

Article

Enhancing Fine-Grained Encrypted Traffic Classification via Temporal Bi-Directional GraphSAGE

Junbin Yang^{1,2,3,*} , Haihua Shen^{1,*}, Zulong Diao⁴ and Yiran He¹ ¹ School of Computer Science and Technology, University of Chinese Academy of Sciences, Beijing 101408, China; heyiran231@mailsucas.ac.cn² Department of Cyber Security, Shanxi Police College, Taiyuan 030401, China³ Intelligence Research Center, Shanxi Police College, Taiyuan 030401, China⁴ School of Computer Science and Engineering, Hunan University of Science and Technology, Xiangtan 411201, China; diaozulong@hnust.edu.cn

* Correspondence: yangjunbin19@mailsucas.ac.cn (J.Y.); shenh@ucas.ac.cn (H.S.)

Abstract

Encrypted traffic classification is essential for network management and security, yet payload inspection is ineffective under modern protocols such as Transport Layer Security (TLS) and Quick UDP Internet Connections (QUIC). Existing metadata-based methods perform well for coarse-grained tasks but often fail to distinguish structurally similar applications because they model temporal behavior only implicitly or coarsely. We propose the Bi-Directional Directed Temporal Graph (BiDT), a framework based on a Directed Temporal Interaction Graph (DTIG) and a Bi-Directional GraphSAGE (BiGraphSAGE). The DTIG represents packets as nodes and explicitly encodes inter-arrival times (IATs) as directed edge attributes, preserving both causal structure and communication rhythm. The BiGraphSAGE then aggregates temporal interaction features from forward and backward perspectives. We evaluated the BiDT on the VNAT benchmark and validated it on ISCX-VPN. On the challenging 10-class VNAT dataset, the BiDT achieves 98.57% accuracy and outperforms strong baselines, including complete separation of easily confused protocols such as SCP and SFTP. The results on ISCX-VPN further confirm the effectiveness of the proposed design. These findings show that explicit temporal edge modeling is effective for fine-grained encrypted traffic classification.

Keywords: encrypted traffic classification; graph neural networks (GNNs); temporal edge features; fine-grained analysis; inter-arrival time (IAT)



Academic Editor: Christos Bouras

Received: 10 March 2026

Revised: 30 March 2026

Accepted: 31 March 2026

Published: 1 April 2026

Copyright: © 2026 by the authors.

Licensee MDPI, Basel, Switzerland.

This article is an open access article distributed under the terms and conditions of the [Creative Commons Attribution \(CC BY\)](https://creativecommons.org/licenses/by/4.0/) license.

1. Introduction

The rapid growth of encrypted traffic, driven by protocols such as Transport Layer Security (TLS) 1.3 and Quick UDP Internet Connections (QUIC), has changed the landscape of network security and traffic analysis. Encryption protects user data, but it also makes traditional Deep Packet Inspection (DPI) ineffective [1–3]. As a result, encrypted traffic monitoring now relies mainly on metadata, such as packet length, direction, and arrival time [4–7].

In this setting, fine-grained traffic classification has received growing attention. The goal is to distinguish specific services within the same application family, such as Netflix and YouTube, rather than only separating broad categories like Video and Chat. This level of granularity is important for accurate Quality of Service (QoS) provisioning and threat detection [8]. Yet the task remains difficult [5,9]. Applications in the same family

often share the same encryption protocols and similar handshake procedures, which leads to highly similar side-channel patterns [10]. Streaming platforms such as Netflix and Vimeo are a typical example because their adaptive bitrate mechanisms produce very similar buffering behavior [5].

