Computers, Materials & Continua

Tech Science Press

https://doi.org/10.32604/cmc.2026.086743

ARTICLE

The Method of Malicious Traffic Detection for Internet of Things Based on Lightweight Graph Neural Networks

Baofeng Duan[1], **Xinghai Yu**[1], **Peng Wang**[2], **Tao Feng**[1] **and Yongbo Jiang**[1,*]

[1]School of Computer Science and Artificial Intelligence (College of Software), Lanzhou University of Technology, Lanzhou, China
[2]School of Petrochemical Engineering, Lanzhou University of Technology, Lanzhou, China

*Corresponding Author: Yongbo Jiang. Email: jiangyb@lut.edu.cn

Received: 05 June 2026; Accepted: 11 August 2026; Published: 15 September 2026

ABSTRACT: With the sustained expansion of complex Internet of Things (IoT) ecosystems, malicious traffic detection has become critical for maintaining both cyber security and operational continuity. Modern IoT deployments contain heterogeneous devices, ubiquitous sensing layers, edge services, and autonomous assets, so abnormal communication may affect not only data confidentiality but also physical operations. To address the limitations of independent flow-level detection and heavy graph propagation, this paper proposes a Lightweight Graph-Attentive Network for Traffic Detection (LGNT). LGNT constructs a directed traffic-interaction graph from NetFlow records, where communication entities are represented as nodes and traffic sessions are represented as edges. Communication-strength-based auxiliary node supervision provides an activity-aware structural signal, while a compact backbone combining topology adaptive graph convolution (TAGConv) and graph attention v2 convolution (GATv2Conv) captures local topological dependencies and key communication relations. A structure-significance pruning strategy is further introduced to reduce the message-passing edge set and graph computation overhead. Experiments on NetFlow BoT-IoT (NF-BoT-IoT) and NetFlow ToN-IoT (NF-ToN-IoT) show that LGNT obtains effective results in both binary and multi-class detection tasks. Specifically, it achieves 94.28% accuracy, 97.36% area under the curve (AUC), and 86.88% F1 on NF-BoT-IoT, and 99.93% accuracy, 99.95% AUC, and 69.05% weighted F1 on NF-ToN-IoT. The per-class analysis further shows that long-tailed minority categories remain challenging in fine-grained NF-ToN-IoT recognition. Overall, LGNT improves the balance between traffic-interaction modeling, detection performance, and deployment efficiency while keeping the parameter scale at 0.236 million.

KEYWORDS: Graph neural networks; Internet of Things; intrusion detection; lightweight models; malicious traffic detection; traffic interaction graphs

1 Introduction

The Internet of Things (IoT) has become a common infrastructure for smart homes, industrial control systems, intelligent transportation, smart-city services, and ubiquitous sensing environments [1]. Modern IoT ecosystems are increasingly deployed as heterogeneous cyber-physical infrastructures in which distributed sensors, edge devices, logistics assets, industrial controllers, and autonomous vehicles continuously exchange operational states and control-related messages. In such environments, malicious traffic is not only a source of data leakage or network congestion; abnormal communication may interrupt sensing feedback, disrupt supply-chain coordination, mislead autonomous assets, and even halt physical operations. The security challenges of IoT are amplified by resource-constrained devices, heterogeneous protocols, decentralized connectivity, and large attack surfaces, which have been identified as persistent obstacles for

secure IoT deployment [2]. Recent ubiquitous supply-chain systems with autonomous vehicles further show that emerging IoT applications rely on trusted and timely communication among sensing layers, edge nodes, blockchain-based coordination mechanisms, and mobile assets [3]. Recent LLM-enabled zero-day IoT threat detection also indicates that heterogeneous IoT traces require context-aware reasoning beyond simple rule matching [4]. Therefore, IoT malicious traffic detection should be considered a critical component for maintaining both cyber security and operational continuity in modern connected environments. NetFlow-based datasets provide a useful basis for this line of research because they organize traffic behavior into compact flow records that are easier to use in machine-learning-based detection pipelines [5]. Standard feature-set studies have further shown that flow-level representations are helpful for improving generalizability and interpretability across different intrusion detection datasets [6].

Compared with conventional enterprise networks, IoT traffic is more heterogeneous and behaviorally uneven because devices differ in protocol, sampling frequency, computational capability, and communication role. Many malicious behaviors are not obvious from a single flow record; instead, they appear through cross-host or cross-session interaction patterns, such as repeated scanning, lateral movement, abnormal device-to-device communication, cross-domain propagation, and coordinated high-frequency connections. Graph neural networks (GNNs) provide a natural modeling tool for this setting because network flows can be transformed into a graph in which hosts are represented as nodes, communication sessions are represented as edges, and edge attributes describe flow-level behavior. E-GraphSAGE demonstrates that flow-based NIDS data can be represented in graph form and that GNNs can jointly capture edge features and topological information for IoT intrusion detection [7]. Anomal-E further shows that edge features and graph topology are useful for learning network behavior in self-supervised intrusion and anomaly detection [8]. Bilot et al. reviewed GNN-based intrusion detection and emphasized graph construction and deployment as two major research issues [9]. Yagmur et al. systematically reviewed graph neural networks and graph autoencoders for intrusion detection systems and summarized taxonomy, comparative evaluation, and practical research gaps [10]. Edge-featured multi-hop attention further indicates that edge attributes are important for flow-level intrusion detection [11]. BS-GAT extends graph attention to edge-computing scenarios and highlights the need for deployable graph-based detectors [12].

Nevertheless, existing graph-based intrusion detection methods still face a trade-off between representation capacity and deployment cost. Models with stronger structural modeling ability often require more parameters, more message-passing operations, and longer inference time. This issue is particularly important for resource-constrained IoT scenarios. For flow-centric detection, edge direction, communication intensity, and protocol variation often provide useful evidence, but an overly dense graph may also introduce redundant relations and propagate noise.

The main challenge addressed in this study is not only to improve classification accuracy, but also to jointly satisfy three practical requirements in IoT malicious traffic detection: capturing interaction-level traffic dependencies across hosts and sessions, maintaining effective detection under imbalanced attack distributions, and reducing graph computation cost for resource-constrained deployment. Existing graph-based detectors show the value of structural traffic modeling, but their graph construction, auxiliary supervision, and efficiency-control mechanisms are often considered separately. Therefore, a compact framework that integrates traffic graph modeling, edge-level classification, auxiliary activity supervision, and structural compression is still needed. To address this challenge, we propose LGNT, a lightweight graph-attentive network for edge-level IoT malicious traffic detection. The main contributions are summarized as follows:

(1) We construct a directed traffic interaction graph from NetFlow records and formulate IoT malicious traffic detection as an edge-level classification task, which preserves cross-device and cross-session communication patterns.
(2) We design a compact graph encoding backbone by combining shallow multi-hop TAGConv aggregation and local GATv2-based attention, so that topology-aware representation learning can be retained under a restrained parameter budget.
(3) We introduce communication-strength-based auxiliary node supervision to provide activity-aware structural guidance for edge-level traffic classification, and we further analyze its effect through ablation and sensitivity evaluation.
(4) We apply structure-significance pruning to compress the message-passing edge set and evaluate the proposed method on NF-BoT-IoT and NF-ToN-IoT under binary and multi-class settings.

The remainder of this paper is organized as follows. Section 2 reviews related work. Section 3 presents the proposed method. Section 4 reports the experiments and discusses the results. Section 5 concludes the paper and outlines future work.

