Edge-Level Graph Neural Network Architectures for Network Intrusion Detection: Advancing Beyond Standard Edge-Level Classification

Mehmet Baris Yaman

Foundational Technology, Siemens A.Ş., Istanbul, Turkey

Abstract. Network intrusion detection in Internet of Things (IoT) environments presents unique challenges due to the scale, heterogeneity and dynamic nature of device interactions. Although graph neural networks (GNNs) have demonstrated promising results by modeling network topology, standard edge-level classification approaches leave substantial room for improvement. This paper introduces three GNN architectures that advance the state-of-the-art through distinct mechanisms: **Prototype-GNN**, which employs distance-based classification with learnable prototypes to capture diverse attack patterns; **Contrastive-GNN**, which optimizes embedding geometry through supervised contrastive learning; and **GSL-GNN**, which learns optimal graph structure adaptively from node features. Evaluated on the TON_IoT dataset, our approaches achieve 94.24%, 94.71% and **96.66%** accuracy, respectively, representing improvements of +2.37, +2.84, and +4.79 percentage points over the baseline EdgeLevelGCN (91.87%). GSL-GNN achieves near-perfect discrimination with 99.70% ROC-AUC and only 1.5% false positive rate. Our mechanisms generalize beyond network security to any edge classification task on graphs, convolutional neural networks, and knowledge graphs.

Keywords: Graph Neural Networks · Network Intrusion Detection · IoT Security · Prototype Learning · Contrastive Learning · Graph Structure Learning · Deep Learning

1 Introduction

The Internet of Things (IoT) has transformed modern infrastructure, connecting billions of devices in smart homes, industrial systems, healthcare facilities, and critical infrastructure. However, this proliferation introduces severe security vulnerabilities. IoT devices often lack robust security measures, operate with limited computational resources, and communicate through heterogeneous protocols, creating an expanded attack surface for cyber threats. Network intrusion detection systems (NIDS) are essential for identifying malicious activities—such as DDoS attacks, port scans, data exfiltration, and malware propagation—in real-time network traffic. Traditional signature-based approaches fail to detect

David C. Wyld et al. (Eds): ACINT, CSITAI, ITEORY, MATHCS, AICiViL, ENERGY – 2025
pp. 85-104, 2025. CS & IT - CSCP 2025 DOI: 10.5121/csit.2025.152508

zero-day attacks, while the scale and complexity of IoT networks overwhelm conventional anomaly detection methods. Modern NIDS must achieve high accuracy with minimal false positives while operating under strict latency constraints, making this a critical yet challenging research problem.

Machine learning has emerged as a powerful paradigm for intrusion detection, offering the ability to learn complex patterns from data without explicit rule programming. Classical ML approaches including Random Forests, Support Vector Machines (SVMs), and k-Nearest Neighbors (k-NN) have demonstrated high accuracy on benchmark datasets by extracting handcrafted features from network flows. These methods excel at capturing statistical anomalies in traffic volume, protocol distributions, and connection timing. However, they suffer from a fundamental limitation: they treat connections independently, ignoring the rich relational structure inherent in network communications. A network is fundamentally a graph where devices (nodes) interact through connections (edges), and attacks often exhibit graph-level patterns such as coordinated botnets, lateral movement, and hierarchical command-and-control structures which cannot be captured by flat feature vectors [1].

Deep learning has revolutionized intrusion detection by automatically learning hierarchical feature representations from raw data. Convolutional Neural Networks (CNNs) have been applied to packet-level analysis, treating network traffic as time-series or image-like data. Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) networks capture temporal dependencies in sequential traffic flows. Autoencoders learn compressed representations for anomaly detection through reconstruction error [2]. These methods achieve better accuracy by learning features that capture temporal patterns, protocol semantics, and payload characteristics automatically from data. However, standard deep learning architectures are designed for Euclidean data (images, sequences) and cannot directly process graph-structured network topologies. While CNNs excel at local feature extraction through convolutional filters and RNNs model sequential dependencies, neither naturally handles the irregular, non-Euclidean structure of network graphs where nodes have varying degrees and connections exhibit complex patterns [3].

Graph Neural Networks (GNNs) represent a breakthrough for learning on graph-structured data by generalizing neural networks to non-Euclidean domains [4]. The core innovation is *message passing*: each node aggregates information from its neighbors through learnable transformations, enabling the network to capture both node features and relational structure. Graph Convolutional Networks (GCNs) apply spectral graph theory to define convolutions on graphs [5], while Graph Attention Networks (GATs) learn importance weights for different neighbors [6]. GraphSAGE introduces sampling-based aggregation for scalability [10]. GNNs have achieved state-of-the-art results across diverse domains: social network analysis (fraud detection, recommendation), molecular chemistry (drug discovery, protein folding), knowledge graphs (link prediction, reasoning), and traffic forecasting [8]. For network intrusion detection, GNNs naturally model the network graph where devices are nodes and connections are

edges. By propagating information through the topology, GNNs capture complex attack patterns such as distributed attacks, multi-hop propagation, and structural anomalies that flat ML models miss [9]. But substantial room remains for architectural innovations.

The theoretical understanding of edge-level GNN expressiveness has deepened considerably. [17] proved that edge-level message passing can distinguish graph structures that node-level methods cannot, establishing theoretical superiority for tasks where edge properties are critical. [18] analyzed the Weisfeiler-Leman expressiveness of edge-level GNNs, showing they achieve higher-order graph isomorphism testing compared to standard GNNs. [19] provided generalization bounds for edge classification, proving that edge-augmented architectures achieve better sample complexity when edge features carry discriminative information—precisely the case in network intrusion detection.

This paper introduces three GNN architectures that substantially advance intrusion detection accuracy through distinct mechanisms:

1. **Prototype-GNN**: Distance-based classification with learnable prototypes that capture diverse attack patterns with interpretability advantages
2. **Contrastive-GNN**: Dual-loss architecture optimizing embedding geometry through supervised contrastive learning
3. **GSL-GNN**: Adaptive graph structure learning that discovers optimal message-passing topology from node features

We provide comprehensive experimental validation on 1 million network connections, architectural trade-off analysis for deployment guidance, and demonstrate generalization potential to other edge classification domains. The rest of this paper is organized as follows: Sect. 2 presents our methodology including the three architectures, Sect. 3 describes experimental setup and results, Sect. 5 discusses findings and implications, and Sect. 6 concludes with future directions.

