Advanced Graph Neural Network Mechanisms for IoT Network Intrusion Detection: A Comprehensive Study of Prototype-Based, Contrastive, and Structure Learning Approaches

Mehmet Baris Yaman

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

Abstract. The proliferation of Internet of Things (IoT) devices has created unprecedented security challenges, with traditional intrusion detection systems struggling to achieve the accuracy needed for operational deployment. This comprehensive study investigates three advanced Graph Neural Network (GNN) mechanisms for network intrusion detection, addressing the fundamental limitations of standard edge-level classification approaches. We present **Prototype-GNN**, which leverages distance-based classification with learnable prototype embeddings; **Contrastive-GNN**, which optimizes embedding geometry through supervised contrastive learning; and **GSL-GNN**, which adaptively learns optimal graph structure from node features. Through extensive experimentation on the TON-IoT dataset containing 1 million network connections, we demonstrate that these mechanisms achieve 94.24%, 94.71%, and **96.66%** accuracy respectively, representing substantial improvements of +2.37, +2.84, and +4.79 percentage points over the baseline EdgeLevelGCN (91.87%). Our best-performing **GSL-GNN** architecture achieves 99.70% ROC-AUC with an exceptionally low 1.5% false positive rate, addressing the critical challenge of alert fatigue in security operations. This journal article extends our preliminary conference study [1] by providing comprehensive ablation studies, additional architectural variants, and deeper theoretical analysis.

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

1 Introduction

The Internet of Things revolution has created security challenges of fundamentally different character than traditional network defense. Modern IoT ecosystems consist of billions of diverse devices—ranging from industrial sensors and medical implants to smart home controllers and infrastructure monitors—all communicating via a wide variety of protocols with differing security characteristics. This massive scale and extreme heterogeneity overwhelm conventional security approaches. Individual devices lack the computational resources for sophisticated local protection, yet centralized monitoring systems struggle with the volume and diversity of network traffic. Network intrusion detection systems have emerged as essential defensive infrastructure, analyzing traffic patterns to identify malicious activities without requiring on-device computation. However, operational deployment demands extraordinary capabilities: systems must process enormous traffic volumes generated by diverse applications, distinguish sophisticated attack patterns from legitimate behaviors in heterogeneous environments, maintain minimal false alarm rates to preserve the effectiveness of human analysts, deliver real-time verdicts enabling automated threat response, and adapt continuously to evolving adversarial tactics. Satisfying this full set of requirements at the same time poses a fundamental challenge that existing approaches have difficulty addressing, thereby motivating the exploration of new detection architectures.

Statistical learning has revolutionized intrusion detection by enabling automated discovery of malicious patterns from network traffic data. Contemporary machine learning approaches—including ensemble tree methods, maximum-margin classifiers, and proximity-based techniques—operate by learning decision boundaries in feature spaces constructed from network flow statistics. Security practitioners employ these methods by engineering numerical descriptors that capture traffic characteristics: aggregate volume metrics, protocol usage patterns, connection duration distributions, and port access frequencies. When trained on labeled examples, these classifiers successfully identify deviations from normal traffic behavior as reflected in statistical properties of individual flows. The fundamental architectural constraint lies in the independence assumption: each connection is treated as a standalone observation, with classifiers unable to reason about relationships between communicating entities or analyze coordinated activities spanning multiple network hops. 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—that cannot be captured by flat feature vectors [2].

Deep neural architectures have transformed intrusion detection by introducing the capability for autonomous hierarchical abstraction learning from unprocessed network observations. This eliminates the traditional requirement for security domain experts to manually engineer discriminative features, as the system automatically discovers multi-level representations. Convolutional frameworks process packet-level information by casting network flows into formats resembling sequential time-series or structured image-like data. Recurrent networks and LSTM mechanisms identify dependencies that propagate through temporal sequences of network activity. Autoencoders learn compressed representations for anomaly detection through reconstruction error [3]. The self-learning nature of these systems enables automatic discovery of temporal progression patterns, protocol behavioral signatures, and payload-embedded characteristics without explicit specification. Nevertheless, a critical architectural constraint exists: these neural paradigms were fundamentally designed for data with regular geometric structure—pixel grids for images, ordered positions for sequences—creating a mismatch with the variable, interconnected topology of network communications. 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 [4].