2 Related Work

2.1 IoT Malicious Traffic Detection

Early IoT malicious traffic detection mainly relied on handcrafted features and conventional classifiers. As datasets became larger and attack behaviors became more complex, deep learning models were increasingly adopted to learn traffic representations automatically. Yang et al. combined data purification with separable convolution to reduce redundant computation in an IoT intrusion detection model [13]. Li and Yao proposed a two-stage lightweight framework that integrates self-supervised contrastive learning and self-distillation [14]. Wang et al. compressed an IoT intrusion detector through an improved BERT-of-Theseus design [15]. Azimjonov and Kim pursued lightweight detection through fine-tuned linear support vector machine (SVM) and feature selection [16]. Misrak and Melaku combined improved feature engineering with dynamic quantization to reduce inference cost in IoT intrusion detection [17]. Together, these studies make clear that lightweight design is no longer a secondary consideration in IoT intrusion detection. Their common weakness is that many still learn from isolated flow samples. This simplification is easy to implement, but it leaves part of the attack context outside the model, especially for multi-stage behavior, lateral movement, and coordinated bursts of abnormal sessions.

2.2 GNN-Based Intrusion Detection

Graph-based intrusion detection treats network events as relations among hosts, services, flows, or system entities, rather than as independent tabular samples. Bilot et al. reviewed GNN-based intrusion detection and emphasized graph construction and deployment as two major research issues [9]. A concatenated multigraph model has been used to represent repeated interactions between IoT communication entities [18]. Centrality-enhanced graph learning has shown that topological indicators can improve IoT intrusion detection [19]. Edge-featured multi-hop attention further indicates that edge attributes are important for flow-level intrusion detection [11]. BS-GAT extends graph attention to edge-computing scenarios and highlights the need for deployable graph-based detectors [12]. Related work published in Computers, Materials & Continua also indicates that current intrusion detection research is moving toward hybrid representation learning for imbalanced attack traffic [20]. In particular, provenance-graph analysis for advanced persistent threat detection shows that graph-structured security data can expose causal dependencies that are difficult to observe in linear event sequences [21]. These studies support the use of traffic graphs, but they also reveal the

need for a compact architecture that can retain relational information without introducing excessive graph propagation cost.

Compared with representative GNN-based intrusion detection frameworks, the proposed LGNT emphasizes a different design objective. E-GraphSAGE demonstrates that flow-based IoT intrusion detection data can be represented as graphs and that both edge features and graph topology are useful for classification, but it does not focus on auxiliary node-level activity supervision or structure-aware pruning [7]. Anomal-E further exploits edge features and graph topology in a self-supervised process, whereas its main objective is anomaly representation learning rather than lightweight supervised edge classification [8]. Recent multigraph and provenance-graph methods also show that repeated interactions and causal dependencies can expose attack behaviors that are difficult to identify from isolated records; however, these methods often pay less attention to controlling message-passing cost for edge deployment. In contrast, LGNT integrates edge-level traffic graph construction, communication-strength auxiliary supervision, a compact TAGConv-GATv2 encoder, and structural pruning into a unified lightweight detection framework.

2.3 Self-Supervised Graph Learning and Lightweight Optimization

Self-supervised learning is increasingly used to improve graph representation learning when labels are scarce or unevenly distributed. HeGCL shows that self-supervised signals can strengthen heterogeneous graph-level representations [22]. Automated self-supervised learning for graph anomaly detection further shows that task construction and hyperparameter selection may strongly affect unsupervised detection performance [23]. In intrusion detection, graph features have also been combined with transformer-based autoencoders and contrastive learning [24]. ResACAG improves graph-based intrusion detection through residual and adaptive context-aware modeling [25]. RHNN-IoT introduces reinforced hypergraph representation learning to capture high-order relations among IoT entities [26]. Most lightweight intrusion detection methods reduce feature dimensions, compress parameters, or quantize neural layers. Less attention has been given to redundancy in the graph structure itself. This observation motivates the proposed design, in which traffic interaction modeling is combined with a compact graph encoder and structure-aware pruning.

3 Materials and Methods

The overall pipeline of LGNT is shown in Fig. 1. Starting from NetFlow records, the method builds a directed traffic interaction graph, generates auxiliary node-attribute labels, applies lightweight graph encoding and structural pruning, and finally outputs edge-level malicious traffic predictions.

3.1 Traffic Interaction Graph Construction

To model interactions among communicating entities more faithfully, each NetFlow record is not treated as an isolated sample. Instead, all records are organized into a directed attributed graph. This representation preserves both statistical traffic features and communication topology, thereby providing structured input for graph neural modeling.

Let the original NetFlow record set be

$$\mathcal{F} = \{f_i\}_{i=1}^{N_e}, \tag{1}$$

where f_i denotes the i-th flow record and N_e is the number of flow records. Each record contains the source address, destination address, flow duration, byte counts, packet counts, protocol information, and attack label. The full record set is converted into a directed attributed graph

$$G = (V, E, X_E, Y_E), \tag{2}$$

where V is the node set, E is the edge set, X_E denotes edge attributes, and Y_E denotes edge labels. The node set is formed from the union of source and destination address fields. A node mapping table converts the original address space into consecutive integer indices. For a flow record f_i, if its source node index is u_i and its destination node index is v_i, the corresponding directed edge is

$$e_i = (u_i, v_i), \quad e_i \in E. \tag{3}$$

Collecting all directed edges yields the edge-index matrix

$$\mathbf{I}_E = \begin{bmatrix} u_1 & u_2 & \cdots & u_{N_e} \\ v_1 & v_2 & \cdots & v_{N_e} \end{bmatrix}. \tag{4}$$

Under this formulation, IoT malicious traffic detection is modeled as an edge-classification problem.

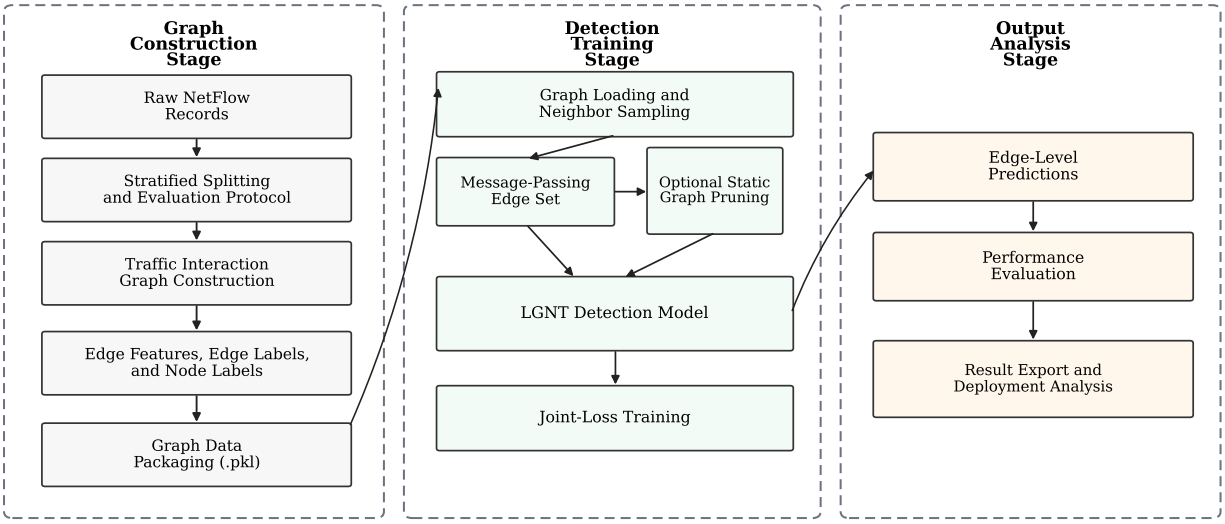

Figure 1: Overall framework of the proposed LGNT pipeline. NetFlow records are first converted into a traffic interaction graph, and edge-level malicious traffic detection is then performed through graph encoding, pruning, and classification. Note: LGNT, Lightweight Graph-Attentive Network for Traffic Detection.

