0.1)	192.168.10.50	Detected

We can have some key observations after the evaluation. The system has certain strengths as follows:

- The NodeGNNAnomalyDetectorsuccessfully highlight the anomalous IPs, including the attacker (although in this case, the IP is the NAT address) and the victim. In our current system, anomaloyus IPs are highlighted thanks to their having high a Reconstruction Error or MLP Score.
- The NodeGNNAnomalyDetector can identify a wide range of popular attacks.

However, the system has a number of points that need improving:

- The NodeGNNAnomalyDetector's alerts are limited to indicating the suspicious IPs without giving further context, such as detected flows, types of attacks, or protocols. Therefore, further log enrichment is required to make alerts actionable.

- The NodeGNNAnomalyDetector suffers from a high rate of False Positives, as benign nodes are included in the alerts. At the momemnt, all nodes having Reconstruction Error or MLP Score bigger than 0 are included in the alert.
- The NodeGNNAnomalyDetector experiences latency in its detection. First, the system does not perform real-time, but period check for new Zeek logs (which is 1 minute be default). Then, the system requires some time to reconstruct the new flows and identify anomalous activities. In the meantime, average latency between the attacks and the detection time varies from 1 to 3 minutes.

The subsequent section will present our approach to addressing the above limitations via implementing a middle log processor. Although this component remains in early development, it provides a fundamental framework for future system improvements

6 Proposed Deployment

6.1 Motivation

While our NodeGNNAnomalyDetector demonstrates promising detection capabilities, several operational challenges limit its practical deployment:

- **Limited Alert Context**: Analysts cannot determine attack type, urgency, or required actions from raw anomaly scores.
- **High False Positive Rate**: System is still flagging a high number of benign IPs in its alerts, making a struggle for security analysts.
- **Integration Gap**: Standalone alerts lack correlation with network forensics, SIEM platforms, and automated response workflows essential for modern security operations.
- **Core Challenge**: The NodeGNNAnomalyDetector can identify unusual machines, but requires improvement to provide actionable security intelligence required for operational response.

To bridge the gap between our academic research and practical deployment, we propose a comprehensive architecture that transforms raw anomaly detection into actionable threat intelligence.

6.2 Introduction

Building upon the NodeGNNAnomalyDetector framework, we developed an integrated detection platform that combines our anomaly detection model with complementary security services to create a comprehensive network threat analysis server.