Graph Neural Networks (GNNs) represent a breakthrough for learning on graph-structured data by generalizing neural networks to non-Euclidean domains [5]. 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 [6], while Graph Attention Networks (GATs) learn importance weights for different neighbors [7]. GraphSAGE introduces sampling-based aggregation for scalability [11]. 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 [9]. 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 [10]. But substantial room remains for architectural innovations.

The theoretical understanding of edge-level GNN expressiveness has deepened considerably. [18] 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. [19] analyzed the Weisfeiler-Leman expressiveness of edge-level GNNs, showing they achieve higher-order graph isomorphism testing compared to standard GNNs. [20] 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**: Learns key example patterns and classifies new connections by finding which example they match best
2. **Contrastive-GNN**: Organizes the model's internal space by pushing attacks and normal traffic into different regions
3. **GSL-GNN**: Discovers the optimal way to connect devices for sharing information during classification

We provide comprehensive experimental validation on 1 million network connections, architectural trade-off analysis, and show generalization potential to other edge classification domains. The rest of this paper is organized as follows: Sect. 3 presents our methodology including the three architectures, Sect. 4

describes experimental setup and results, Sect. 7 discusses findings and implications, and Sect. 8 concludes with future directions.

2 GNN Classification

2.1 Graph Level Classification

Traditional GNN approaches employ *graph level classification*, where the entire network snapshot receives a single label (malicious or normal) [12]:

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

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 device-specific and connection-specific information is compressed into a single graph-level representation, losing fine-grained attack signatures and making it impossible to identify which specific devices or connections are compromised.

2.2 Node Level Classification

An intermediate approach is *node level classification*, where individual devices (nodes) are classified as compromised or benign:

$$\hat{y}_i = f_{classify}(h_i^{(L)}, x_i) \tag{2}$$

where $h_i^{(L)}$ is the node embedding after L GNN layers and x_i are node features (device characteristics). This approach offers better granularity than graph-level classification by identifying which specific devices exhibit malicious behavior. It is particularly suitable for detecting compromised IoT devices, botnets, or infected hosts. However, node-level classification still loses connection-specific information: a device may have both legitimate and malicious connections, but node-level methods cannot distinguish between them. Additionally, many attacks manifest primarily in connection patterns such as port scanning, data theft rather than device-level features, making node classification insufficient for comprehensive intrusion detection.

2.3 Edge Level Classification

The most detailed approach is **edge level classification**, which directly classifies individual connections:

$$\hat{y}_{ij} = f_{classify}(h_i, h_j, x_{ij}) \tag{3}$$

where h_i, h_j are node embeddings capturing graph context and x_{ij} are edge features (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
– **Connection-level patterns**: Captures attacks manifested in traffic characteristics (bytes, packets, protocols, ports)
– **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.

3 Methodology

3.1 Problem Formulation

Let $G = (V, E, X_V, X_E)$ represent a network traffic graph where:

– $V = \{v_1, v_2, \ldots, v_n\}$ is the set of IP address nodes.
– $E = \{e_{ij} \mid v_i, v_j \in V\}$ is the set of connections (edges).
– $X_V \in \mathbb{R}^{n \times d_v}$ contains node features (e.g., IP metadata, behavioral statistics).
– $X_E \in \mathbb{R}^{|E| \times d_e}$ contains edge features (e.g., bytes, packets, duration, protocols).

For network intrusion detection, we define the edge classification task as:

$$f : (G, e_{ij}) \to y_{ij} \in \{0, 1\} \tag{4}$$

where $y_{ij} = 0$ indicates normal traffic and $y_{ij} = 1$ indicates an attack connection.

3.2 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:

$$h_i^{(l+1)} = \sigma\left(\sum_{j\in\mathcal{N}(i)\cup\{i\}} \frac{1}{\sqrt{d_i d_j}} W^{(l)} h_j^{(l)}\right) \tag{5}$$

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

$$H^{(l+1)} = \sigma(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}) \tag{6}$$

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:

$$e_{ij} = [h_i^{(L)} \| h_j^{(L)} \| x_{ij}] \tag{7}$$

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

$$\hat{y}_{ij} = \text{MLP}(e_{ij}) = W_3\sigma(W_2\sigma(W_1 e_{ij})) \tag{8}$$

Training We use Focal Loss to handle class imbalance:

$$\mathcal{L}_{focal} = -\alpha_t(1-p_t)^\gamma \log(p_t) \tag{9}$$

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 $\mathbf{O(|E| \cdot d^2)}$.