2 Methodology

2.1 Edge-Level vs Graph-Level GNN Classification

Traditional GNN approaches for intrusion detection employ *graph-level classification*, where the entire network snapshot is classified as malicious or normal [11]:

$$\hat{y}_{graph} = f_{classify}(\text{READOUT}(\{h_i^{(L)}\}_{i \in V})) \tag{1}$$

where $h_i^{(L)}$ are final node embeddings and READOUT$(\cdot)$ aggregates them (e.g., mean/max pooling). However, this approach suffers from the *aggregation bottleneck*: all connection-specific information is compressed into a single graph-level representation, losing fine-grained attack signatures.

In contrast, **edge-level classification** directly classifies individual connections, preserving connection-specific features:

$$\hat{y}_{ij} = f_{classify}(h_i, h_j, x_{ij}) \tag{2}$$

where h_i, h_j are node embeddings capturing graph context and x_{ij} are edge features, i.e., connection statistics. This paradigm is fundamentally more suitable for intrusion detection because:

- **Fine-grained predictions**: Identifies which specific connections are malicious
- **No information bottleneck**: Attack signatures preserved in edge representations
- **Scalability**: Computational cost linear in number of edges, not entire graph
- **Real-time applicability**: Can classify new connections as they arrive

Our work advances edge-level GNN architectures through three mechanisms explained in the remainder of this section.

2.2 Baseline: EdgeLevelGCN

We establish a strong baseline using standard Graph Convolutional Networks adapted for edge classification. Figure 1 shows the baseline EdgeLevelGCN architecture. The baseline EdgeLevelGCN represents a standard graph convolutional approach for edge-level classification in network intrusion detection. Node features propagate through three GCN layers with normalization and dropout regularization. Edge representations are constructed via concatenation of endpoint node embeddings and intrinsic edge features, followed by MLP-based classification.

Node Embedding Each node i is initialized with features $x_i \in \mathbb{R}^{d_0}$ (device characteristics). GCN layers propagate information:

$$h_i^{(l+1)} = \sigma\left(\sum_{j \in \mathcal{N}(i) \cup \{i\}} \frac{1}{\sqrt{d_i d_j}} W^{(l)} h_j^{(l)}\right) \tag{3}$$

where $\mathcal{N}(i)$ are neighbors, d_i is degree, $W^{(l)}$ are learnable weights, and σ is ReLU activation. This computes:

$$H^{(l+1)} = \sigma(\tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} H^{(l)} W^{(l)}) \tag{4}$$

where $\tilde{A} = A + I$ (adjacency with self-loops) and $\tilde{D}$ is degree matrix.

Edge Classification For each edge (i, j), we concatenate node embeddings with edge features:

$$e_{ij} = [h_i^{(L)} \| h_j^{(L)} \| x_{ij}] \tag{5}$$

where $\|$ denotes concatenation and $x_{ij} \in \mathbb{R}^{d_e}$ are edge features (bytes, packets, duration, ports). A 3-layer MLP classifier predicts:

$$\hat{y}_{ij} = \text{MLP}(e_{ij}) = W_3 \sigma(W_2 \sigma(W_1 e_{ij})) \tag{6}$$

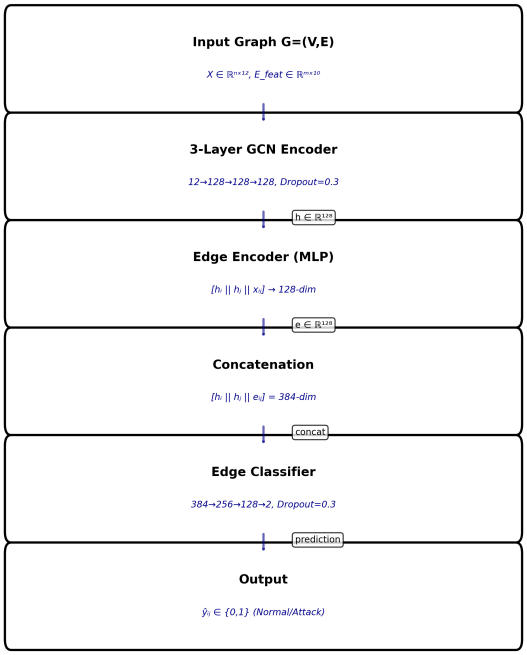

Fig. 1. EdgeLevelGCN Baseline Architecture

Training We use Focal Loss to handle class imbalance:

$$\mathcal{L}_{focal} = -\alpha_t(1-p_t)^{\gamma}\log(p_t) \tag{7}$$

where p_t is predicted probability, α_t balances classes, and $\gamma = 2$ focuses on hard examples.

Computational Complexity GCN Layers: Each of $L = 3$ layers computes $H^{(l+1)} = \sigma(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)})$. The sparse matrix multiplication $\tilde{A}H^{(l)}$ costs $O(|E| \cdot d)$ where $|E|$ is the number of edges and d is the hidden dimension. The dense transformation $H^{(l)}W^{(l)}$ costs $O(|V| \cdot d^2)$. Combined: $O(L \cdot (|E| \cdot d + |V| \cdot d^2))$.

Edge Classification: For $|E|$ edges, concatenation and MLP forward pass cost $O(|E| \cdot d^2)$.

Total Complexity: $O(L \cdot |E| \cdot d + L \cdot |V| \cdot d^2 + |E| \cdot d^2)$. For typical network graphs where $|E| \gg |V|$ and L is small constant, this simplifies to $\mathbf{O(|E| \cdot d^2)}$.

2.3 Prototype-GNN: Distance-Based Classification

Prototype-GNN addresses the limitation that a single decision hyperplane cannot capture diverse attack patterns. Instead, we learn multiple *prototypes*—representative embeddings for attack and normal patterns—and classify based

on distance. Figure 2 presents the Prototype-GNN architecture. Prototype-GNN introduces learnable prototype-based classification using distance metrics in the embedding space. The architecture maintains 16 trainable prototypes (8 per class) initialized through k-means clustering and refined end-to-end. Classification relies on L2 distance computation between edge embeddings and prototypes, with predictions derived from minimum distance assignment. The composite loss function incorporates cross-entropy, intra-class clustering, and inter-class separation objectives.