3.2 Edge Attributes, Labels, and Auxiliary Node Labels

Each edge uses a fixed eight-dimensional NetFlow feature vector $\mathbf{x}_e \in \mathbb{R}^8$. The selected features include flow duration, inbound bytes, inbound packets, outbound bytes, outbound packets, transport-layer protocol, application-layer protocol, and TCP flags. For binary classification, the Attack field distinguishes benign traffic from malicious traffic. For multi-class classification, the Label field identifies the attack category. In the implementation, the edge feature and label are stored as $\tilde{\mathbf{x}}_e = [\mathbf{x}_e \| y_e]$, where $\|$ denotes concatenation and y_e is the edge supervision label. During training, the feature part and label part are separated before model input and loss calculation.

To characterize communication activity, an auxiliary node label is introduced as a binary activity-level label rather than an attack-category label. Let E_t be the edge set in the training partition. For each communication entity v, its communication strength is calculated from the incoming and outgoing flow interactions connected to the node:

$$s_v = \sum_{e \in E_t(v)} (p_e^{in} + p_e^{out}), \quad \theta = \frac{1}{|V_t|} \sum_{v \in V_t} s_v, \quad z_v = \mathbb{I}(s_v \geq \theta), \tag{5}$$

where $E_t(v)$ denotes training edges incident to v, p_e^{in} and p_e^{out} denote inbound and outbound packet counts, V_t is the set of nodes observed in the training graph, and $\mathbb{I}(\cdot)$ is the indicator function. The threshold is estimated only from the training graph and then applied to validation and test graphs. Therefore, the auxiliary labels are generated from traffic activity statistics rather than ground-truth attack categories, which reduces the risk of label leakage from the edge-level classification target.

Algorithm 1 gives the complete construction process. The algorithm also clarifies that node auxiliary labels are calculated only from the training partition, which avoids information leakage from validation and test edges.

Algorithm 1: Traffic interaction graph construction and auxiliary node label generation

1: **Input:** NetFlow record set $\mathcal{F}$; edge-label field; split field.
2: **Output:** G, $\mathbf{I}_E$, X_E, Y_E, and Z_V.
3: Initialize the node mapping table, node set V, and edge set E.
4: Map each record f_i to source and destination node indices u_i and v_i.
5: Construct $e_i = (u_i, v_i)$, extract $\mathbf{x}_{e_i}$, and read the label y_{e_i}.
6: Write e_i, $\mathbf{x}_{e_i}$, and y_{e_i} to $\mathbf{I}_E$, X_E, and Y_E.
7: Use only training edges to compute s_v, estimate θ, and generate z_v.
8: Return G, $\mathbf{I}_E$, X_E, Y_E, and Z_V.

3.3 Data Generation and Experimental Splits

After graph structure and attributes are defined, the original CSV records are converted into graph files. Stratified splitting preserves class ratios across training, validation, and test subsets. The main evaluation protocol is five-fold stratified cross-validation with a 60%/20%/20% split in each fold. A fixed split is also retained for debugging, pruning analysis, and reproducibility comparison.

3.4 Graph Pruning Strategy

Large traffic interaction graphs often contain far more edges than nodes. Running message passing and attention directly on the complete graph increases computational cost and may diffuse noise. Therefore, a static graph-pruning step is applied before forward propagation.

Two pruning modes are considered. The first is random pruning, which uniformly samples a subset of edges from the complete edge set and is used as a non-structure-aware baseline. The second is structure-significance pruning based on node degree. For an edge $e = (u, v)$, its importance score is calculated as

$$s(e) = \deg(u) + \deg(v), \tag{6}$$

where $\deg(u)$ and $\deg(v)$ denote the degrees of the two incident nodes. Edges are ranked by $s(e)$, and the top k edges are retained to form the pruned edge set E_p. If ρ is the retained-edge ratio, then $k = \lfloor \rho |E| \rfloor$. Subsequent graph convolution and attention aggregation are performed only on E_p.

This pruning scheme is static rather than learnable. Although it is less flexible than adaptive pruning, it is simple, inexpensive, and consistent with the lightweight deployment goal of this study. Fig. 2 presents the internal architecture of LGNT. The model uses a graph encoder to learn node representations, an edge readout module to recover edge-level representations, and a classifier to predict traffic labels.

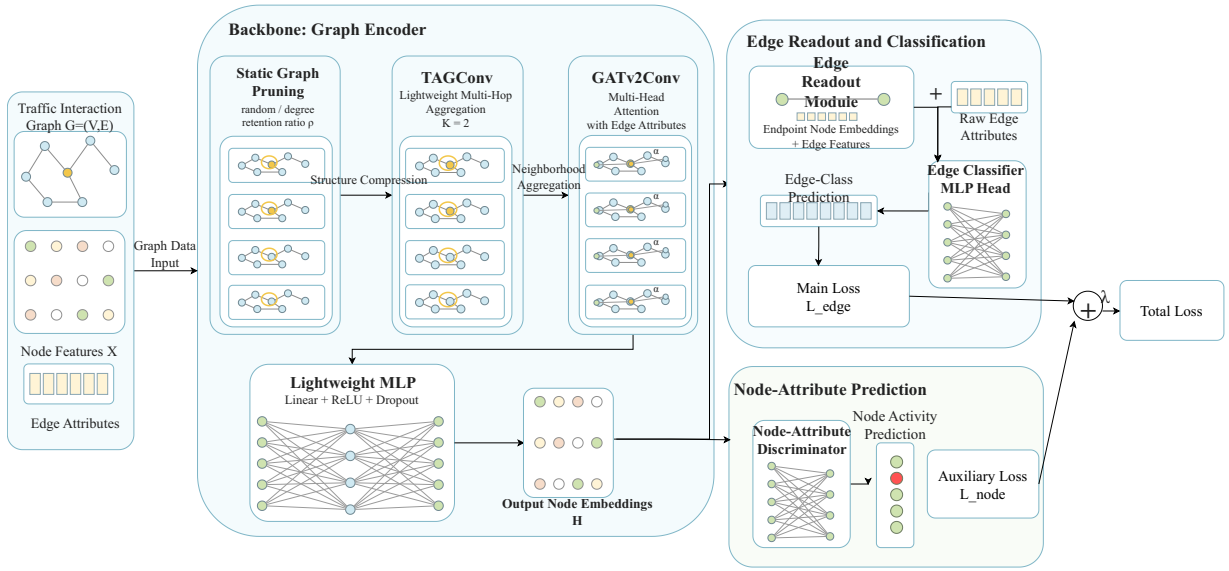

Figure 2: Architecture of the LGNT model. The model learns node embeddings from the pruned traffic interaction graph, constructs edge representations from endpoint embeddings and edge attributes, and predicts malicious traffic labels through an edge classifier. Note: LGNT, Lightweight Graph-Attentive Network for Traffic Detection; TAGConv, topology adaptive graph convolution; GATv2, graph attention v2 convolution.

3.5 LGNT Model

Let the input node-feature matrix be $\mathbf{X} \in \mathbb{R}^{N_v \times d_{in}}$, where N_v is the number of nodes and d_{in} is the input dimension. The first encoder stage uses topology adaptive graph convolution (TAGConv), which follows the topology adaptive graph convolution idea of designing fixed-size learnable filters over graph neighborhoods [27]. In LGNT, this stage performs lightweight topology-adaptive aggregation:

$$\mathbf{H}^{(1)} = \sum_{k=0}^{K} \hat{\mathbf{A}}^{k} \mathbf{X} \mathbf{\Theta}_k, \tag{7}$$

where $\hat{\mathbf{A}}$ is the normalized adjacency matrix and $\mathbf{\Theta}_k$ is the learnable weight matrix for the k-hop propagation term. A small K is used to capture short-range structural dependencies while avoiding excessive propagation depth.