Early studies mainly relied on statistical feature engineering [11,12]. These methods are useful for describing global traffic characteristics, but they depend heavily on handcrafted features and often miss fine-grained sequential and interaction information. To address these inherent drawbacks, deep learning [13] has gradually been introduced into network traffic analysis tasks as it enables automatic and adaptive feature extraction from raw or preprocessed traffic data without relying on labor-intensive handcrafted features. This automatic feature learning capability allows deep learning models to mine potential patterns hidden in traffic data that are difficult to define manually. Sequence-based and image-based deep learning models [1,14,15] have also shown strong ability in capturing local patterns. Federated learning has also been applied to encrypted traffic identification to protect private traffic data and user privacy [16,17]. Even so, it still struggles to model the non-Euclidean and interactive structure of network exchanges. This limitation has motivated the use of Graph Neural Networks (GNNs), which represent traffic as interaction graphs and learn from topological dependencies [18,19]. Existing studies, such as IBGC [9] and GraphDApp [5], show that mapping packet bursts to graph nodes is effective for encrypted traffic classification.

Even with these advances, current graph-based methods still face clear limitations in fine-grained settings. One issue is directionality. Many GNN-based methods construct undirected graphs, which can hide the request–response structure of client–server communication. The other issue is temporal modeling. In many existing studies, time is represented only through flow-level statistics or implicit positional order, rather than through explicit edge-level interaction delays [20]. These two limitations reduce the model’s ability to capture sequential causality and communication pacing. As a result, it remains difficult to distinguish services that are structurally similar but temporally different.

To address these issues, we propose the Bi-Directional Directed Temporal Graph (BiDT), a framework built on a Directed Temporal Interaction Graph (DTIG). The DTIG uses only three protocol-agnostic metadata features: packet size, direction, and arrival time. It maps packet sizes to node semantics and encodes inter-arrival time (IAT) directly as directed edge attributes. In this way, the graph preserves both causal order and temporal pacing. On top of the DTIG, we designed a Bi-Directional GraphSAGE (BiGraphSAGE) model. The BiGraphSAGE aggregates spatio-temporal information from both forward (causal) and backward (retrospective) directions, which allows later responses to provide context for earlier encrypted packets.

We evaluated the proposed framework on the primary VNAT benchmark and conducted a validation experiment on ISCX-VPN. On the challenging 10-class fine-grained VNAT benchmark, the BiDT achieves 98.57% accuracy and shows strong discrimination among highly similar applications, including Netflix, Vimeo, and YouTube. We further tested the model on ISCX-VPN to examine whether the same design remains effective on an additional public benchmark.

The main contributions of this paper are:

1. We analyzed the limited temporal modeling in existing traffic graph methods and propose the DTIG. Using only three protocol-agnostic metadata features, the DTIG represents request–response directionality with directed edges and embeds IAT into edge attributes to capture communication pacing.

2. We designed the BiGraphSAGE to extend standard GraphSAGE to edge-aware bi-directional aggregation. Tailored to the DTIG, it integrates temporal edge attributes and captures dependencies from both forward and backward directions.
3. We evaluated the proposed method on VNAT and ISCX-VPN. On the fine-grained VNAT benchmark, the BiDT reaches 98.57% accuracy. The results indicate that the framework can distinguish structurally similar encrypted traffic classes effectively.

The central research hypothesis of this study is that explicit modeling of edge-level temporal rhythm (via directed IATs) combined with bi-directional message passing significantly improves the model's ability to discriminate structurally similar but temporally distinct encrypted applications.

The remainder of this paper is organized as follows: Section 2 reviews related work. Section 3 details the proposed BiDT framework, including DTIG construction and the BiGraphSAGE. Section 4 presents the experimental setup, performance evaluation, and ablation studies. Section 5 discusses practical implications and limitations. Finally, Section 6 concludes the paper.

2. Related Work

Encrypted traffic classification has developed substantially over the years. In this section, we review prior studies from five perspectives: traditional methods, statistical feature-based methods, sequence- and image-based methods, pre-trained foundation models, and graph-based methods. We also discuss their limitations in fine-grained classification and clarify the position of the BiDT within this research landscape.

2.1. Traditional Methods

Early traffic classification mainly relied on port-based and payload-based methods. Port-based methods map TCP/UDP port numbers to registered applications (e.g., port 80 for HTTP) [21,22]. They are simple and efficient, but their accuracy has declined because of dynamic port allocation, NAT, and port obfuscation [2,14]. Payload-based methods, often referred to as DPI [23], inspect packet contents for specific signatures. They are highly accurate for unencrypted traffic, but they become ineffective under modern encryption protocols such as TLS and QUIC, where the payload is hidden [24].