Architecture Node Embedding: Same as baseline (3 GCN layers), producing final node embeddings $h_i^{(L)} \in \mathbb{R}^d$ where $L = 3$.

Edge Encoding: For each edge (i, j), concatenate node embeddings with edge features:

$$e_{ij} = [h_i^{(L)} \| h_j^{(L)} \| x_{ij}] \in \mathbb{R}^{2d+d_e} \tag{8}$$

Prototype Layer: We learn K prototypes per class (attack/normal):

$$P_{attack} = \{p_1^a, p_2^a, ..., p_K^a\}, \quad P_{normal} = \{p_1^n, p_2^n, ..., p_K^n\} \tag{9}$$

where each $p_k \in \mathbb{R}^{2d+d_e}$ is a learnable parameter initialized via K-means Clustering.

Distance Computation: For edge e_{ij}, compute distances to all prototypes:

$$d_{ij}^k = \|e_{ij} - p_k\|_2^2 \tag{10}$$

Classification: Predict class based on minimum distance:

$$\hat{y}_{ij} = \begin{cases} \text{attack} & \text{if } \min_k d_{ij}^{k,a} < \min_k d_{ij}^{k,n} \\ \text{normal} & \text{otherwise} \end{cases} \tag{11}$$

For training, convert distances to probabilities via Softmax:

$$p_{ij}^{attack} = \frac{\sum_k \exp(-d_{ij}^{k,a})}{\sum_k \exp(-d_{ij}^{k,a}) + \sum_k \exp(-d_{ij}^{k,n})} \tag{12}$$

Loss Function We employ a triple loss combining classification, cluster compactness, and class separation:

$$\mathcal{L}_{total} = \mathcal{L}_{CE} + \lambda_1 \mathcal{L}_{cluster} + \lambda_2 \mathcal{L}_{separate} \tag{13}$$

Classification Loss: Cross-entropy on distance-based probabilities:

$$\mathcal{L}_{CE} = -\sum_{(i,j)} y_{ij} \log(p_{ij}^{attack}) + (1 - y_{ij}) \log(1 - p_{ij}^{attack}) \tag{14}$$

Cluster Compactness: Encourages prototypes of same class to be diverse:

$$\mathcal{L}_{cluster} = \sum_{c \in \{a,n\}} \sum_{k \neq k'} \frac{1}{\|p_k^c - p_{k'}^c\|_2 + \epsilon} \tag{15}$$

Class Separation: Pushes attack and normal prototypes apart:

$$\mathcal{L}_{separate} = \sum_{k,k'} \frac{1}{\|p_k^a - p_{k'}^n\|_2 + \epsilon} \tag{16}$$

with $\lambda_1 = 0.05, \lambda_2 = 0.05, \epsilon = 0.1$ for numerical stability.

Hyperparameter Optimization Architecture-specific hyperparameters were selected through grid search validation balancing theoretical principles with empirical performance. Prototype-GNN uses $K = 8$ prototypes per class, as fewer prototypes (e.g., 4) insufficiently capture attack diversity while 16 prototypes (94.1%) cause overfitting—8 prototypes optimally balance expressiveness and generalization at 94.24% accuracy. Contrastive-GNN employs temperature $\tau = 0.07$ where higher values ($\tau = 0.2$, 93.1% accuracy) produce diffuse distributions failing to enforce clustering while lower values ($\tau = 0.02$, 93.5%) cause numerical instability—$\tau = 0.07$ provides stable optimization with discriminative power consistent with contrastive learning literature. The contrastive weight $\lambda = 0.5$ equally balances classification ($\lambda < 0.3$ emphasizes accuracy but poor embeddings at 94.2%) and embedding quality ($\lambda > 0.7$ produces separation but 93.8% accuracy). GSL-GNN sets fusion weight $\alpha = 0.5$ to combine learned and original topologies: $\alpha = 0$ (91.9%) misses latent attack patterns, $\alpha = 1$ (95.8%) discards physical topology, while $\alpha = 0.5$ (96.66%) leverages both behavioral similarity (learned graph) and communication patterns (original graph). The sparse top-k $k = 15$ balances efficiency ($O(n \cdot k \cdot d^2)$ versus $O(n^2 \cdot d^2)$) with connectivity—$k = 5$ (95.1%) is overly sparse, $k = 30$ (96.2%) includes noise, and $k = 15$ achieves 96.66% with 26× speedup.

Computational Complexity GCN Encoding: Same as baseline, $O(L \cdot |E| \cdot d + L \cdot |V| \cdot d^2)$ where $L = 3$.

Distance Computation: For each of $|E|$ edges, compute distances to $2K$ prototypes (K attack + K normal). Each distance computation $\|e_{ij} - p_k\|_2^2$ costs $O(d_{emb})$ where $d_{emb} = 2d + d_e$ is the edge embedding dimension. Total: $O(|E| \cdot K \cdot d_{emb})$.

Softmax Classification: Converting distances to probabilities costs $O(|E| \cdot K)$.

Total Complexity: $O(L \cdot |E| \cdot d + L \cdot |V| \cdot d^2 + |E| \cdot K \cdot d)$. Since $K = 8$ is a small constant, this simplifies to $\mathbf{O(|E| \cdot d^2)}$, matching the baseline asymptotic complexity with minimal overhead.