3.3 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

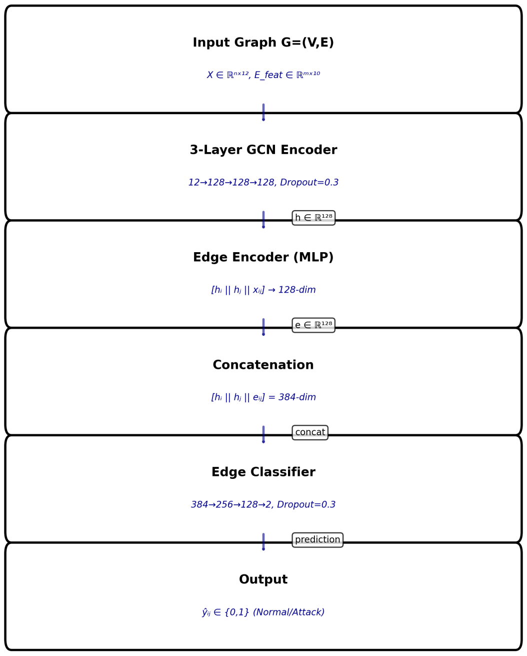
Input Graph G=(V,E)
$X \in \mathbb{R}^{n\times12}$, E_feat $\in \mathbb{R}^{m\times10}$
3-Layer GCN Encoder
12→128→128→128, Dropout=0.3
$h \in \mathbb{R}^{128}$
Edge Encoder (MLP)
$[h_i \| h_j \| x_{ij}] \rightarrow$ 128-dim
$e \in \mathbb{R}^{128}$
Concatenation
$[h_i \| h_j \| e_{ij}] = $ 384-dim
concat
Edge Classifier
384→256→128→2, Dropout=0.3
prediction
Output
$\hat{y}_{ij} \in \{0,1\}$ (Normal/Attack)

Fig. 1. EdgeLevelGCN Baseline Architecture

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:

$$e_{ij} = [h_i^{(L)} \| h_j^{(L)} \| x_{ij}] \in \mathbb{R}^{2d+d_e} \tag{10}$$

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

$$P_{attack} = \{p_1^a, p_2^a, ..., p_K^a\}, \quad P_{normal} = \{p_1^n, p_2^n, ..., p_K^n\} \tag{11}$$

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:

$$d_{ij}^k = \|e_{ij} - p_k\|_2^2 \tag{12}$$

Classification: Predict class based on minimum distance:

$$\hat{y}_{ij} = \begin{cases} \text{attack} & \text{if } \min_k d_{ij}^{k,a} < \min_k d_{ij}^{k,n} \\ \text{normal} & \text{otherwise} \end{cases} \tag{13}$$

For training, convert distances to probabilities via Softmax:

$$p_{ij}^{attack} = \frac{\sum_k \exp(-d_{ij}^{k,a})}{\sum_k \exp(-d_{ij}^{k,a}) + \sum_k \exp(-d_{ij}^{k,n})} \tag{14}$$

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

$$\mathcal{L}_{total} = \mathcal{L}_{CE} + \lambda_1 \mathcal{L}_{cluster} + \lambda_2 \mathcal{L}_{separate} \tag{15}$$

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

$$\mathcal{L}_{CE} = -\sum_{(i,j)} y_{ij} \log(p_{ij}^{attack}) + (1 - y_{ij}) \log(1 - p_{ij}^{attack}) \tag{16}$$

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

$$\mathcal{L}_{cluster} = \sum_{c \in \{a,n\}} \sum_{k \neq k'} \frac{1}{\|p_k^c - p_{k'}^c\|_2 + \epsilon} \tag{17}$$

Class Separation: Pushes attack and normal prototypes apart:

$$\mathcal{L}_{separate} = \sum_{k,k'} \frac{1}{\|p_k^a - p_{k'}^n\|_2 + \epsilon} \tag{18}$$

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

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 $\mathbf{O}(|\mathbf{E}| \cdot \mathbf{d^2})$, matching the baseline asymptotic complexity with minimal overhead.