2.2. Statistical Feature-Based Methods

Statistical features of traffic flows (e.g., packet length statistics, IAT, and flow duration) can be used in combination with traditional machine learning (ML) classifiers like Random Forest (RF) or C4.5. Taylor et al. proposed AppScanner, which utilizes statistical features of packet size sequences to fingerprint smartphone apps [11]. Panchenko et al. introduced cumulative packet length features for website fingerprinting, achieving high accuracy with support vector machines (SVMs) [25]. Xu et al. applied path signature theory to enhance packet length sequences for effective traffic service classification [26]. These methods rely heavily on manually designed global features. As a result, they may discard the sequential or structural details needed to separate highly similar protocols.

2.3. Sequence-Based and Image-Based Methods

Traffic flows can be treated as time-series sequences or pseudo-images with the application of deep learning models such as convolutional neural networks (CNNs) and long short-term memory (LSTM) networks. Wang et al. pioneered the use of 1D-CNNs for end-to-end encrypted traffic classification, treating raw traffic data as a sequence of bytes [27]. Liu et al. proposed FS-Net, an end-to-end classification model using recurrent neural networks (RNNs) to learn from raw packet length sequences [1]. Sirinam et al.

developed Deep Fingerprinting (DF), leveraging CNNs to extract features from packet direction sequences [28]. Shapira et al. proposed FlowPic, which transforms packet size and time series into 2D images and utilizes 2D-CNNs for classification [15]. These methods are effective at capturing sequential patterns and local spatial features. Still, they often overlook the non-Euclidean interaction topology of network communication, such as request–response bursts, which is more naturally represented as a graph.

2.4. Pre-Trained Traffic Foundation Models

Inspired by the success of Large Language Models (LLMs) in natural language processing (NLP), recent studies have introduced pre-training paradigms into encrypted traffic analysis. These methods treat traffic bytes or packets as tokens and pre-train Transformer-based models on large unlabeled datasets to learn transferable representations. He et al. proposed PERT, which applies ALBERT-based architectures to learn contextual representations from payload bytes [21]. Lin et al. introduced ET-BERT, which adapts BERT to learn from datagram byte sequences using Masked Language Modeling (MLM) [14]. Zhao et al. developed YaTC, a self-supervised framework based on Masked Autoencoders (MAEs) to capture robust traffic features [29].

Although these foundation models often achieve strong performance through large-scale pre-training, they are computationally expensive. They also mainly model one-dimensional token patterns and still lack explicit representations of interaction topology and cross-flow dependencies that are more naturally expressed as graphs [30,31].

2.5. Graph-Based Traffic Methods

Traffic data can be converted into graph structures (e.g., interaction graphs), and GNNs can be applied to learn topological features. Shen et al. proposed GraphDApp, constructing Traffic Interaction Graphs (TIGs) based on packet bursts and using GNNs for decentralized application fingerprinting [5]. Zhang et al. introduced a byte-level traffic graph approach (TFE-GNN), using Point-wise Mutual Information (PMI) to model relationships between bytes [32]. Wang et al. developed MFSI, using Multi-Flow Multi-Relational Graphs (MMRG) and Relational Graph Convolutional Networks (RGCNs) to identify services across multiple correlated flows [30]. Zhang et al. (MH-Net) explore heterogeneous graphs to capture diverse correlations between traffic units [31]. Graph-based methods have clear structural advantages. Even so, most existing approaches, such as IBGC [9], still model network communication with undirected graphs and focus mainly on topology, correlation, or relation existence, often represented as binary links. This design obscures the request–response directionality that is central to client–server interaction. Some studies also include time information but usually through coarse flow-level statistics (e.g., IAT summaries), node attributes, or implicit sequential order [9,33]. While some recent graph or hybrid models incorporate temporal attributes via positional encodings or event-based node features, they predominantly focus on node-level representations [5,9,15,26,31,33–36]. They still largely lack explicit, continuous edge-level interaction delays that directly quantify request–response pacing. As a result, graph edges still carry limited temporal rhythm information, especially for within-class fine-grained distinctions where services share similar structural patterns but differ in pacing, such as Netflix versus Vimeo/YouTube. Our work addresses this gap by introducing a DTIG with explicit temporal edge embeddings.