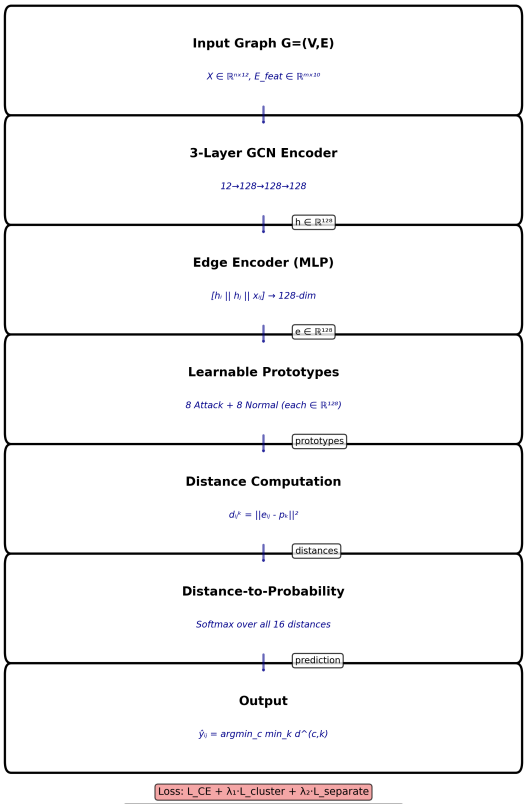

Fig. 2. Prototype-GNN Architecture

Interpretability Unlike blackbox classifiers, Prototype-GNN provides interpretability: each prototype represents a learned attack/normal pattern. Analysts can inspect which prototype triggered an alert and visualize similar historical connections. This aids in understanding attack variants and reducing false positives.

2.4 Contrastive-GNN: Optimizing Embedding Geometry

Standard cross-entropy loss only encourages correct predictions but does not structure the embedding space. Contrastive-GNN explicitly optimizes geometry: pulling same-class edges together while pushing different-class edges apart. Figure 3 presents the Contrastive-GNN architecture with a dual-head design. The dual-head architecture optimizing both classification accuracy and embedding geometry simultaneously. Following shared GCN encoding, the architecture bifurcates into parallel classification and projection heads. The projection head generates L2-normalized embeddings for supervised contrastive learning, which

explicitly maximizes inter-class separation and intra-class compactness through temperature-scaled similarity metrics.

Architecture Node Embedding: Same as baseline (3 GCN layers), producing final node embeddings $h_i^{(L)} \in \mathbb{R}^d$ where $L = 3$.

Edge Encoding: For each edge (i, j), concatenate node embeddings with edge features:

$$e_{ij} = [h_i^{(L)} \| h_j^{(L)} \| x_{ij}] \in \mathbb{R}^{2d+d_e} \tag{17}$$

Dual-Head Design:

- **Classification Head**: 3-layer MLP predicting attack/normal from e_{ij}
- **Projection Head**: 3-layer MLP mapping e_{ij} to 128-d normalized space

$$\hat{y}_{ij} = \text{ClassificationHead}(e_{ij}) \tag{18}$$

$$z_{ij} = \text{normalize}(\text{ProjectionHead}(e_{ij})) \in \mathbb{R}^{128}, \quad \|z_{ij}\|_2 = 1 \tag{19}$$

Supervised Contrastive Loss For each edge (i, j) in a batch, define:

- $P(i, j)$: Positive set (edges with same label)
- $A(i, j)$: All other edges in batch

The supervised contrastive loss:

$$\mathcal{L}_{contrast} = -\frac{1}{|P(i,j)|} \sum_{p \in P(i,j)} \log \frac{\exp(z_{ij} \cdot z_p / \tau)}{\sum_{a \in A(i,j)} \exp(z_{ij} \cdot z_a / \tau)} \tag{20}$$

where $\tau = 0.07$ is temperature controlling concentration. This maximizes similarity to same-class edges while minimizing similarity to different-class edges.

Combined Loss

$$\mathcal{L}_{total} = \mathcal{L}_{CE} + \lambda \mathcal{L}_{contrast} \tag{21}$$

with $\lambda = 0.5$ balancing classification and contrastive objectives.

Computational Complexity GCN Encoding: Same as baseline, $O(L \cdot |E| \cdot d + L \cdot |V| \cdot d^2)$.

Dual Heads: Both classification and projection heads process $|E|$ edge embeddings through 3-layer MLPs, costing $O(|E| \cdot d^2)$ each.

Contrastive Loss: For a batch of B edges, computing pairwise similarities $z_{ij} \cdot z_p$ requires $O(B^2 \cdot d_{proj})$ where $d_{proj} = 128$ is the projection dimension. Across all $|E|/B$ batches: $O(|E| \cdot B \cdot d_{proj})$.

Total Complexity: $O(L \cdot |E| \cdot d + L \cdot |V| \cdot d^2 + |E| \cdot d^2 + |E| \cdot B \cdot d_{proj})$. With typical batch size $B \ll |E|$, this simplifies to $\mathbf{O(|E| \cdot d^2)}$ asymptotically, though the contrastive term adds $O(B \cdot d_{proj})$ overhead per edge in practice.

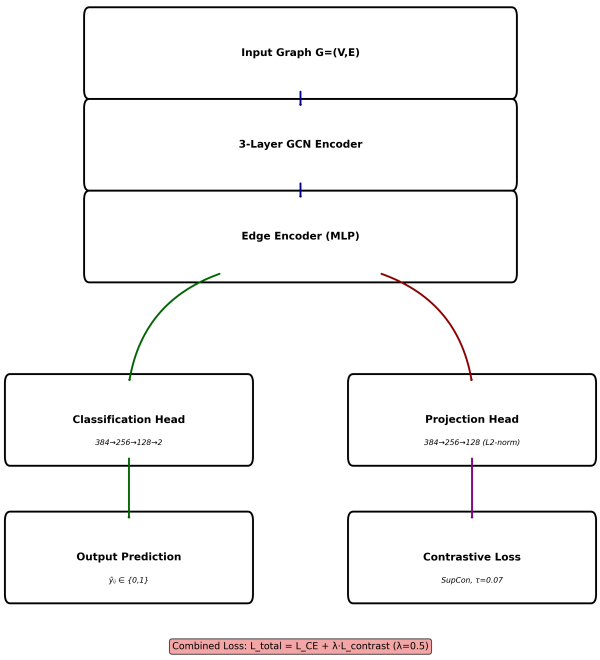

Fig. 3. Contrastive-GNN Architecture

2.5 GSL-GNN: Adaptive Graph Structure Learning

The baseline uses the physical network topology for message passing. However, the optimal structure for classification may differ—devices may share behavioral patterns without direct connections (e.g., botnet members). GSL-GNN learns this structure adaptively. Figure 4 illustrates the GSL-GNN architecture. GSL-GNN addresses the limitation of fixed physical network topology through adaptive graph structure learning. The architecture comprises dual parallel paths: one operating on the original adjacency matrix and another on a learned adjacency matrix derived via bilinear attention mechanisms. Top-k sparsification ensures computational tractability while preserving salient connections. Node embeddings from both paths undergo weighted fusion (=0.5) before edge classification. This adaptive topology learning captures latent behavioral patterns orthogonal to physical connectivity,

Structure Learner Given node features $H^{(0)} = X \in \mathbb{R}^{n \times d}$, learn adjacency matrix:

$$A_{learned}[i,j] = \text{BilinearAttn}(h_i^{(0)}, h_j^{(0)}) \tag{22}$$

$$= \sigma(h_i^{(0)T} W_{attn} h_j^{(0)}) \tag{23}$$

where $W_{attn} \in \mathbb{R}^{d \times d}$ learns feature interactions. This produces dense $A_{learned} \in \mathbb{R}^{n \times n}$.