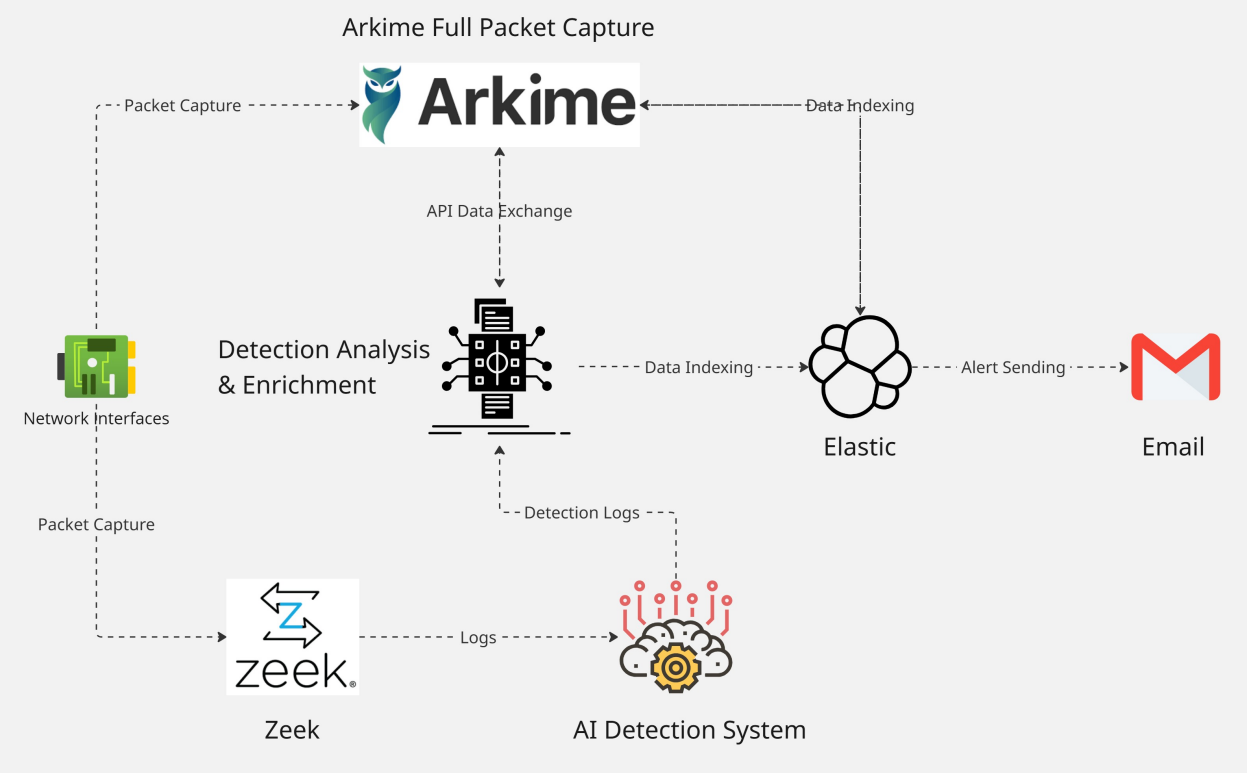

Figure 2: Detection Server Deployment

The design principles and goals of our proposed deployment are as follows:

- **Portability**: Docker containerization ensures the system can be easily deployed, transferred, and automated across different environments.
- **Explainable Detection**: Before being sent to analysts via email, alerts are parsed and enriched to ensure context clearance.
- **Scalability**: Modular architecture allows independent scaling of components based on network load and analysis requirements.
- **False Positive Reduction**: Implements intelligent filtering and correlation logic to eliminate the high false positive rates inherent in unsupervised AI detection systems.

Our deployment server comprises of the following modules, which run within separate docker containers:

1. **Zeek**: Zeek is a network monitoring service that continuously monitors traffic on the server's network interface to generate network-level logs. These logs are subsequently consumed by the NodeGNNAnomalyDetector for anomaly detection.
2. **Arkime**: Arkime is a full packet capture service that stores all network traffic sent from and to the server's interface for later retrieval and analysis. Arkime provides API access for extracting valuable network information.

3. **NodeGNNAnomalyDetector**: The AI-based detection service that identify anomalous IPs based on their network behaviours.

4. **Detection Analysis & Enrichment**: This service parses NodeGNNAnomalyDetector's alerts to extract key information. It filter out potentially false-positive warnings by using static threshold values. It will then enrich alerts with vital contextual information to aid analysts' investigation.

5. **Elastic Stack**: The Elastic Stack stores and indexes logs from Arkime and Detection Analysis & Enrichment.

6. **Elast Alert**: Elast Alert monitors Elastic Stack logs and send alerts to a specified destination, which will be analysts' emails in this projects.

6.3 Detection Analysis & Enrichment

6.3.1 Introduction

The Detection Analysis & Enrichment service act as a middle-man between the NodeGNNAnomalyDetector and the final detection reporting service (Elast Alert). It provides the following features:

- **NodeGNNAnomalyDetector's Alerts Parsing**: Given an alert file from NodeGNNAnomalyDetector, The Detection Analysis & Enrichment service will sort the nodes in terms of calculated risk scores. After that, it will filter for the nodes whose scores exceed a configured threshold. The filtering aims to reduce false positives.

- **Alert Contextual Enrichment**: Given a set of suspicious nodes (IPs), The Detection Analysis & Enrichment service will query Arkime's data to retrieve traffic information about each node, including the number of incoming/outgoing connections and unique connected IPs. This is to provide more context for analysts.

- **Log Saving**: The Detection Analysis & Enrichment service's logs will be sent to Elasticsearch, and later sent to analysts' emails thanks to ElastAlert.

Each NodeGNNAnomalyDetector's alert that is sent to Detection Analysis & Enrichment service follows a structured enrichment process: (1) Parse JSON-based alert file and calculate composite scores, (2) Filter false positives and select top-priority threats, (3) Query Arkime API for network context, (4) Generate timestamped JSON output for downstream processing and reporting.

The Detection Analysis & Enrichment service aims to address the current limitations of the NodeGNNAnomalyDetector while proposing a practical integration workflow for AI detection tools within existing SOC infrastructures.

6.3.2 Components Description

To facilitate its log analysis and enrichment, the Detection Analysis & Enrichment service relies on a number of components.