3. Methodology

In this section, we describe the proposed framework for fine-grained encrypted traffic classification. We first formalize the task and summarize the main challenges. We then present the construction of the DTIG, which explicitly models communication rhythm.

Finally, we introduce the BiGraphSAGE used to learn spatio-temporal representations from the constructed graphs.

3.1. Problem Formulation and Framework Overview

3.1.1. Problem Definitions

Let an encrypted traffic flow be denoted as a sequence of N packets, $\mathcal{F} = \{p_1, p_2, \dots, p_N\}$. Each packet p_i represents an atomic interactive action between the client and the server. We characterize each packet p_i by a tuple (s_i, d_i, t_i) , where s_i is the packet size (in bytes), $d_i \in \{0, 1\}$ denotes the direction (0 for downlink, 1 for uplink), and t_i is the arrival timestamp. The objective of fine-grained traffic classification is to learn a mapping function $f: \mathcal{F} \rightarrow \mathcal{Y}$, where $y \in \mathcal{Y}$ represents the specific application service or protocol variant.

We restrict the input to three protocol-agnostic metadata features only: packet size s_i , packet direction d_i , and packet arrival time t_i . No payload bytes, TLS fields, or handcrafted header fields are used. This design preserves user privacy and helps the model remain applicable under evolving encryption protocols and with deliberate obfuscation.

3.1.2. Main Challenges

To achieve accurate fine-grained classification, we identify two primary challenges that existing methods fail to address effectively:

1. **Characterizing Interactive Rhythm:** Different applications may exhibit identical structural patterns (e.g., similar packet size sequences) but differ significantly in their time distributions (e.g., the distinct buffering dynamics of Netflix vs. YouTube). Standard graph methods often discard this rhythm.
2. **Modeling Bi-Directional Dependency:** Network communication is inherently a dialogue. A request determines the potential response (causality), but observing a response also clarifies the intent of the preceding request (retrospection). Undirected graph models inherently fail to capture this directional dependency.

To address these challenges, we propose the BiDT framework, whose overall architecture is shown in Figure 1.

The framework processes traffic flows through four sequential phases:

1. **Traffic Preprocessing:** We first parse raw traffic files (packet capture and PCAP) and divide them into individual flows based on the 5-tuple. Each flow is then converted into a packet sequence with extracted metadata (size, direction, and timestamp).
2. **DTIG Construction:** The packet sequences are transformed into the DTIG. In each graph, nodes represent packets, and directed edges encode transmission direction. The edges also carry explicit IAT attributes to describe interaction rhythm.
3. **BiGraphSAGE Learning:** The constructed graphs are processed by the BiGraphSAGE. This module uses two parallel branches to aggregate context from both the causal view (forward in time) and the retrospective view (backward in time).
4. **Fusion and Classification:** The representations from both views are fused and passed to a classifier to predict the fine-grained application service.

3.2. Traffic Preprocessing

To construct the DTIG, raw network traffic in PCAP format must first be parsed and divided into individual logical sessions. The preprocessing and abstraction pipeline is defined as follows.