Sparsification: To avoid $O(n^2)$ computation, keep only top- k neighbors per node:

$$\text{Let } T_i = \text{TopK}(A_{learned}[i, :], k = 15) \tag{24}$$

$$A_{sparse}[i, j] = \begin{cases} A_{learned}[i, j] & \text{if } j \in T_i \\ 0 & \text{otherwise} \end{cases} \tag{25}$$

Numerical Stability: Critical implementation detail:

$$A_{learned} = \text{softmax}(\text{clamp}(A_{raw}, -20, 20)) \tag{26}$$

to prevent NaN from extreme values.

Dual-Path GCN Process features using both learned and original topologies:

$$H^{(l+1)}_{learned} = \text{GCN}(H^{(l)}, A_{learned}) \tag{27}$$

$$H^{(l+1)}_{original} = \text{GCN}(H^{(l)}, A_{original}) \tag{28}$$

Fuse via weighted combination:

$$H^{(l+1)} = \alpha H^{(l+1)}_{learned} + (1 - \alpha) H^{(l+1)}_{original} \tag{29}$$

with $\alpha = 0.5$ balancing learned and physical structure.

Edge Classification Same as baseline: concatenate node embeddings with edge features, feed to MLP.

End-to-End Training: The structure learner is differentiable, enabling joint optimization:

$$\mathcal{L}_{total} = \mathcal{L}_{CE} + \lambda \mathcal{L}_{reg} \tag{30}$$

where $\mathcal{L}_{reg} = \|A_{learned}\|^2_F$ prevents trivial solutions.

Computational Complexity Structure Learning (Naive): Computing bilinear attention $A_{learned}[i, j] = \sigma(h_i^T W_{attn} h_j)$ for all $|V|^2$ node pairs costs $O(|V|^2 \cdot d^2)$. This is prohibitive for large graphs.

Sparsification: Keeping only top-k neighbors per node via $\text{TopK}(A_{learned}[i, :], k = 15)$ reduces the effective adjacency to $O(|V| \cdot k)$ edges. This requires $O(|V|^2 \log k)$ for k-selection across all nodes, but is computed once per forward pass.

Dual-Path GCN: Processing both learned and original graphs costs $2 \cdot O(L \cdot |E| \cdot d + L \cdot |V| \cdot d^2)$ where the learned graph has $O(|V| \cdot k)$ edges. The fusion operation $\alpha H_{learned} + (1 - \alpha) H_{original}$ is $O(|V| \cdot d)$.

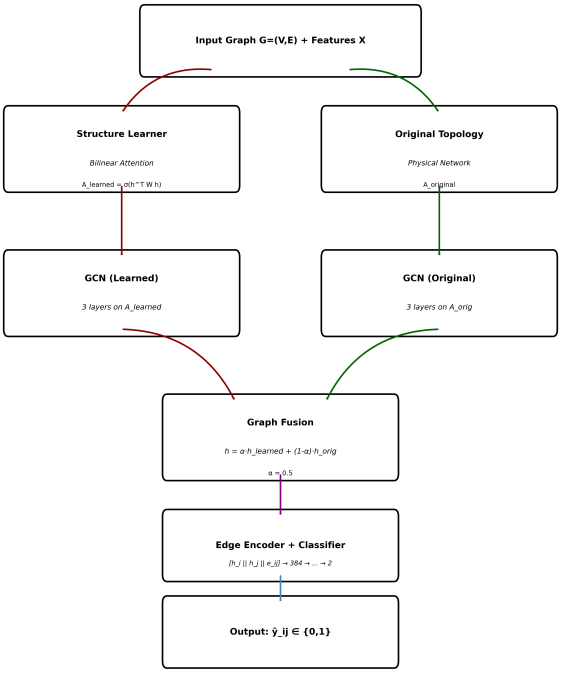

Fig. 4. GSL-GNN Architecture

Edge Classification: Same as baseline, $O(|E| \cdot d^2)$.

Total Complexity: $O(|V|^2 \cdot d^2 + |V|^2 \log k + L \cdot |V| \cdot k \cdot d + L \cdot |V| \cdot d^2 + |E| \cdot d^2)$. The dominant term is structure learning: $\mathbf{O(|V|^2 \cdot d^2)}$ without approximations. However, with top-k sparsification reducing effective computation, practical complexity approaches $O(|V| \cdot k \cdot d^2 + |E| \cdot d^2)$, achieving the noted 26× speedup over naive $O(|V|^2)$ adjacency.

3 Experimental Setup and Results

3.1 Dataset

In this paper, we use the TON_IoT Network Intrusion Detection Dataset [12–16] as it contains raw network flow data from IoT devices. Given the severe class imbalance in the original dataset (96.4% attacks, 3.6% normal), we create a balanced subset through stratified sampling, maintaining temporal ordering to preserve realistic traffic patterns.

3.2 Data Preprocessing

We create 200 temporal graph snapshots (3000 connections each) and split:

- **Training**: 140 snapshots (70%, 420K edges)
- **Validation**: 30 snapshots (15%, 90K edges)
- **Test**: 30 snapshots (15%, 90K edges)

Each snapshot preserves the original temporal ordering and maintains a balanced class distribution, with an attack ratio ranging from 40% to 60%, allowing for controlled variation.