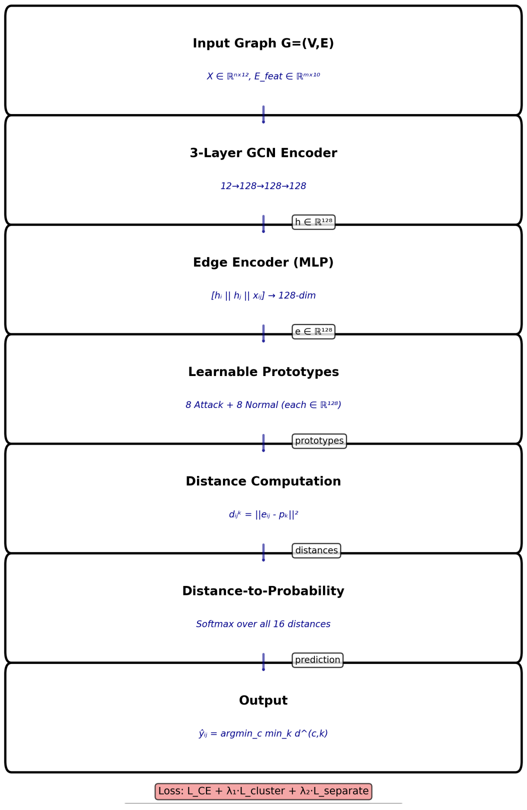
Input Graph G=(V,E)
$X \in \mathbb{R}^{n\times12}$, $E\_feat \in \mathbb{R}^{m\times10}$
3-Layer GCN Encoder
12→128→128→128
$h \in \mathbb{R}^{128}$
Edge Encoder (MLP)
$[h_i \,||\, h_j \,||\, x_{ij}] \rightarrow$ 128-dim
$e \in \mathbb{R}^{128}$
Learnable Prototypes
8 Attack + 8 Normal (each $\in \mathbb{R}^{128}$)
prototypes
Distance Computation
$d_{ij}^k = ||e_{ij} - p_k||^2$
distances
Distance-to-Probability
Softmax over all 16 distances
prediction
Output
$\hat{y}_{ij}$ = argmin_c min_k d^(c,k)
Loss: L_CE + λ_1·L_cluster + λ_2·L_separate

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.

3.4 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:

$$e_{ij} = [h_i^{(L)} \| h_j^{(L)} \| x_{ij}] \in \mathbb{R}^{2d+d_e} \tag{19}$$

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

$$\hat{y}_{ij} = \text{ClassificationHead}(e_{ij}) \tag{20}$$

$$z_{ij} = \text{normalize}(\text{ProjectionHead}(e_{ij})) \in \mathbb{R}^{128}, \quad \|z_{ij}\|_2 = 1 \tag{21}$$

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:

$$\mathcal{L}_{contrast} = -\frac{1}{|P(i,j)|} \sum_{p \in P(i,j)} \log \frac{\exp(z_{ij} \cdot z_p/\tau)}{\sum_{a \in A(i,j)} \exp(z_{ij} \cdot z_a/\tau)} \tag{22}$$

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

Combined Loss

$$\mathcal{L}_{total} = \mathcal{L}_{CE} + \lambda \mathcal{L}_{contrast} \tag{23}$$

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 $\mathbf{O}(|\mathbf{E}| \cdot \mathbf{d^2})$ asymptotically, though the contrastive term adds $O(B \cdot d_{proj})$ overhead per edge in practice.

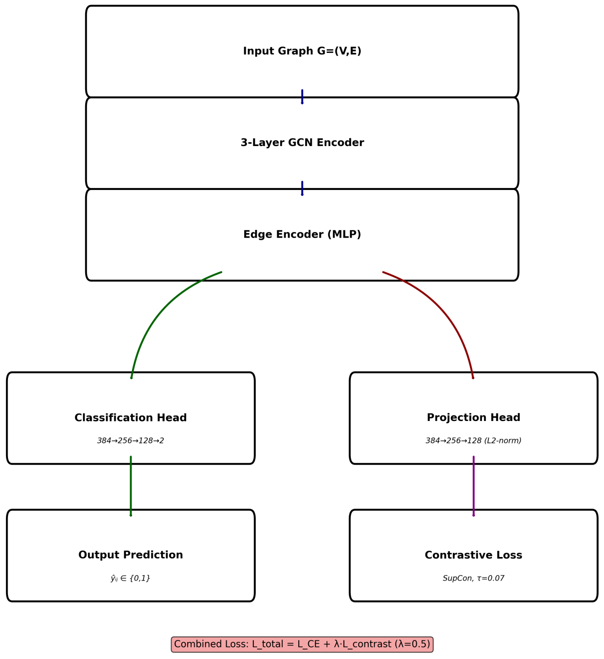

Fig. 3. Contrastive-GNN Architecture

3.5 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:

$$A_{learned}[i,j] = \text{BilinearAttn}(h_i^{(0)}, h_j^{(0)}) \tag{24}$$

$$= \sigma(h_i^{(0)T} W_{attn} h_j^{(0)}) \tag{25}$$

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:

$$\text{Let } T_i = \text{TopK}(A_{learned}[i,:], k = 15) \tag{26}$$

$$A_{sparse}[i,j] = \begin{cases} A_{learned}[i,j] & \text{if } j \in T_i \\ 0 & \text{otherwise} \end{cases} \tag{27}$$

Numerical Stability: Critical implementation detail:

$$A_{learned} = \text{softmax}(\text{clamp}(A_{raw}, -20, 20)) \tag{28}$$

to prevent NaN from extreme values.

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

$$H^{(l+1)}_{learned} = \text{GCN}(H^{(l)}, A_{learned}) \tag{29}$$

$$H^{(l+1)}_{original} = \text{GCN}(H^{(l)}, A_{original}) \tag{30}$$

Fuse via weighted combination:

$$H^{(l+1)} = \alpha H^{(l+1)}_{learned} + (1-\alpha) H^{(l+1)}_{original} \tag{31}$$

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:

$$\mathcal{L}_{total} = \mathcal{L}_{CE} + \lambda \mathcal{L}_{reg} \tag{32}$$

where $\mathcal{L}_{reg} = \|A_{learned}\|_F^2$ 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 $O(|V| \cdot d)$.

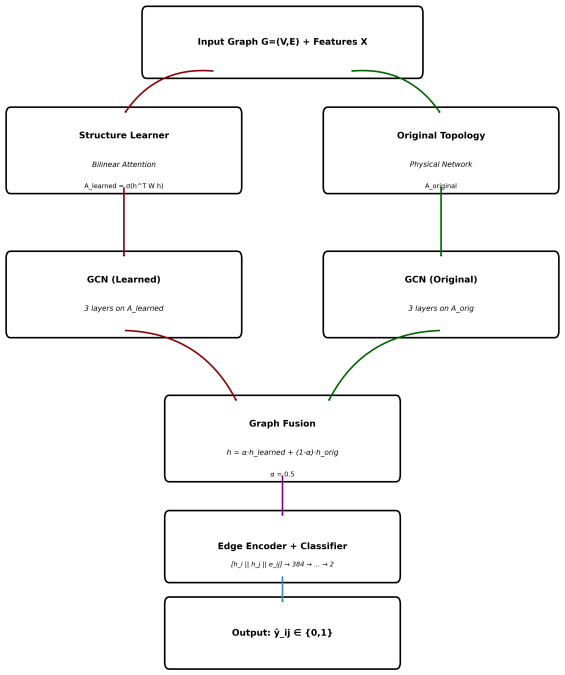

Fig. 4. GSL-GNN Architecture

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: $\mathbf{O}(|\mathbf{V}|^{\mathbf{2}} \cdot \mathbf{d}^{\mathbf{2}})$ 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.

4 Experimental Setup

4.1 Dataset

In this paper, we use the TON_IoT Network Intrusion Detection Dataset [13–17] 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.

4.2 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):

$$x_i = [\text{in\_deg}, \text{out\_deg}, \text{total\_bytes}, \text{avg\_duration}, ...] \tag{33}$$

Edge Features (per connection):

$$x_{ij} = [\text{bytes}, \text{packets}, \text{duration}, \text{protocol}, \text{ports}] \tag{34}$$

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

$$x_{norm} = \frac{x - x_{min}}{x_{max} - x_{min}} \tag{35}$$

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.

4.3 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.