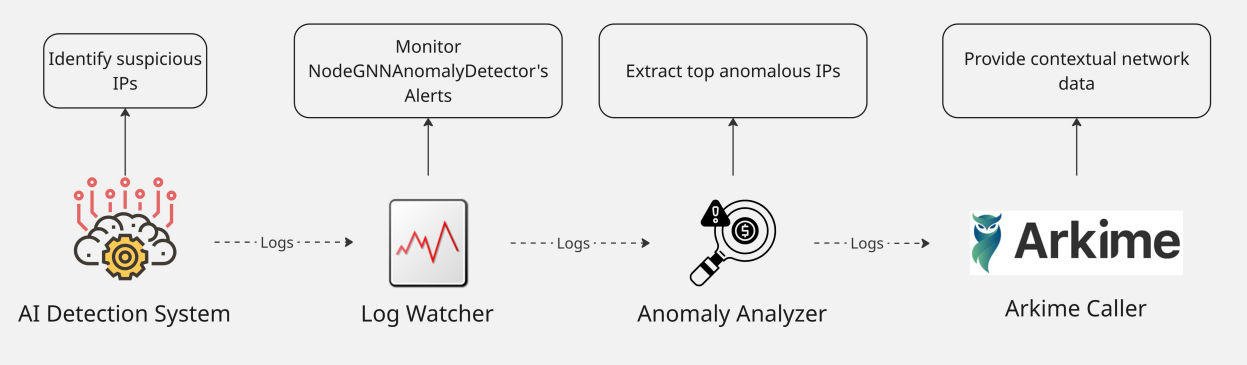

Figure 3: Detection Analysis & Enrichment Components

- **Log Watcher**: The Log Watcher is responsible for continuously monitoring NodeGNNAnomalyDetector's alert output using the Python Watchdog library. Whenever a new alert is found, it triggers the Anomaly Analyzer for further analysis. The implementation uses a producer-consumer pattern where file detection (fast) is decoupled from alert processing (slow) through a thread-safe queue, ensuring zero alert loss during high-activity periods.

- **Anomaly Analyzer**: The Anomaly Analyzer processes alert files from the NodeGNNAnomalyDetector by calculating composite risk scores for flagged IPs, filtering false positives using configurable thresholds, and selecting the highest-risk IP for detailed enrichment. The composite score is calculated using the formula: $0.8 \times \textit{Reconstruction Score} + 0.2 \times \textit{MLP Score}$. These weightings were determined through empirical analysis during simulated attack evaluation (detailed in Section 5). While not eliminating all false positives, this approach substantially reduces their occurrence.

- **Arkime Caller**: The Arkime Caller enriches alerts by querying Arkime's packet capture database for network behavioral context. Given a suspicious IP flagged by NodeGNNAnomalyDetector, it retrieves activity summaries including connection counts (incoming/outgoing), unique communication peers, and traffic volume patterns. This network intelligence helps analysts understand the scope and nature of anomalous behavior, enabling faster threat assessment and response decisions.

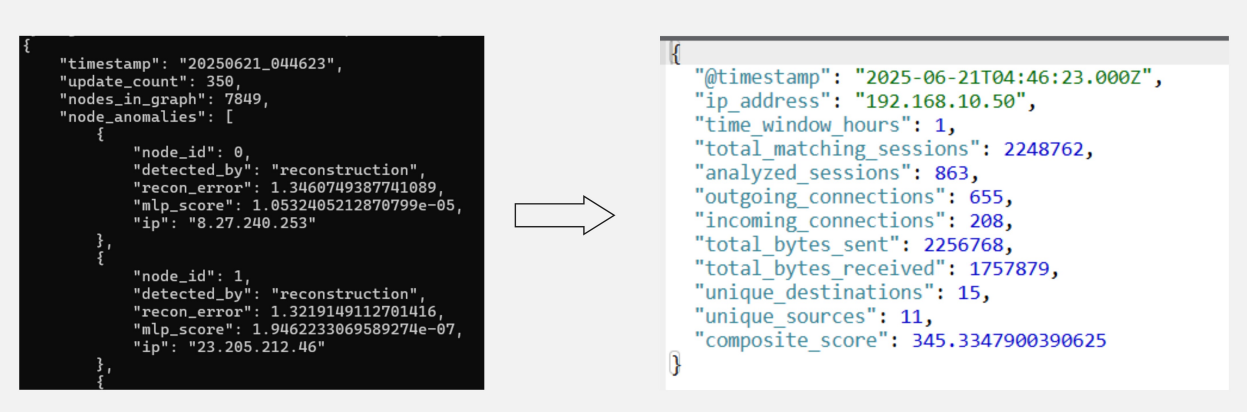

Figure 4: Enriching Raw Alerts with Network Contextual Data

7 Conclusion

In this project, we proposed a novel machine learning-based network attack detection system. Our system employs a dual-module architecture consisting of a high-performance C++ component for efficient network graph construction and a Python module leveraging Graph Attention Networks (GATs) to identify anomalous traffic patterns. We have also demonstrated the practical deployment of our detection system through automated alert processing and integration with network monitoring tools, enabling real-time assistance for security analysts. Our system demonstrates strong potential when evaluated against the CIC-IDS2017 dataset, successfully alerting on all suspicious IPs across all test scenarios.