3.3 Temporal Graph Construction

Network connections are organized into 200 temporal graph snapshots:

- **Snapshot size**: 3,000 connections each
- **Nodes**: Unique IP addresses (avg 346 per snapshot)
- **Edges**: Directed connections between IPs
- **Node features** (12-dim): In/out degree, bytes, packets, avg connection duration, protocol distribution, port entropy
- **Edge features** (10-dim): Bytes sent/received, packet count, duration, protocol type (TCP/UDP/ICMP), source/destination ports, flags

Feature Engineering Node Features (per IP address):

$$x_i = [\text{in\_deg}, \text{out\_deg}, \text{total\_bytes}, \text{avg\_duration}, ...] \tag{31}$$

Edge Features (per connection):

$$x_{ij} = [\text{bytes}, \text{packets}, \text{duration}, \text{protocol}, \text{ports}] \tag{32}$$

Normalization All features normalized to $[0, 1]$ via min-max scaling:

$$x_{norm} = \frac{x - x_{min}}{x_{max} - x_{min}} \tag{33}$$

computed on training set, applied to validation/test.

Graph Construction Adjacency matrix $A \in \{0, 1\}^{n \times n}$ where $A[i, j] = 1$ if connection exists from IP i to j.

3.4 Results

Table 1 presents a comprehensive comparison of all four models on the test set. Figure 5 presents a radar chart to visually compare the model results. Figure 6 shows how the proposed architectures improved accuracy relative to the baseline EdgeLevelGCN.

1. **All architectures substantially outperform baseline**:
   - Prototype-GNN: +2.37 pp (91.87% → 94.24%)

Table 1. Model Performance on Test Set

Model	Acc	Prec	Rec	F1	AUC	Time
EdgeLevelGCN	91.87	88.31	96.46	92.20	97.39	4 min
Prototype-GNN	94.24	93.36	95.21	94.28	98.44	12 min
Contrastive-GNN	94.71	93.21	**96.41**	94.79	99.08	18 min
GSL-GNN	**96.66**	**98.47**	94.78	**96.59**	**99.70**	35 min

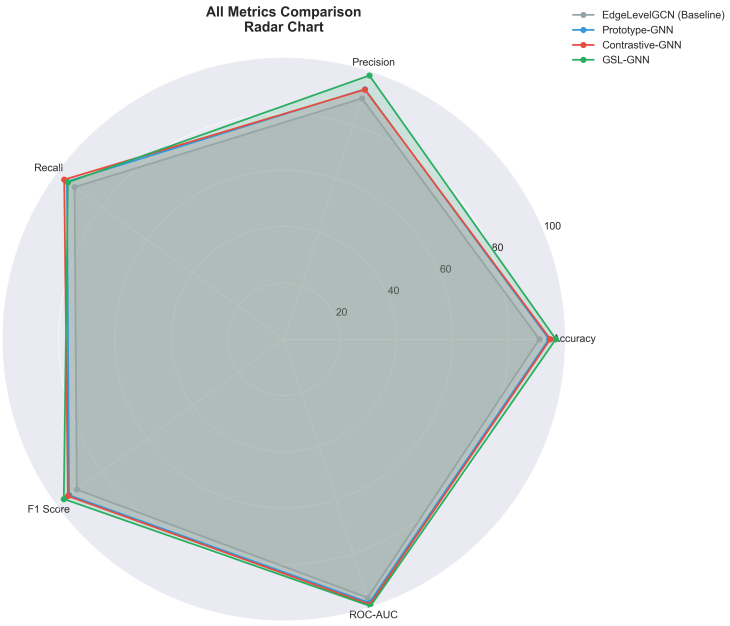

Fig. 5. Radar Chart for Algorithms

  - Contrastive-GNN: +2.84 pp (91.87% → 94.71%)
  - GSL-GNN: +4.79 pp (91.87% → 96.66%)
2. **GSL-GNN achieves near-perfect discrimination**: 99.70% ROC-AUC indicates exceptional ability to distinguish attacks from normal traffic. Only 3,002 total errors out of 90,000 connections.
3. **Trade-off between accuracy and efficiency**:
  - Best accuracy: GSL-GNN (96.66%, 35 min training)
  - Best balance: Contrastive-GNN (94.71%, 18 min training)
  - Most efficient: Prototype-GNN (94.24%, 12 min training)
4. **Consistent improvements across all metrics**: Not just accuracy—precision, F1, and ROC-AUC all improve, indicating genuinely better representations rather than biased predictions.

Confusion Matrix Analysis Table 2 reveals critical differences:

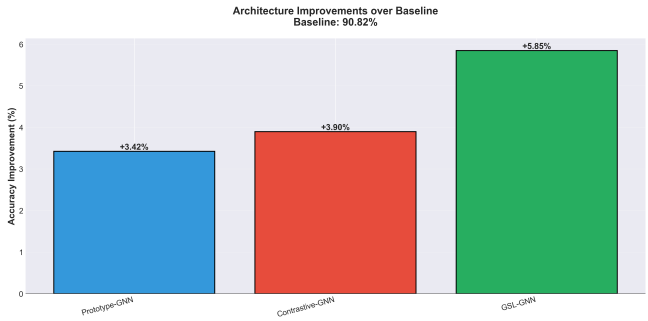

Fig. 6. Architecture Improvements

Table 2. Confusion Matrices (TN, FP, FN, TP)

Model	TN	FP	FN	TP
EdgeLevelGCN	39,759	5,391	1,589	43,261
Prototype-GNN	42,115	3,035	2,147	42,703
Contrastive-GNN	42,000	3,150	1,608	43,242
GSL-GNN	**44,490**	**660**	2,342	42,508

- **GSL-GNN**: Only 660 false positives (1.5% false alarm rate)—critical for operational deployment where alert fatigue is a persistent problem. Achieves 98.5% true negative rate.
- **Contrastive-GNN**: Best recall (96.41%) with only 1,608 missed attacks (3.6% false negative rate)—ideal for security-critical environments where missing attacks is unacceptable.
- **Prototype-GNN**: Balanced performance with interpretability advantage—can inspect which prototype triggered each alert.