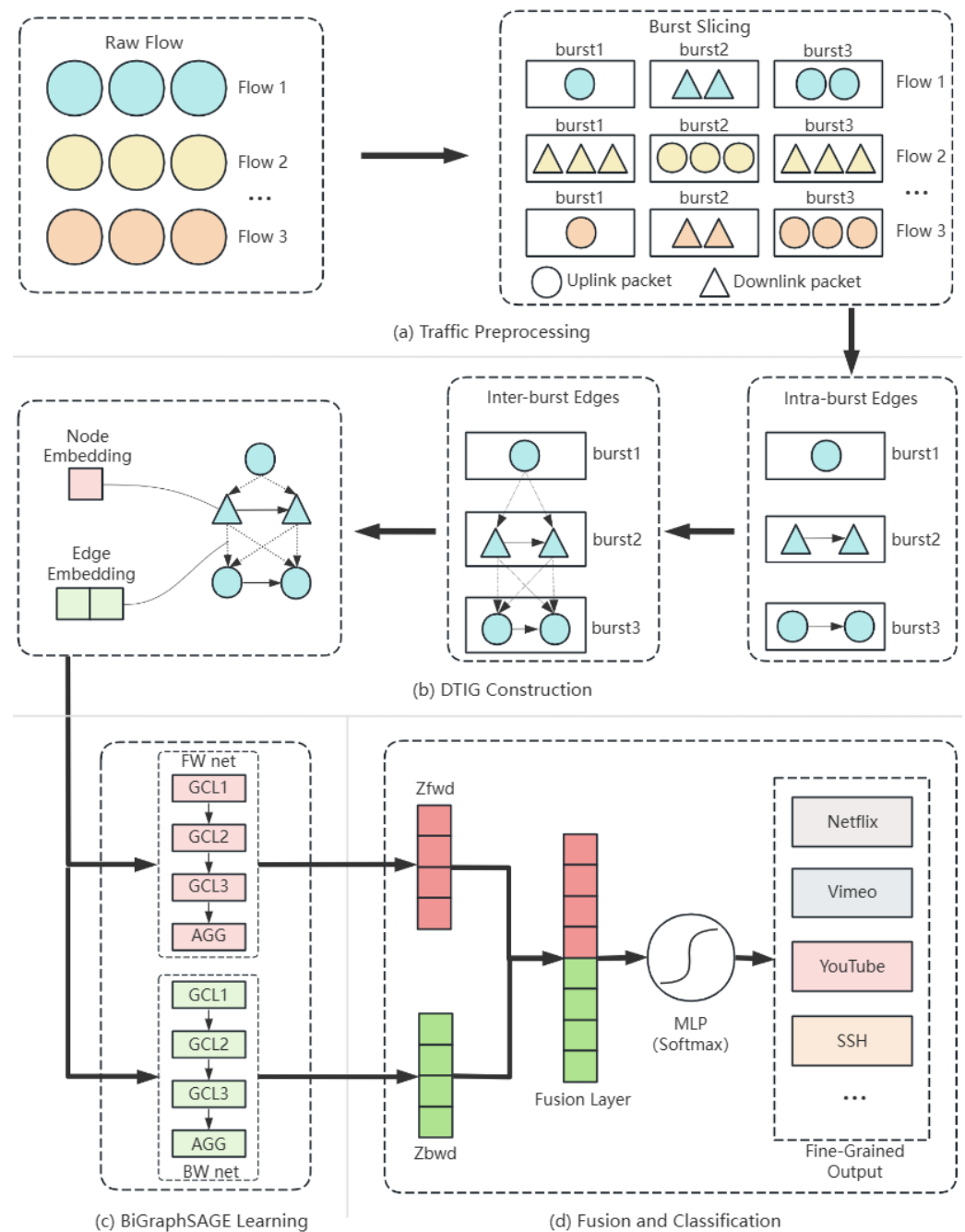


Figure 1. The overall architecture of the proposed BiDT framework. **(a)** Traffic Preprocessing: raw traffic flows are parsed into packet sequences. **(b)** DTIG Construction: packets become nodes, and temporal interactions form directed edges within and between bursts. **(c)** BiGraphSAGE Learning: a dual-path graph network aggregates features from both forward (causal) and backward (retrospective) perspectives. **(d)** Fusion and Classification: fused representations predict fine-grained application classes. Additionally, color coding is used to distinguish components: pink and green blocks represent the initial node and edge embeddings in **(b)**, as well as the forward and backward computation paths in **(c)** and **(d)**, respectively.