To make our system ready for real-world use cases, enhancements should be made in these aspects:

- **False Positive Reduction**: The current high false positive rate needs addressing via improving threshold adaptation mechanisms or fine-tuning AI detection logic.
- **Flow-level Detection**: The system should highlight anomalous flows, instead of merely flagging IP addresses. Successful flow identification will allow analysts to swiftly pinpoint attack origins and nature.
- **Real-time Processing**: Detection latency can be reduced through streaming graph updates and optimized feature extraction workflow.

References

[1] Ocheme Anthony Ekle and William Eberle. Anomaly detection in dynamic graphs: A comprehensive survey. *ACM Transactions on Knowledge Discovery from Data*, 18(8):1–44, July 2024.

[2] Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric, 2019.

[3] Chuanpu Fu, Qi Li, and Ke Xu. Detecting unknown encrypted malicious traffic in real time via flow interaction graph analysis, 2023.

[4] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[5] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks, 2017.

[6] Iman Sharafaldin, Arash Habibi Lashkari, Ali A Ghorbani, et al. Toward generating a new intrusion detection dataset and intrusion traffic characterization. *ICISSp*, 1(2018):108–116, 2018.

[7] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.

[8] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks, 2018.

[9] Li Zheng, Zhenpeng Li, Jian Li, Zhao Li, and Jun Gao. Addgraph: Anomaly detection in dynamic graph using attention-based temporal gcn. In *IJCAI*, volume 3, page 7, 2019.

A Appendix

A Overall GAT-based Anomaly Detection Architecture

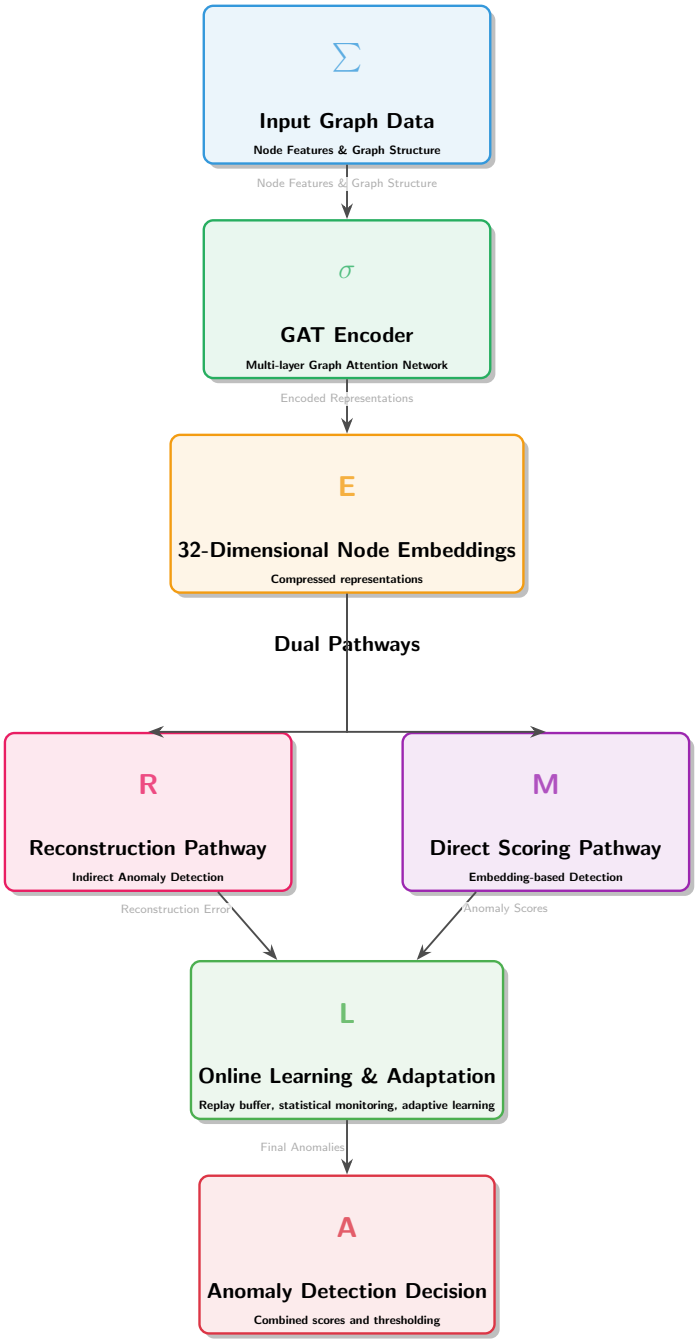

Figure 5: Complete GAT-based Anomaly Detection Architecture showing the pipeline from input data to anomaly detection decision. The architecture consists of a GAT encoder, dual detection pathways, and online learning components.