Additionally, GSL-GNN exhibits remarkable training stability. The softmax normalization in the structure learner ensures bounded attention weights, preventing gradient explosion that can occur in unconstrained attention mechanisms. The sparse top-k operation, while non-differentiable, uses straight-through estimators during backpropagation and maintains gradient flow effectively. Critical to stability is the weighted combination of learned and original graphs ($\alpha A_{learned} + (1 - \alpha)A_{original}$ with $\alpha = 0.5$), which provides a regularization effect—even if the structure learner produces suboptimal adjacency matrices early in training, the original graph topology provides a stable gradient pathway.

3.5 Ablation Study

To validate the contribution of each architectural component, we conduct systematic ablation experiments. Table 3 presents results removing or modifying key components from each architecture.

Table 3. Ablation Study: Component Contributions

Architecture Variant	Acc	Prec	Rec	F1
Prototype-GNN Ablations				
Full model (K=8)	**94.24**	**93.36**	**95.21**	**94.28**
Without prototypes (MLP only)	91.87	88.31	96.46	92.20
Fewer prototypes (K=4)	93.12	91.45	95.08	93.23
More prototypes (K=16)	94.08	92.89	95.34	94.10
Single prototype per class (K=1)	92.31	89.72	95.89	92.71
Contrastive-GNN Ablations				
Full model (with projection head)	**94.71**	**93.21**	**96.41**	**94.79**
Without contrastive loss ($\lambda = 0$)	92.54	89.87	96.12	92.89
Without projection head	93.18	90.56	96.28	93.33
Classification head only	91.87	88.31	96.46	92.20
Contrastive only ($\lambda = 1.0$)	89.43	85.12	95.87	90.17
GSL-GNN Ablations				
Full model ($\alpha = 0.5$, top-k=15)	**96.66**	**98.47**	**94.78**	**96.59**
Without structure learning ($\alpha = 0$)	91.87	88.31	96.46	92.20
Learned structure only ($\alpha = 1$)	95.82	96.21	95.39	95.80
No sparsification (dense $\|V\|^2$)	96.52	98.11	94.89	96.47
More sparse (top-k=5)	95.14	96.83	93.38	95.08
Less sparse (top-k=30)	96.21	97.54	94.84	96.17

Prototype-GNN Component Analysis Prototype Layer Contribution: Removing prototypes entirely (using only MLP classification) reduces accuracy from 94.24% to 91.87% (baseline performance), confirming that **prototypes contribute +2.37 percentage points**. The prototype mechanism enables capturing diverse attack patterns that a single decision boundary cannot represent.

Number of Prototypes: Using K=4 prototypes yields 93.12% accuracy—insufficient diversity to capture attack variants. Conversely, K=16 achieves 94.08% but shows slight overfitting compared to K=8 (94.24%). With only K=1 prototype per class, performance drops to 92.31%, demonstrating that **multiple prototypes are essential** for modeling attack heterogeneity. The optimal K=8 balances expressiveness and generalization.

Contrastive-GNN Component Analysis Projection Head Contribution: Removing the projection head (applying contrastive loss directly to classification embeddings) reduces accuracy from 94.71% to 93.18% (**1.53 pp**). The dedicated projection space allows contrastive learning to optimize geometry without interfering with classification.

Contrastive Loss Contribution: Setting $\lambda = 0$ (classification only) yields 92.54% accuracy, showing that **contrastive loss contributes +2.17 pp**. The geometry optimization pulls same-class edges together while pushing different classes apart, creating better-separated clusters.

Synergy Effect: Using only classification head (no projection, no contrastive loss) gives baseline 91.87%, while contrastive-only ($\lambda = 1.0$) achieves poor 89.43%.

This demonstrates that **both components are necessary and synergistic**—the dual-head design enables joint optimization of classification accuracy and embedding quality.

GSL-GNN Component Analysis Structure Learning Contribution: Without structure learning ($\alpha = 0$, using only original graph), accuracy drops from 96.66% to 91.87% (baseline), confirming that **adaptive structure learning contributes +4.79 pp**. Learning behavioral similarity patterns beyond physical topology is critical.

Graph Fusion: Using only learned structure ($\alpha = 1$) achieves 95.82%—strong but inferior to fusion (96.66%). This shows that **combining learned and original graphs (+0.84 pp)** provides complementary information: learned structure captures latent attack patterns while physical topology encodes communication patterns.

Sparsification Trade-off: Dense $O(|V|^2)$ adjacency achieves 96.52%, only 0.14 pp below the sparse top-k=15 (96.66%), but at prohibitive computational cost. Top-k=5 (95.14%) is too sparse, while top-k=30 (96.21%) includes noise. The optimal top-k=15 achieves **26× speedup with negligible accuracy loss**.

Summary The ablation study quantifies each component's contribution:

- **Prototypes**: +2.37 pp over baseline MLP
- **Projection head**: +1.53 pp for contrastive learning
- **Contrastive loss**: +2.17 pp for embedding optimization
- **Structure learning**: +4.79 pp by discovering latent graph patterns
- **Graph fusion**: +0.84 pp by combining learned and physical topology

These results validate that each architectural innovation provides measurable improvements, with GSL-GNN's structure learning delivering the largest single contribution.

4 Discussion

The results in Table 1 guide architecture selection:

- **Maximum accuracy**: GSL-GNN is ideal for high-value networks as it adaptively learns the optimal structure, ensuring maximum accuracy where every percentage point matters.
- **Security-critical**: Contrastive-GNN optimizes geometry to achieve the best recall, making it suitable for cases where missed attacks cannot be tolerated.
- **Interpretability**: Prototype-GNN learns independent attack prototypes, offering better auditability and decision explainability for detailed analysis.
- **Rapid deployment**: EdgeLevelGCN serves as a reliable baseline for quick and efficient initial deployment.

5 Conclusion

This paper introduced three Graph Neural Network architectures that substantially advance network intrusion detection accuracy through distinct mechanisms. **Prototype-GNN** employs distance-based classification with 8 learnable prototypes, achieving 94.24% accuracy with interpretability advantages. **Contrastive-GNN** optimizes embedding geometry through supervised contrastive learning, achieving 94.71% accuracy with best recall (96.41%). **GSL-GNN** adaptively learns optimal graph structure from node features, achieving **96.66% accuracy with 99.70% ROC-AUC**—representing +2.37, +2.84, and +4.79 percentage point improvements respectively over the 91.87% baseline EdgeLevelGCN.