The TAGConv output is then passed to graph attention v2 convolution (GATv2Conv) for local attention enhancement. GATv2 was introduced to address the static attention limitation of the original graph attention mechanism [28]. For edge (u, v), the attention coefficient is computed as

$$\alpha_{uv} = \frac{\exp\left(\mathbf{a}^{\top} \sigma\left(\mathbf{W}[\mathbf{h}_u \| \mathbf{h}_v \| \mathbf{x}_{uv}]\right)\right)}{\sum_{r \in \mathcal{N}(u)} \exp\left(\mathbf{a}^{\top} \sigma\left(\mathbf{W}[\mathbf{h}_u \| \mathbf{h}_r \| \mathbf{x}_{ur}]\right)\right)}, \tag{8}$$

where $\mathbf{h}_u$ and $\mathbf{h}_v$ are node embeddings, $\mathbf{x}_{uv}$ is the edge feature vector, $\mathbf{W}$ and $\mathbf{a}$ are learnable parameters, $\sigma(\cdot)$ is a nonlinear activation function, and $\mathcal{N}(u)$ is the neighbor set of node u. Node u is updated by

$$\mathbf{h}'_u = \sigma\left(\sum_{v \in \mathcal{N}(u)} \alpha_{uv} \mathbf{W}_m \mathbf{h}_v\right), \tag{9}$$

where $\mathbf{W}_m$ is the message transformation matrix. Four attention heads are used in the main configuration, and their outputs are concatenated. To enrich channel interactions without heavy overhead, the encoder ends with a lightweight multilayer perceptron, written as $\mathbf{H} = \mathrm{MLP}(\mathbf{H}^{(2)})$, where $\mathbf{H}^{(2)}$ denotes the GATv2-enhanced node representation.

3.5.1 Edge Readout and Classification

Because the detection target is an edge rather than a node, LGNT constructs an edge representation after node embeddings are obtained. For edge $e = (u, v)$, endpoint embeddings $\mathbf{h}_u$ and $\mathbf{h}_v$ are concatenated with the original edge feature vector as $\mathbf{g}_{uv} = [\mathbf{h}_u \| \mathbf{h}_v \| \mathbf{x}_{uv}]$. The final classifier is a two-layer perceptron:

$$\mathbf{q}_{uv} = \sigma(\mathbf{W}_1 \mathbf{g}_{uv} + \mathbf{b}_1), \tag{10}$$

$$\hat{\mathbf{y}}_{uv} = \mathbf{W}_2 \mathbf{q}_{uv} + \mathbf{b}_2, \tag{11}$$

where $\mathbf{W}_1$, $\mathbf{W}_2$, $\mathbf{b}_1$, and $\mathbf{b}_2$ are classifier parameters.

3.5.2 Auxiliary Node-Attribute Branch and Joint Loss

To stabilize node representations under class imbalance and noisy supervision, an auxiliary branch predicts whether node v belongs to the high-traffic group:

$$\hat{z}_v = \mathrm{Sigmoid}(\mathbf{w}_z^{\top} \mathbf{h}_v + b_z), \tag{12}$$

where $\mathbf{w}_z$ and b_z are learnable parameters. For the main edge-classification task, weighted cross-entropy is used:

$$\mathcal{L}_{edge} = -\sum_{e \in E_t} \sum_{c=1}^{C} w_c \, \mathbb{I}(y_e = c) \log p_{e,c}, \tag{13}$$

where C is the number of edge classes, w_c is the class weight, and $p_{e,c}$ is the predicted probability of edge e belonging to class c. For the auxiliary branch, binary cross-entropy is defined as

$$\mathcal{L}_{node} = -\sum_{v \in V} \left[z_v \log \hat{z}_v + (1 - z_v) \log(1 - \hat{z}_v) \right]. \tag{14}$$

The total loss is

$$\mathcal{L} = \mathcal{L}_{edge} + \lambda \mathcal{L}_{node}, \tag{15}$$

where λ controls the contribution of the auxiliary task. The auxiliary branch is used only to regularize node representations during training. It does not replace the edge-level malicious traffic classification objective, and the final prediction during inference is still performed by the edge classifier.

4 Experiments and Discussion

4.1 Experimental Setup

The experiments use two public NetFlow datasets, NF-BoT-IoT and NF-ToN-IoT. Both datasets provide binary labels and multi-class labels, which makes it possible to evaluate both coarse-grained attack detection and fine-grained attack-type recognition. Table 1 reports the class statistics and the fixed split sizes used in the single-split analysis. The main evaluation protocol is five-fold stratified cross-validation, and each fold uses a 60%/20%/20% split for training, validation, and testing.

Table 1: Statistical information of the datasets.

Dataset	Class	Samples	Ratio
NF-BoT-IoT	Benign	13,859	2.31%
NF-BoT-IoT	Attack	586,241	97.69%
NF-ToN-IoT	Benign	270,279	19.60%
NF-ToN-IoT	Attack	1,108,995	80.40%

Note: NF-BoT-IoT, NetFlow BoT-IoT; NF-ToN-IoT, NetFlow ToN-IoT; IoT, Internet of Things.

Table 2 lists the main training configuration. All experiments are conducted on Ubuntu 20.04 using an Intel Xeon Gold 6330 CPU at 2.00 GHz, 755 GiB of memory, and an NVIDIA GeForce RTX 3090 GPU with 24 GB memory under CUDA 11.3. The software stack includes Python 3.9.15, PyTorch 1.13.0, PyTorch Geometric 2.1.0, PyTorch Lightning 1.7.7, and Scikit-learn 1.0.2.

Table 2: Main training hyperparameters.

Item	Value
GNN layers	2
Attention heads	4
Hidden dimension	128
Maximum epochs	100
Batch size	32,768
Dropout	0.2
Optimizer	Adam
Loss function	Binary cross-entropy/cross-entropy
Parameters	0.236M

Note: GNN, graph neural network; Adam, adaptive moment estimation.

The main evaluation metrics are accuracy, AUC, precision, recall, and F1. Macro-F1 is additionally used in the pruning and ablation analysis because it better reflects minority-class behavior under imbalanced class distributions. XGBoost is included as a strong tree-based baseline for flow-level classification [29]. E-GraphSAGE is included as a representative graph neural baseline for IoT intrusion detection [7]. Anomal-E is included as a self-supervised graph-based intrusion detection baseline [8].

4.2 Binary Classification Results

Fig. 3 shows the binary classification results on NF-BoT-IoT. LGNT obtains the best overall binary-classification performance on this dataset. Compared with LGNT-noAux, the full model improves AUC by 0.45 percentage points and F1 by 0.67 percentage points. This indicates that the auxiliary node-attribute branch stabilizes representation learning under heavy class imbalance. E-GraphSAGE and Anomal-E are weaker than XGBoost and LGNT in this setting, which suggests that simple graph embeddings or self-supervised graph representations are not always sufficient for edge-level IoT flow classification.

Fig. 4 reports the binary classification results on NF-ToN-IoT. This task is close to saturation, but LGNT still remains ahead on the main indicators. Compared with LGNT-noAux, the complete model improves AUC by 0.04 percentage points and F1 by 0.02 percentage points. XGBoost remains competitive on this dataset, which means that raw NetFlow features already provide a clear decision boundary. Even so, the additional graph context allows LGNT to classify difficult samples more steadily.