3.2.1. Flow Grouping

Encrypted network communications are naturally composed of discrete traffic flows. We logically segregate the raw mixed packets based on the classic 5-tuple, denoted as

$$FlowID = (srcIP, dstIP, srcPort, dstPort, protocol), \quad (1)$$

where $srcIP$ and $dstIP$ denote the source and destination IP addresses, $srcPort$ and $dstPort$ denote the corresponding port numbers, and $protocol$ indicates the transport layer protocol (e.g., TCP or UDP). Packets matching a specific $FlowID$ or its reverse-direction counterpart are merged and ordered chronologically to form an independent bi-directional flow sequence:

$$\mathcal{F} = \{p_1, p_2, \dots, p_N\}, \quad (2)$$

where p_i represents the i -th chronologically ordered packet in the flow, and N denotes the total number of packets in the flow sequence.

3.2.2. Interactive Action Abstraction

Once flows are grouped, we discard payloads, specific IP addresses, and superficial protocol headers. This reduces the risk that the model learns dataset-specific artifacts, such as hardcoded IP subnets, and also preserves user privacy. Instead, we extract only three metadata attributes from each packet: packet size, direction, and arrival time. Consistent with the problem formulation, the original flow is transformed into a temporally ordered metadata sequence:

$$\mathcal{F}_{meta} = \{(s_i, d_i, t_i)\}_{i=1}^N, \quad (3)$$

where s_i is the packet size, $d_i \in \{0, 1\}$ dictates the transmission direction (e.g., 0 for downlink/server-to-client, 1 for uplink/client-to-server), and t_i is the exact arrival timestamp. This step removes irrelevant detail and preserves the interaction context used for the subsequent graph construction.

3.3. DTIG Construction

Unlike previous approaches [5,9] that build undirected graphs around packet bursts, the DTIG is defined as $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{X}, \mathbf{E}_{attr})$. Here, $\mathcal{V}$ is the set of nodes representing individual packets, $\mathcal{E}$ is the set of directed edges representing interaction dependencies, $\mathbf{X}$ denotes the node feature matrix derived from packet sizes, and $\mathbf{E}_{attr}$ represents the temporal attributes (IAT) associated with the edges. This design preserves directionality and embeds temporal intervals directly into graph edges.

3.3.1. Node Representation via Learnable Embeddings

The node set $\mathcal{V} = \{v_1, \dots, v_N\}$ corresponds to the packets in the flow. For each node v_i , the feature vector $\mathbf{x}_i$ is derived principally from the packet size s_i . In standard approaches, packet size is often normalized to a float value in $[0, 1]$. However, this destroys the semantic meaning of specific protocol sizes (e.g., a 60-byte acknowledgment (ACK) packet vs. a 1400-byte data segment).

To preserve these discrete semantics, we use a learnable embedding layer. We map the integer packet size s_i to a dense vector space $\mathbb{R}^{d_{node}}$:

$$\mathbf{h}_i^{(0)} = \text{Embedding}(\min(s_i, S_{max})), \quad (4)$$

where S_{max} is the maximum transmission unit (MTU) size (e.g., 1500), and $\text{Embedding}(\cdot)$ is a lookup table learned during training. This allows the model to associate values such as $s_i = 60$ with control packets without assuming a linear relationship between packet sizes.

3.3.2. Burst-Aware Topology Construction

We construct the edge set $\mathcal{E}$ based on the concept of interactive bursts. An interactive burst B_k is defined as a maximal subsequence of packets starting from index i to $i + m$, having the same direction $d \in \{0, 1\}$:

$$B_k = \{p_i, p_{i+1}, \dots, p_{i+m} \mid d_i = d_{i+1} = \dots = d_{i+m} = d\}. \quad (5)$$

We define two types of directed edges to capture transition patterns:

- Intra-burst edges (Sequential): Within a burst B_k , packets are strictly ordered. We add directed edges $v_j \rightarrow v_{j+1}$ for all $j \in \{i, i+1, \dots, i+m-1\}$. This represents the continuous transmission of data fragments.
- Inter-burst edges (Interactive): Between adjacent bursts B_k and B_{k+1} (where direction switches), we connect all nodes in B_k to all nodes in B_{k+1} .