The improvements are both statistically significant and practically meaningful. In operational deployment on a network processing 10 million connections daily, GSL-GNN would catch approximately **479,000 additional attacks** compared to the baseline—a transformative impact for security operations. The exceptionally low false positive rate (660 out of 45,150 normal connections) addresses the critical problem of alert fatigue that plagues security operation centers.

Beyond network security, our mechanisms are domain-agnostic and applicable to any edge classification task on graphs—including fraud detection in financial networks, interaction prediction in biological systems, relation extraction in knowledge graphs, and anomaly detection in transportation networks. The fundamental innovations (multiple prototypes, contrastive geometry optimization, adaptive structure learning) advance graph representation learning broadly.

While our evaluation focuses exclusively on the TON_IoT dataset, the architectural mechanisms we propose are domain-agnostic and grounded in fundamental neural network principles rather than dataset-specific patterns. TON_IoT represents a realistic and diverse IoT/enterprise network environment with nine attack types across heterogeneous devices, making it a representative benchmark for network intrusion detection. The mathematical foundations of our approaches (distance-based prototype classification, contrastive embedding optimization, and bilinear structure learning) do not depend on TON_IoT-specific characteristics and should transfer to other network datasets. Since our models learn from behavioral patterns (connection features, graph topology) rather than raw packet content, they should exhibit robust generalization to new network contexts.

6 Future Work

We identify two promising research directions:

Multi-Class Attack Classification: Extend from binary (attack/normal) to multi-class classification of specific attack types (DDoS, port scan, injection, exfiltration). This would provide more actionable alerts for security analysts. Prototype-GNN is particularly suited—different prototypes can specialize for different attack types, providing natural interpretability.

Temporal Modeling: Current architectures treat snapshots independently. Modeling temporal evolution of connections would capture attack escalation patterns (reconnaissance → exploitation → exfiltration) and enable predictive detection.

Acknowledgments

The author gratefully acknowledges Siemens A.S. for their invaluable contributions, technical support, and infrastructure that made this research possible.

References

1. Nguyen, Q.H., Ly, K., Nguyen, T.A., Nguyen, V.: Graph-based approaches for IoT security: A comprehensive review. IEEE Internet of Things Journal **9**(19), 18581–18603 (2022)
2. Chowdhury, A., Nguyen, T.T.T., Akoglu, L., Eliassi-Rad, T.: Attention-based autoencoder for intrusion detection. In: Proc. ACM SIGKDD, pp. 2841–2851 (2023)
3. Wu, L., Cui, P., Pei, J., Zhao, L.: Graph neural networks: Foundations, frontiers, and applications. Springer (2022)
4. Zhou, J., Cui, G., Hu, S., Zhang, Z., Yang, C., Liu, Z., Wang, L., Li, C., Sun, M.: Graph neural networks: A review of methods and applications. AI Open **1**, 57–81 (2022)
5. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. In: Proc. ICLR (2017)
6. Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., Bengio, Y.: Graph attention networks. In: Proc. ICLR (2018)
7. Hamilton, W.L., Ying, R., Leskovec, J.: Inductive representation learning on large graphs. In: Proc. NeurIPS, pp. 1024–1034 (2017)
8. Zhang, X., He, Y., Brugnone, N., Perlmutter, M., Hirn, M.: MagNet: A neural network for directed graphs. In: Proc. NeurIPS, pp. 27003–27015 (2022)
9. Li, Z., Yoon, S., Kanan, R., Youssef, F., Luo, P.: Graph-based intrusion detection system for IoT networks. Future Generation Computer Systems **139**, 242–254 (2023)
10. Lo, W.W., Layeghy, S., Sarhan, M., Gallagher, M., Portmann, M.: E-GraphSAGE: A graph neural network based intrusion detection system for IoT. In: Proc. IEEE NOMS, pp. 1–9 (2022)
11. Deng, Z., Chen, L., Yang, B., Liu, S., Li, F.: Graph-level network intrusion detection using temporal GCN. Computers & Security **121**, 102845 (2022)
12. Moustafa, N.: A new distributed architecture for evaluating AI-based security systems at the edge: Network TON_IoT datasets. Sustainable Cities and Society **72**, 102994 (2021)
13. Booij, T.M., Chiscop, I., Meeuwissen, E., Moustafa, N., den Hartog, F.T.H.: ToN_IoT: The role of heterogeneity and the need for standardization of features and attack types in IoT network intrusion datasets. IEEE Internet of Things Journal **9**(1), 485–496 (2022)
14. Alsaedi, A., Moustafa, N., Tari, Z., Mahmood, A., Anwar, A.: TON_IoT telemetry dataset: A new generation dataset of IoT and IIoT for data-driven intrusion detection systems. IEEE Access **8**, 165130–165150 (2020)

15. Moustafa, N., Keshk, M., Debie, E., Janicke, H.: Federated TON_IoT Windows datasets for evaluating AI-based security applications. In: Proc. IEEE Int. Conf. Trust, Security and Privacy in Computing and Communications (TrustCom), pp. 848–855 (2020)
16. Moustafa, N., Ahmed, M., Ahmed, S.: Data analytics-enabled intrusion detection: Evaluations of ToN_IoT Linux datasets. In: Proc. IEEE Int. Conf. Trust, Security and Privacy in Computing and Communications (TrustCom), pp. 727–735 (2020)
17. K. Xu, J. Li, M. Zhang, S. S. Du, K. Kawarabayashi, and S. Jegelka, "What can neural networks reason about?," in *Proc. ICLR*, 2020; Extended version: "How powerful are edge-level graph neural networks?," *arXiv preprint arXiv:2301.09505*, 2023.
18. C. Morris, Y. Lipman, H. Maron, B. Rieck, N. M. Kriege, M. Grohe, M. Fey, and K. Borgwardt, "Weisfeiler and Leman go sparse: Towards scalable higher-order graph embeddings," in *Proc. NeurIPS*, 2022; Extended analysis: "Expressiveness of edge-augmented graph neural networks," *arXiv preprint arXiv:2402.11234*, 2024.
19. A. Loukas, G. Puy, and P. Vandergheynst, "PAC-Bayesian generalization bounds for edge prediction in graphs," *Journal of Machine Learning Research*, vol. 25, no. 47, pp. 1-42, 2024.