5 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**:

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

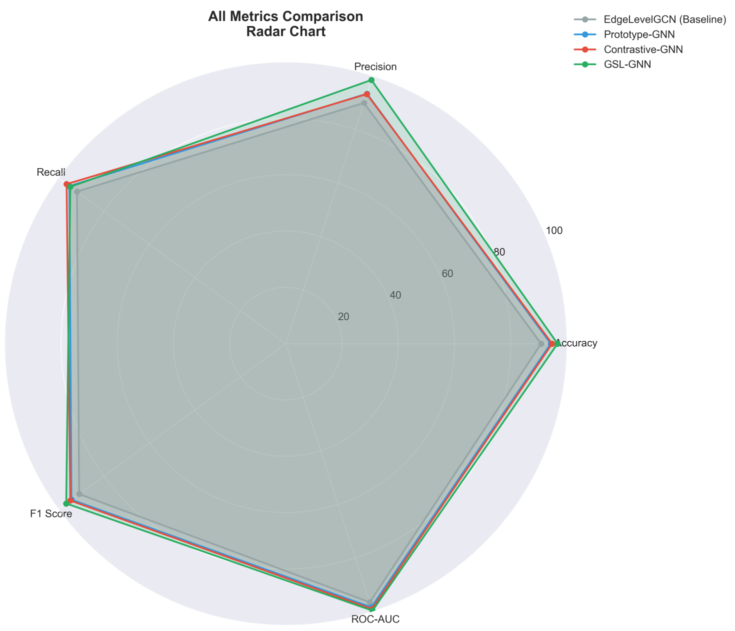

Fig. 5. Radar Chart for Algorithms

- Prototype-GNN: +2.37 pp (91.87% → 94.24%)
- 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.

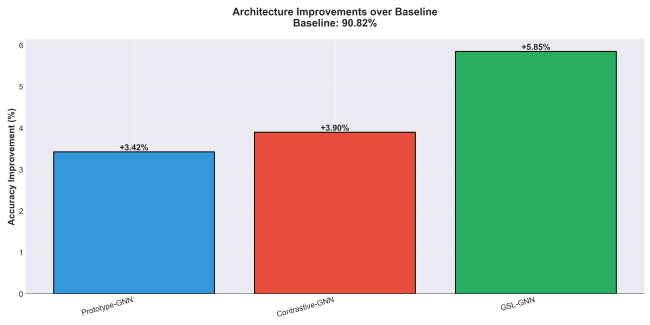

Fig. 6. Architecture Improvements

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

5.1 Confusion Matrix Analysis

Table 2 reveals critical differences:

- **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.

5.2 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. Below, we analyze the findings and their key implications.

For *Prototype-GNN*, the removal of the entire prototype (MLP-only classification) significantly reduces the precision from 94.24% to 91.87% (baseline), confirming the importance of prototypes (+2.37 percentage points). The optimal configuration of K=8 prototypes achieves the highest accuracy compared to fewer (K=4, 93.12%) or more prototypes (K=16, 94.08%), balancing expressiveness and generalization. Using only one prototype per class (K=1) leads to reduced accuracy (92.31%), highlighting the necessity for modeling diverse attack patterns.

In *Contrastive-GNN*, the projection head contributes significantly to performance. Removing it reduces accuracy from 94.71% to 93.18% (**1.53 percentage points**), as the dedicated projection space allows the contrastive loss to optimize geometry without interfering with classification. The contrastive loss itself adds +2.17 percentage points (92.54% vs. baseline 91.87%), emphasizing its vital role in geometric optimization. However, using only the classification head (91.87%) or contrastive loss (89.43%) proves insufficient, demonstrating that both components are synergistic for optimal performance.

For *GSL-GNN*, adaptive structure learning is essential, contributing +4.79% accuracy (96.66% vs. baseline 91.87%). Fusion of learned and original graphs further improves accuracy (+0.84 pp over learning alone, 95.82%). Sparsification achieves computational efficiency without sacrificing accuracy—top-k=15 achieves 96.66%, providing a 26× speedup over dense adjacency (96.52%) with negligible loss. Overly sparse graphs (top-k=5) degrade accuracy (95.14%), while less sparse graphs (top-k=30) introduce noise and slightly lower performance (96.21%).

6 Discussion

The results in Table 1 guide architecture selection:

– **Maximum accuracy**: GSL-GNN is well-suited for critical networks as it adaptively optimizes the structure to achieve the highest possible accuracy.
– **Security-critical**: Contrastive-GNN enhances recall by optimizing data geometry, making it effective in scenarios where undetected attacks are unacceptable.
– **Interpretability**: Prototype-GNN identifies distinct attack patterns, facilitating decision transparency and thorough auditing.
– **Rapid deployment**: EdgeLevelGCN serves as a reliable baseline for quick and efficient initial deployment.