B Detailed GAT Encoder Architecture

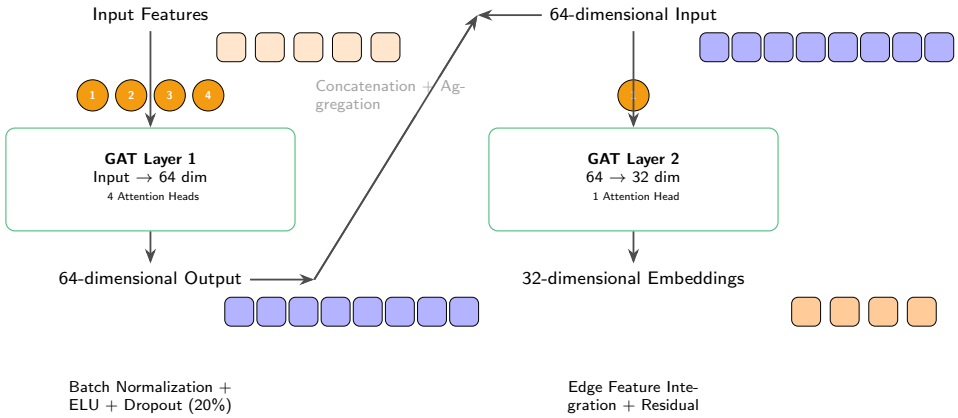

Figure 6: Detailed view of the two GAT layers in the encoder. Layer 1 uses multi-head attention to process input features, while Layer 2 produces the final embeddings with single-head attention and residual connections.

C Dual Pathway Decoders and Anomaly Detection

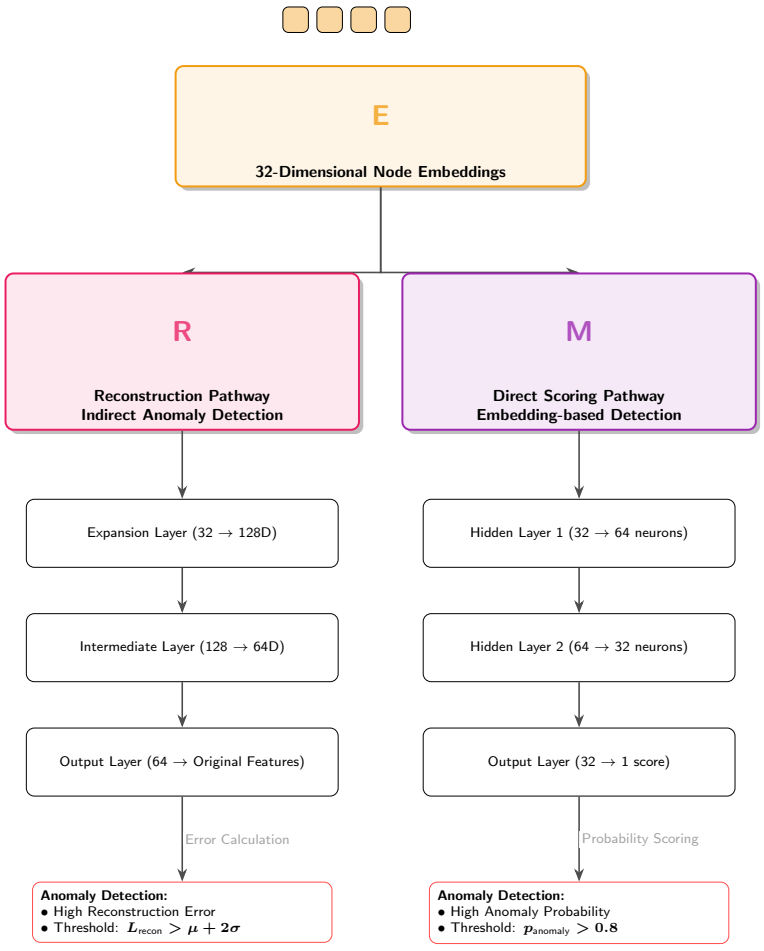

Figure 7: Comparison of the two anomaly detection pathways. The reconstruction pathway (left) measures feature reconstruction error, while the direct scoring pathway (right) computes anomaly probabilities through an MLP.