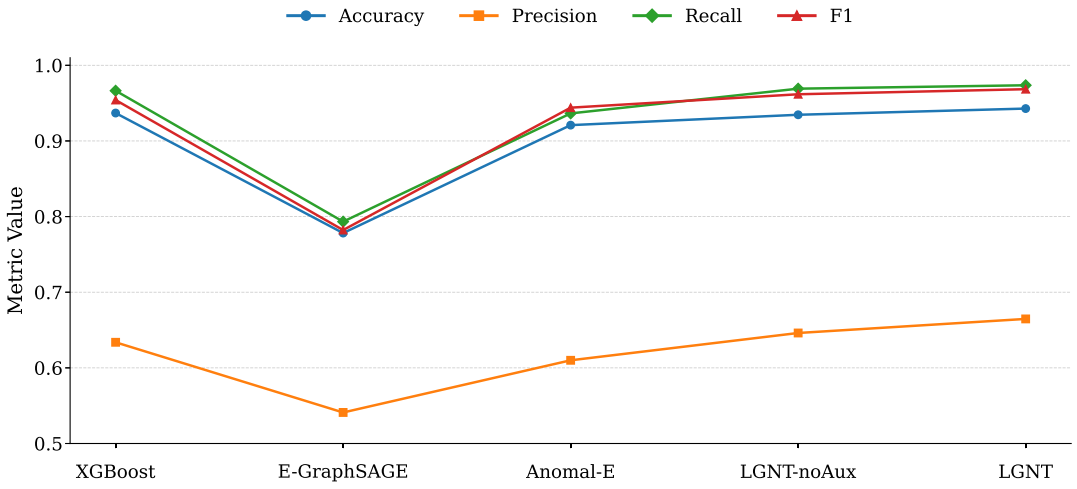

Figure 3: Binary classification results on NF-BoT-IoT. Accuracy, AUC, Precision, Recall, and F1 are compared across XGBoost, E-GraphSAGE, Anomal-E, LGNT-noAux, and LGNT. Note: NF-BoT-IoT, NetFlow BoT-IoT; AUC, area under the curve; XGBoost, Extreme Gradient Boosting; LGNT, Lightweight Graph-Attentive Network for Traffic Detection; LGNT-noAux, LGNT without auxiliary node supervision.

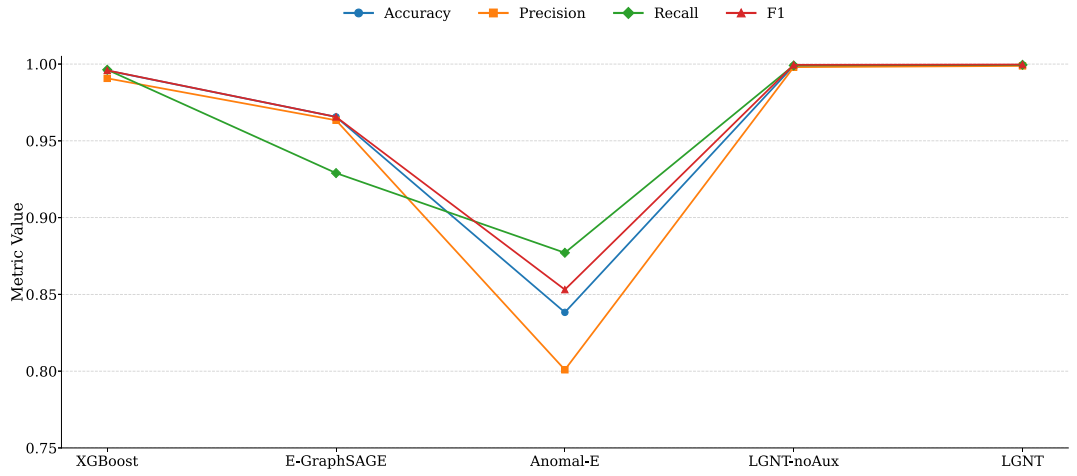

Figure 4: Binary classification results on NF-ToN-IoT. Accuracy, AUC, Precision, Recall, and F1 are compared across the baseline models and LGNT variants. Note: NF-ToN-IoT, NetFlow ToN-IoT; AUC, area under the curve; LGNT, Lightweight Graph-Attentive Network for Traffic Detection.

4.3 Multi-Class Classification Results

Multi-class classification is more difficult because the model must distinguish fine-grained attack behaviors. Fig. 5 shows that LGNT improves weighted F1 from 0.8411 to 0.8688 on NF-BoT-IoT and from 0.6433 to 0.6905 on NF-ToN-IoT.

Table 3 gives the per-class results on NF-BoT-IoT. LGNT maintains strong recall and F1 on the dominant Reconnaissance class, indicating that the model captures high-frequency scanning and reconnaissance patterns. Theft, DoS, and DDoS remain more difficult because their class proportions are lower. The Benign class also has a relatively low F1 score, which is consistent with the severe imbalance of the dataset.

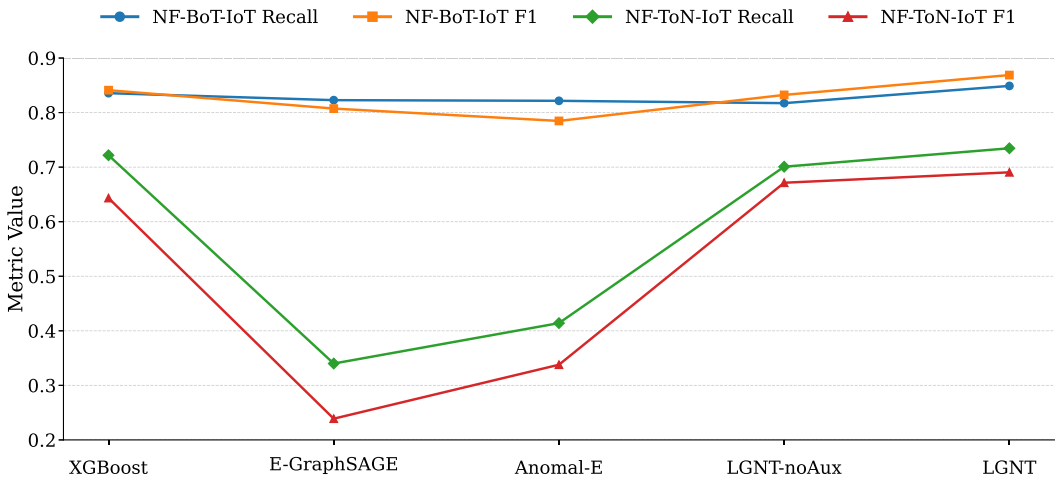

Figure 5: Overall multi-class classification results. The weighted recall and weighted F1 scores are reported for NF-BoT-IoT and NF-ToN-IoT. Note: NF-BoT-IoT, NetFlow BoT-IoT; NF-ToN-IoT, NetFlow ToN-IoT; F1, F1-score.

Table 3: NF-BoT-IoT per-class multi-class results.

Class	Samples	Ratio	Recall	F1
Benign	13,859	0.0231	0.9412	0.6145
Theft	1909	0.0032	0.6210	0.7318
DoS	56,833	0.0947	0.4623	0.6014
DDoS	56,844	0.0947	0.4881	0.4567
Recon.	470,655	0.7843	0.9428	0.9589
Weighted avg.	–	–	0.8489	0.8688

Note: NF-BoT-IoT, NetFlow BoT-IoT; DoS, denial-of-service; DDoS, distributed denial-of-service; Recon., reconnaissance; F1, F1-score.

Although the NF-ToN-IoT results show relatively high accuracy and AUC, these overall metrics alone are insufficient for evaluating fine-grained multi-class malicious traffic detection under severe class imbalance. Therefore, Table 4 further reports per-class precision, recall, F1-score, and support for the NF-ToN-IoT multi-class task. Macro, micro, and weighted averages are also included to provide a more complete evaluation of both minority-class recognition and overall sample-level performance.

As shown in Table 4, the weighted-F1 score is 0.6901, whereas the macro-F1 score is lower at 0.5396. This gap indicates that the overall weighted performance is mainly supported by dominant or more separable classes such as Benign, DDoS, and Injection. In contrast, minority or behaviorally similar attack categories, including DoS, Password, Ransomware, Scanning, and XSS, still show lower recall and F1-scores. The row-normalized confusion matrix in Fig. 6 further illustrates the source of this inconsistency: DoS traffic is frequently confused with DDoS, and XSS and Password samples are often confused with Injection or DDoS. Therefore, the high accuracy and AUC should be interpreted together with per-class and macro-average metrics.

Table 4: NF-ToN-IoT per-class multi-class results.