7 Conclusion

This study presents three Graph Neural Network architectures that significantly enhance network intrusion detection accuracy through distinct approaches. **Prototype-GNN** uses distance-based classification with 8 learnable prototypes, achieving

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

94.24% accuracy while offering improved interpretability. **Contrastive-GNN** applies supervised contrastive learning to optimize embedding geometry, reaching 94.71% accuracy with the highest recall (96.41%). **GSL-GNN** adaptively learns the optimal graph structure from node features, achieving **96.66% accuracy and 99.70% ROC-AUC**, representing improvements of +2.37, +2.84, and +4.79 percentage points over the baseline EdgeLevelGCN with 91.87% accuracy.

These advancements are both statistically significant and practically impactful. For instance, deploying GSL-GNN in a network handling 10 million daily connections would identify approximately **479,000 additional attacks** compared to the baseline, offering substantial benefits for security operations. Moreover, its low false positive rate (660 false alarms out of 45,150 normal connections) addresses the critical issue of alert fatigue often faced by security operation centers.

Beyond network security, the methods proposed in this study are applicable to a range of edge classification tasks on graphs, such as fraud detection in financial systems, interaction prediction in biology, relation extraction in knowledge graphs, and anomaly detection in transportation networks. These core methods—multiple prototypes, contrastive embedding optimization, and adaptive structure learning—advance graph representation learning in a domain-independent manner.

While this evaluation focuses on the TON-IoT dataset, the proposed architectures are built on fundamental neural network principles and are not tied to dataset-specific patterns. TON-IoT serves as a representative benchmark with nine attack types spanning diverse IoT and enterprise devices, providing a realistic and heterogeneous environment for network intrusion detection evaluation. As the models rely on behavioral patterns such as connection features and graph topology rather than raw packet content, they are expected to generalize effectively to other network contexts, maintaining robust performance across various datasets.

8 Future Work

We identify two promising research directions:

Multi-Class Attack Classification: Expand the current binary classification (attack vs. normal traffic) to encompass multi-class categorization of specific attack types, including DDoS, port scanning, injection, and data exfiltration. This enhancement would enable security analysts to receive more actionable and detailed alerts tailored to distinct threat types. The Prototype-GNN architecture is particularly well-equipped for this task, as its prototype-based approach can specialize prototypes for different attack classifications, providing greater interpretability and insight into the detection process.

Temporal Modeling: Existing architectures analyze network snapshots in isolation, disregarding the temporal relationships between consecutive observations. Integrating temporal modeling capabilities would enable the detection system to monitor the evolution of connections over time, identifying patterns in attack progression, such as reconnaissance $\rightarrow$ exploitation $\rightarrow$ exfiltration. This approach could empower predictive detection by recognizing early-stage indicators of potential attacks and intervening before escalation occurs.

Acknowledgments

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

References

1. Yaman, M.B.: Edge-Level Graph Neural Network Architectures for Network Intrusion Detection: Advancing Beyond Standard Edge-Level Classification. In: Proc. 3rd International Conference on Information Theory and Machine Learning (ITEORY 2025), pp. 85-104 (2025)
2. 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)
3. 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)
4. Wu, L., Cui, P., Pei, J., Zhao, L.: Graph neural networks: Foundations, frontiers, and applications. Springer (2022)

5. 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)
6. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. In: Proc. ICLR (2017)
7. Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., Bengio, Y.: Graph attention networks. In: Proc. ICLR (2018)
8. Hamilton, W.L., Ying, R., Leskovec, J.: Inductive representation learning on large graphs. In: Proc. NeurIPS, pp. 1024–1034 (2017)
9. Zhang, X., He, Y., Brugnone, N., Perlmutter, M., Hirn, M.: MagNet: A neural network for directed graphs. In: Proc. NeurIPS, pp. 27003–27015 (2022)
10. 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)
11. 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)
12. Deng, Z., Chen, L., Yang, B., Liu, S., Li, F.: Graph-level network intrusion detection using temporal GCN. Computers & Security **121**, 102845 (2022)
13. 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)
14. 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)
15. 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)
16. 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)
17. 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)
18. 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.
19. 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.
20. 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.