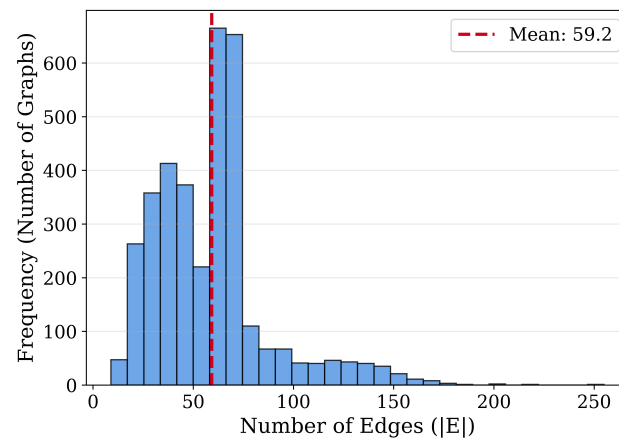
$$\mathcal{E}_{inter} = \{(u, v) \mid u \in B_k, v \in B_{k+1}\}. \quad (6)$$

This dense bipartite connection precisely models the action–response dependency, effectively indicating that the entire request burst collectively triggers the corresponding response burst.

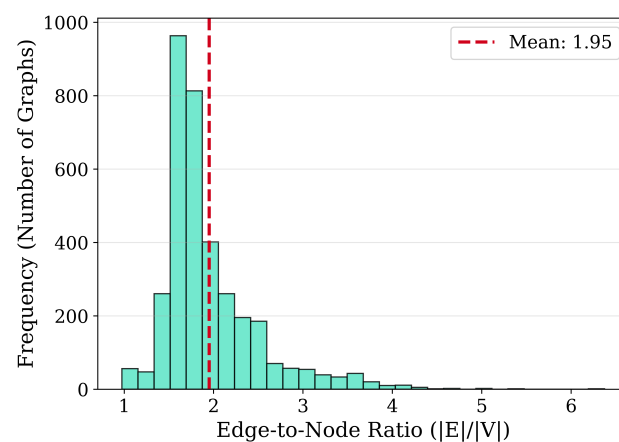
It is crucial to consider the scalability implications of this dense connectivity logic: bipartite inter-burst mapping yields a theoretical worst-case edge complexity of $O(|B_k| \times |B_{k+1}|)$. This characteristic intuitively raises concerns regarding graph memory explosion for continuously enormous data bursts. However, our proposed architecture natively circumvents this bottleneck via two fundamental mechanisms. First, the robust application-layer “request–response” interaction rhythm and Transmission Control Protocol (TCP) acknowledgment mechanisms natively prevent continuous unidirectional massive transfers, keeping individual burst sizes ($|B_k|$) practically small. Second, we strictly constrain the flow sequence to a maximal boundary of $N = 40$ packets.

To empirically validate the topological stability under these constraints, we performed a comprehensive graph complexity analysis on the VNAT dataset, purposefully filtering out trivial noise sequences containing fewer than 5 packets to ensure statistical rigor. As illustrated in Figure 2, the empirical distributions convincingly confirm that the generated DTIGs are inherently sparse. Specifically, for valid graphs processing an average of 30.04 nodes, the average number of edges is merely 59.21 (Figure 2a). More conclusively, the average Edge-to-Node ratio ($|E|/|V|$) rigidly sits at 1.95 (Figure 2b), meaning the constructed topology statistically approximates a tree-like cascading structure rather than a dense combinatorial mesh, yielding an extremely low overall average graph density of 7.57%. Even the absolute worst-case scenario across all evaluated instances maxes out at only 255 edges. This profound topological sparsity guarantees deterministic memory footprints that modern Message Passing Neural Networks can compute effortlessly.