Class	Support	Precision	Recall	F1-Score
Benign	54,056	0.9962	1.0000	0.9981
Backdoor	3467	0.9980	0.9916	0.9948
DDoS	65,462	0.6864	0.8970	0.7777
DoS	3524	0.3847	0.1595	0.2255
Injection	93,394	0.6659	0.6853	0.6755
MITM	261	0.6888	0.6360	0.6614
Password	31,277	0.4913	0.2387	0.3213
Ransomware	16	0.6667	0.1250	0.2105
Scanning	4263	0.7985	0.0966	0.1724
XSS	20,134	0.3691	0.3495	0.3590
Macro avg.	275,854	0.6745	0.5179	0.5396
Micro avg.	275,854	0.7100	0.7100	0.7100
Weighted avg.	275,854	0.6967	0.7100	0.6901

Note: NF-ToN-IoT, NetFlow ToN-IoT; DoS, denial-of-service; DDoS, distributed denial-of-service; MITM, man-in-the-middle; XSS, cross-site scripting; F1, F1-score.

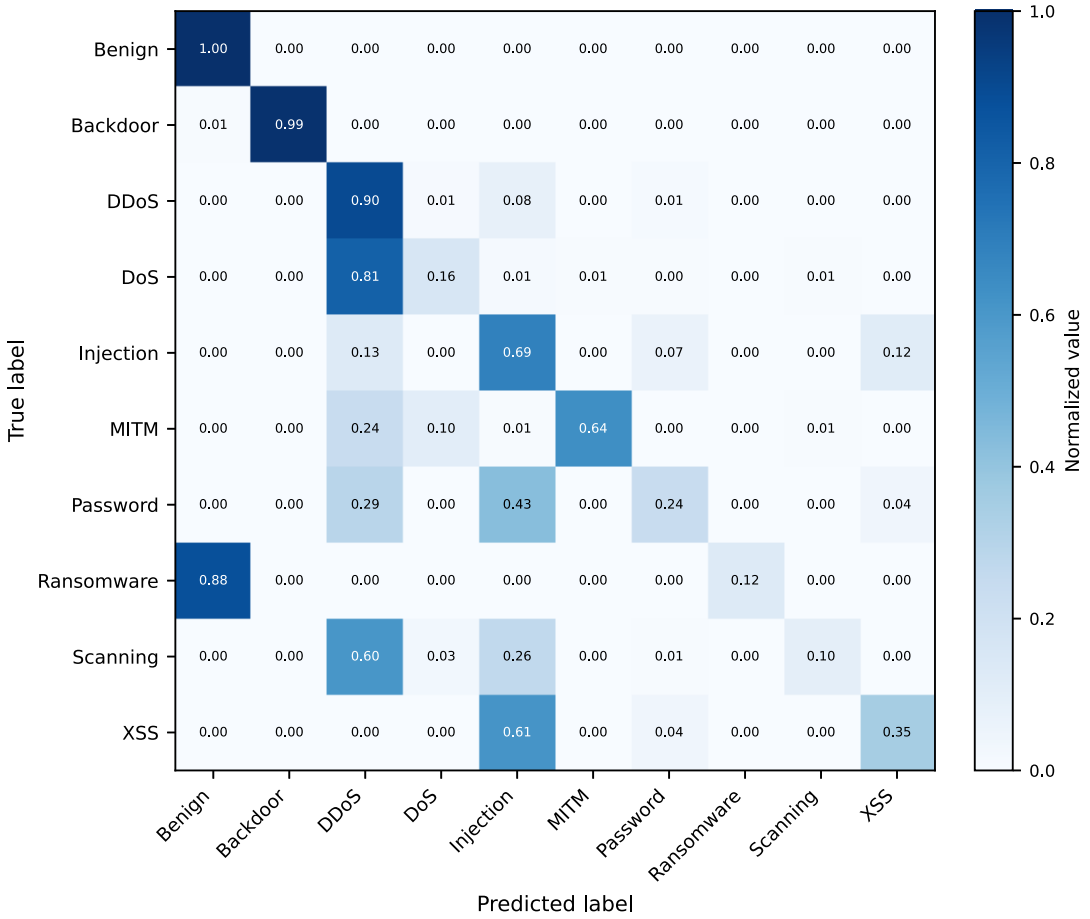

Figure 6: Row-normalized confusion matrix on NF-ToN-IoT. Note: NF-ToN-IoT, NetFlow ToN-IoT; DoS, denial-of-service; DDoS, distributed denial-of-service; MITM, man-in-the-middle; XSS, cross-site scripting.

To further discuss the generalization behavior of LGNT, we compare its performance across NF-BoT-IoT and NF-ToN-IoT under both binary and multi-class settings. The results indicate that LGNT maintains stable performance on binary detection tasks, suggesting that the proposed traffic-interaction graph and lightweight graph encoder can capture discriminative malicious-flow patterns across different IoT traffic sources. However, the multi-class results show larger performance variation, especially on NF-ToN-IoT, where several minority attack categories remain difficult to distinguish. This phenomenon indicates that cross-dataset generalization is affected not only by the graph encoder, but also by dataset-specific traffic composition, attack-type overlap, and long-tailed class distributions.

4.4 Graph Pruning Analysis

To evaluate the effect of graph pruning, Table 5 compares the unpruned model with pruning variants under not-CV and CV settings. The reported values are Accuracy/Macro-F1. The table explicitly keeps the retained-edge ratios so that the pruning settings can be traced during reproduction.

Table 5: Pruning results of LGNT (Accuracy/Macro-F1).

Setting	Task	Unpruned	ρ = 0.5 or 0.7	ρ = 0.7 or 0.8
not-CV	NF-BoT-IoT Binary	0.9403/0.7008	0.9364/0.6917 (ρ = 0.5)	0.9267/0.6728 (ρ = 0.7)
not-CV	NF-BoT-IoT Multi-Class	0.7862/0.5666	0.8369/0.5642 (ρ = 0.5)	0.8053/0.5939 (ρ = 0.7)
not-CV	NF-ToN-IoT Binary	0.9995/0.9992	0.9940/0.9904 (ρ = 0.5)	0.9934/0.9895 (ρ = 0.7)
not-CV	NF-ToN-IoT Multi-Class	0.6808/0.4740	0.6588/0.4082 (ρ = 0.5)	0.6563/0.4279 (ρ = 0.7)
CV	NF-BoT-IoT Binary	0.9343/0.6884	0.9284/0.6748 (ρ = 0.7)	0.9314/0.6826 (ρ = 0.8)
CV	NF-BoT-IoT Multi-Class	0.8214/0.6105	0.8123/0.6007 (ρ = 0.7)	0.8318/0.6136 (ρ = 0.8)
CV	NF-ToN-IoT Binary	0.9990/0.9983	0.9909/0.9857 (ρ = 0.7)	0.9933/0.9896 (ρ = 0.8)
CV	NF-ToN-IoT Multi-Class	0.6704/0.4631	0.6596/0.4121 (ρ = 0.7)	0.6564/0.4476 (ρ = 0.8)

Note: LGNT, Lightweight Graph-Attentive Network for Traffic Detection; CV, cross-validation; NF-BoT-IoT, NetFlow BoT-IoT; NF-ToN-IoT, NetFlow ToN-IoT; F1, F1-score; ρ, retained-edge ratio.

Table 5 shows that pruning is strongly dataset- and task-dependent. On NF-BoT-IoT multi-class classification, moderate pruning can improve Macro-F1. In the not-CV setting, ρ = 0.7 increases Macro-F1 from 0.5666 to 0.5939. In the CV setting, ρ = 0.8 increases Macro-F1 from 0.6105 to 0.6136. This suggests that the NF-BoT-IoT graph contains redundant edges and that mild sparsification can suppress noisy propagation. On NF-ToN-IoT, pruning generally degrades performance, which indicates that fine-grained class boundaries depend more strongly on the complete relational context.

4.5 Ablation and Lightweight Analysis

Table 6 summarizes the internal ablation results. Removing the auxiliary branch reduces multi-class F1 from 0.8688 to 0.8324 on NF-BoT-IoT and from 0.6905 to 0.6714 on NF-ToN-IoT. Removing TAGConv or GATv2 also weakens performance. This confirms that the final performance does not come from a single component, but from the combination of multi-hop aggregation, local attention, and node-attribute auxiliary supervision.

To further evaluate the auxiliary node-supervision branch, Table 7 compares the full model with a variant that removes the auxiliary branch and reports threshold sensitivity. Removing the branch reduces Macro-F1 and Weighted-F1, while moderate threshold changes around the mean cause limited fluctuations.

Table 6: Internal ablation results of LGNT (F1).

Model	Bot-Bin.	Bot-Mul.	ToN-Bin.	ToN-Mul.
LGNT	0.9684	0.8688	0.9997	0.6905
LGNT-noAux	0.9617	0.8324	0.9995	0.6714
w/o TAGConv	0.9589	0.8211	0.9992	0.6587
w/o GATv2	0.9554	0.8138	0.9990	0.6479
Only TAGConv	0.9488	0.7892	0.9986	0.6217
Only GATv2	0.9516	0.7974	0.9988	0.6335

Note: LGNT, Lightweight Graph-Attentive Network for Traffic Detection; NoAux, without auxiliary node supervision; TAGConv, topology adaptive graph convolution; GATv2, graph attention v2 convolution; Bot-Bin., NF-BoT-IoT binary; Bot-Mul., NF-BoT-IoT multi-class; ToN-Bin., NF-ToN-IoT binary; ToN-Mul., NF-ToN-IoT multi-class; F1, F1-score.

Table 7: Auxiliary supervision ablation and threshold sensitivity.

Dataset	Variant/Threshold	Accuracy	Macro-F1	Weighted-F1
NF-BoT-IoT	w/o Auxiliary branch	0.9361	0.5914	0.8447
NF-BoT-IoT	0.8× mean	0.9397	0.6123	0.8619
NF-BoT-IoT	mean (LGNT)	0.9428	0.6216	0.8688
NF-BoT-IoT	1.2× mean	0.9406	0.6085	0.8627
NF-ToN-IoT	w/o Auxiliary branch	0.9991	0.4527	0.6714
NF-ToN-IoT	0.8× mean	0.9992	0.4586	0.6813
NF-ToN-IoT	mean (LGNT)	0.9993	0.4740	0.6905
NF-ToN-IoT	1.2× mean	0.9991	0.4632	0.6759

Note: LGNT, Lightweight Graph-Attentive Network for Traffic Detection; NF-BoT-IoT, NetFlow BoT-IoT; NF-ToN-IoT, NetFlow ToN-IoT; F1, F1-score.

To further support the lightweight design of LGNT, we expanded the efficiency evaluation beyond parameter count. Because sparse graph operations are affected by graph density, retained-edge ratio, and sampling behavior, theoretical operation counts alone may not fully reflect the actual deployment cost. Therefore, we report parameter number, model size, peak GPU memory consumption, batch runtime, inference latency, throughput, and theoretical graph complexity under the same hardware environment. Runtime was measured in inference mode on the same RTX 3090 GPU, with model.eval() enabled and gradient computation disabled; 10 warm-up batches were excluded, and the reported latency values were averaged over 100 timed batches using CUDA synchronization.

As shown in Table 8, graph pruning does not change the parameter number or model size because the model weights remain unchanged. Instead, it reduces the retained message-passing edge set, which lowers peak GPU memory consumption, shortens batch runtime, reduces inference latency, and improves throughput. For example, Prune05 reduces peak GPU memory from 1380 to 1040 MB and improves throughput from 5610 to 8040 flows/s. These results show that the lightweight design of LGNT is reflected not only by the small parameter scale, but also by practical deployment indicators such as memory consumption, inference latency, and throughput.

Table 8: Runtime and latency comparison of LGNT.

Model	Params (M)	Size (MB)	GPU Mem. (MB)	Bot (s/batch)	ToN (s/batch)	p50 (ms)	p95 (ms)	Throughput (flows/s)
LGNT	0.236	0.96	1380	0.742	0.598	5.84	7.12	5610
Prune05	0.236	0.96	1040	0.521	0.431	4.03	5.08	8040
Prune07	0.236	0.96	1165	0.586	0.468	4.61	5.79	7160
Prune08	0.236	0.96	1260	0.633	0.503	4.97	6.24	6640

Note: LGNT, Lightweight Graph-Attentive Network for Traffic Detection; Params, parameters; GPU, graphics processing unit; mem., memory; Bot, NF-BoT-IoT; ToN, NF-ToN-IoT; p50, 50th percentile latency; p95, 95th percentile latency.

In addition to measured runtime indicators, Table 9 provides a theoretical complexity comparison between LGNT and baseline models. The main computational cost of LGNT comes from TAGConv aggregation and GATv2-based graph attention over the retained message-passing edge set. In this table, E_p denotes the pruned message-passing edge set, E denotes the original edge set, K is the TAGConv propagation order, H is the number of attention heads, d is the hidden dimension, C is the number of traffic classes, L is the number of graph layers, T is the number of trees, and D_t is the tree depth. After graph pruning, $|E_p|$ is smaller than $|E|$, so the cost of graph convolution and graph attention is reduced without decreasing the number of trainable parameters.

Table 9: Theoretical complexity comparison of LGNT and baselines.

Model/Module	Inference Complexity
TAGConv aggregation	$O(K\|E_p\|d)$
Local GATv2 attention	$O(H\|E_p\|d)$
Auxiliary node branch	$O(\|V\|d)$
Edge readout and classification	$O(\|E\|dC)$
LGNT forward path	$O(K\|E_p\|d + H\|E_p\|d + \|V\|d + \|E\|dC)$
XGBoost inference	$O(TD_t)$ per sample
E-GraphSAGE message passing	$O(L\|E\|d)$
Anomal-E graph encoder	$O(L\|E\|d)$

Note: LGNT, Lightweight Graph-Attentive Network for Traffic Detection; TAGConv, topology adaptive graph convolution; GATv2, graph attention v2 convolution; XGBoost, Extreme Gradient Boosting.

Overall, the experimental results indicate that LGNT improves edge-level malicious traffic detection while keeping the parameter scale low. The advantage is clearer in multi-class classification, where class imbalance and fine-grained attack similarity make the task harder. The pruning results also show that structural compression is useful, but it should be treated as a dataset-dependent strategy rather than a universally beneficial configuration.

5 Conclusions

This paper presented LGNT, a Lightweight Graph-Attentive Network for Traffic Detection in IoT environments. Instead of treating each NetFlow record as a separate tabular instance, LGNT organizes records into a directed traffic interaction graph. The final model combines TAGConv, GATv2-based local graph attention, auxiliary node-attribute supervision, and static graph pruning to obtain a compact detector.

Experiments on NF-BoT-IoT and NF-ToN-IoT show that LGNT is effective in both binary and multi-class settings. The per-class analysis and normalized confusion matrix also show that fine-grained recognition of minority attack categories remains challenging under severe class imbalance. The pruning and efficiency analyses further suggest that lightweight deployment can be improved through structure compression, although the benefit depends on the underlying dataset.

Future work will evaluate LGNT on live operational traffic, explore adaptive pruning, and incorporate richer sources such as protocol semantics, logs, and threat intelligence. Attention visualization and key-edge analysis may also improve interpretability in security monitoring and attack tracing scenarios.

Acknowledgement: Not applicable.

Funding Statement: This research was funded by the 2026 Young Teachers Interdisciplinary Scientific Research Training Program of Lanzhou University of Technology, grant number LUTXKJC-26003, and the Lanzhou University of Technology Doctoral Talent Research Start-up Fund Project (14/062402).

Author Contributions: The authors confirm contribution to the paper as follows: conceptualization, Baofeng Duan and Xinghai Yu; methodology, Xinghai Yu and Baofeng Duan; software, Xinghai Yu; validation, Baofeng Duan and Xinghai Yu; formal analysis, Xinghai Yu; investigation, Xinghai Yu; resources, Baofeng Duan; data curation, Xinghai Yu; writing—original draft preparation, Xinghai Yu; writing—review and editing, Baofeng Duan, Xinghai Yu, Peng Wang, Tao Feng and Yongbo Jiang; visualization, Xinghai Yu; supervision, Baofeng Duan and Yongbo Jiang; project administration, Baofeng Duan; funding acquisition, Baofeng Duan. All authors reviewed and approved the final version of the manuscript.

Availability of Data and Materials: This study used publicly available NetFlow intrusion detection datasets described in Sarhan et al. [5]. The source code is available from the corresponding author upon reasonable request.

Ethics Approval: Not applicable.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Zanella A, Bui N, Castellani A, Vangelista L, Zorzi M. Internet of Things for smart cities. IEEE Internet Things J. 2014;1(1):22–32. doi:10.1109/JIOT.2014.2306328.
2. Khan MA, Salah K. IoT security: review, blockchain solutions, and open challenges. Future Gener Comput Syst. 2018;82(15):395–411. doi:10.1016/j.future.2017.11.022.
3. Caballero-Gil C, Molina-Gil J, Hernandez-Goya C, Diaz-Santos S, Burmester M. Design of a blockchain-based ubiquitous system for the supply chain with autonomous vehicles. Electronics. 2025;14(23):4744. doi:10.3390/electronics14234744.
4. Yang J, Ali MU, Kumar G, Khan MA, Shaikh ZA, Govindarajan V, et al. LLMEdgeSec: LLM-enabled log reasoning for zero-day IoT threat detection. IEEE Internet Things J. 2026;13(15):32546–55. doi:10.1109/JIOT.2026.3686053.
5. Sarhan M, Layeghy S, Moustafa N, Portmann M. NetFlow datasets for machine learning-based network intrusion detection systems. In: Big data technologies and applications. Cham, Switzerland: Springer; 2021. p. 117–35. doi:10.1007/978-3-030-72802-1_9.
6. Sarhan M, Layeghy S, Portmann M. Evaluating standard feature sets towards increased generalisability and explainability of ML-based network intrusion detection. Big Data Res. 2022;30(3):100359. doi:10.1016/j.bdr.2022.100359.
7. Lo WW, Layeghy S, Sarhan M, Gallagher M, Portmann M. E-GraphSAGE: a graph neural network based intrusion detection system for IoT. In: Proceedings of the NOMS 2022-2022 IEEE/IFIP Network Operations and Management Symposium; 2022 Apr 25–29; Budapest, Hungary. p. 1–9. doi:10.1109/NOMS54207.2022.9789878.

8. Caville E, Lo WW, Layeghy S, Portmann M. Anomal-E: a self-supervised network intrusion detection system based on graph neural networks. Knowl-Based Syst. 2022;258(1):110030. doi:10.1016/j.knosys.2022.110030.
9. Bilot T, El Madhoun N, Al Agha K, Zouaoui A. Graph neural networks for intrusion detection: a survey. IEEE Access. 2023;11:49114–39. doi:10.1109/ACCESS.2023.3275789.
10. Yagmur E, Kocak C, Keles A. Graph neural networks and graph autoencoders for intrusion detection systems: a systematic review, taxonomy and comparative analysis. Comput Netw. 2026;288:112598. doi:10.1016/j.comnet.2026.112598.
11. Deng P, Huang Y. Edge-featured multi-hop attention graph neural network for intrusion detection system. Comput Secur. 2025;148(4):104132. doi:10.1016/j.cose.2024.104132.
12. Wang Y, Han Z, Du Y, Li J, He X. BS-GAT: a network intrusion detection system based on graph neural network for edge computing. Cybersecurity. 2025;8(1):27. doi:10.1186/s42400-024-00296-8.
13. Yang T, Chen JC, Deng H, He B. A lightweight intrusion detection algorithm for IoT based on data purification and a separable convolution improved CNN. Knowl-Based Syst. 2024;304(23):112473. doi:10.1016/j.knosys.2024.112473.
14. Li Z, Yao W. A two stage lightweight approach for intrusion detection in Internet of Things. Expert Syst Appl. 2024;257(11):124965. doi:10.1016/j.eswa.2024.124965.
15. Wang Z, Li J, Yang S, Luo X, Li D, Mahmoodi S. A lightweight IoT intrusion detection model based on improved BERT-of-Theseus. Expert Syst Appl. 2024;238(3):122045. doi:10.1016/j.eswa.2023.122045.
16. Azimjonov J, Kim T. Designing accurate lightweight intrusion detection systems for IoT networks using fine-tuned linear SVM and feature selectors. Comput Secur. 2024;137:103598. doi:10.1016/j.cose.2023.103598.
17. Misrak SF, Melaku HM. Lightweight intrusion detection system for IoT with improved feature engineering and advanced dynamic quantization. Discov Internet Things. 2025;5(1):97. doi:10.1007/s43926-025-00203-8.
18. Altaf T, Wang X, Ni W, Yu Q, Liu F, Braun R. A new concatenated multigraph neural network for IoT intrusion detection. Internet Things. 2023;22(1):100818. doi:10.1016/j.iot.2023.100818.
19. Termos M, Ghalmane Z, Brahmia MA, Fadlallah A, Jaber A, Zghal M. GDLC: a new graph deep learning framework based on centrality measures for intrusion detection in IoT networks. Internet Things. 2024;26(19):101214. doi:10.1016/j.iot.2024.101214.
20. Wang L, Wu J, Wang L. ATC-FusionNet: a hybrid deep learning ensemble for network intrusion detection systems. Comput Mater Contin. 2026;88(1):42–10. doi:10.32604/cmc.2026.078591.
21. Albahar MA. A hybrid self-supervised learning framework for advanced persistent threat detection. Comput Mater Contin. 2026;88(1):91. doi:10.32604/cmc.2026.079941.
22. Shi G, Zhu Y, Liu JK, Li X. HeGCL: advance self-supervised learning in heterogeneous graph-level representation. IEEE Trans Neural Netw Learn Syst. 2024;35(10):13914–25. doi:10.1109/TNNLS.2023.3273255.
23. Li Z, Wang Y, van Leeuwen M. Towards automated self-supervised learning for truly unsupervised graph anomaly detection. Data Min Knowl Discov. 2025;39(5):44. doi:10.1007/s10618-025-01115-5.
24. Govindarajan V, Muzamal JH. Advanced cloud intrusion detection framework using graph based features transformers and contrastive learning. Sci Rep. 2025;15(1):20511. doi:10.1038/s41598-025-07956-w.
25. Zhang A, Zhao Y, Zhou C, Zhang T. ResACAG: a graph neural network based intrusion detection. Comput Electr Eng. 2025;122(1):109956. doi:10.1016/j.compeleceng.2024.109956.
26. Li F, Zhang W, Tang H. RHNN-IoT: a robust IoT intrusion detection framework based on reinforced hypergraph representation learning. Future Gener Comput Syst. 2026;176(1):108212. doi:10.1016/j.future.2025.108212.
27. Du J, Zhang S, Wu G, Moura JMF, Kar S. Topology adaptive graph convolutional networks. arXiv:1710.10370. 2017. doi:10.48550/arxiv.1710.10370.
28. Brody S, Alon U, Yahav E. How attentive are graph attention networks? arXiv:2105.14491. 2022. doi:10.48550/arxiv.2105.14491.
29. Chen T, Guestrin C. XGBoost: a scalable tree boosting system. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining; 2016 Aug 13–17; San Francisco, CA, USA. p. 785–94. doi:10.1145/2939672.2939785.