Nonetheless, we acknowledge that for future variations adapting the model to untruncated high-throughput long-lived flows without length boundary limitations, integrating a sparser alternative inter-burst schema would be an essential consideration to maximize systemic scalability. Promising simpler alternatives to our dense bipartite design include structural heuristics—such as exclusively connecting physical boundary endpoints (e.g., first-to-first or last-to-first edges) or localized centroid nodes (representative-to-representative edges) of adjacent bursts. Alternatively, employing attention-based burst linking could dynamically weight the most critical inter-burst dependencies while maintaining computational efficiency. Exploring these sparse mechanisms to handle unbounded session lengths remains a key direction for our future work.



(a) Absolute edge count distribution.



(b) Edge-to-node ratio distribution.

Figure 2. Empirical topological complexity analysis of the generated DTIGs under the $N \leq 40$ sequence boundary ($N \geq 5$ filtered). (a) The frequency distribution showing the total number of graphical edges ($|E|$) per graph. (b) The Edge-to-Node ratio ($|E|/|V|$) distribution highlighting extreme topological sparsity around 1.95.

3.3.3. Temporal Edge Embedding

To encode interaction rhythm, we embed IAT directly into graph edges. For any directed edge $(v_i, v_j) \in \mathcal{E}$ (where $1 \leq i, j \leq N$), let $\Delta t_{ij} = |t_j - t_i|$ represent the raw time interval (in seconds) between packet i and packet j . Directly using raw IAT values creates difficulties for neural network optimization. First, raw IATs span a wide dynamic range, from microseconds (rapid intra-burst packet fragmentation) to several seconds (user think-time or TCP keep-alive delays). Second, very fine-grained intervals may contain meaningless variation caused by operating system scheduling jitter and network queuing latency.

To address these issues, we designed a dual-channel temporal edge embedding $\mathbf{e}_{ij} \in \mathbb{R}^2$, where $(v_i, v_j) \in \mathcal{E}$. The embedding $\mathbf{e}_{ij}$ is formulated as the concatenation of a log-quantization channel e_{ij}^{log} and a semantic zero indicator e_{ij}^{zero} :

$$\mathbf{e}_{ij} = [e_{ij}^{log}, e_{ij}^{zero}], \quad \forall (v_i, v_j) \in \mathcal{E}. \quad (7)$$

e_{ij}^{log} and e_{ij}^{zero} are defined and function as follows:

- e_{ij}^{log} : Unprocessed IATs form a heavily right-skewed distribution. By applying the transformation $e_{ij}^{log} = \log(1 + \text{Round}(\Delta t_{ij} \times s))$, where s denotes a scaling factor that

defines the temporal quantization granularity, we map the continuous Δt_{ij} into discrete bins and smooth out micro-level hardware latency noise. The logarithmic compression then reduces the numerical dominance of large delays (e.g., user think-times), which helps stabilize optimization.

- e_{ij}^{zero} : We introduce a boolean indicator $e_{ij}^{zero} = \mathbb{I}(\text{Round}(\Delta t_{ij} \times s) = 0)$, which outputs 1 only when the quantized interval is exactly zero. This serves as an explicit structural marker for topological bursts. It informs the model that the current packet pair belongs to an uninterrupted, high-speed transmission phase. The model can therefore distinguish rapid intra-burst fragments from delayed interactive responses more clearly.

In our empirical implementation, the choice of the scaling factor s balances noise reduction and temporal resolution. As shown later in our sensitivity analysis, we set the quantization precision to 100 μs ($s = 10^4$). This granularity filters out negligible sub-100-microsecond operating system (OS) jitter while preserving the millisecond-level request–response pacing needed to distinguish structurally similar protocols (e.g., adaptive streaming variations).

These steps complete the construction of the DTIG for an encrypted flow. The nodes correspond to individual interactive actions (packets), and each edge represents a transition relation between actions. Because client–server interaction is a sequence of causally linked events, the edges are strictly directed and carry explicit temporal rhythm embeddings. Given an encrypted flow’s interactive process, as shown in Figure 3a, the resulting DTIG structure is illustrated in Figure 3b. In the graph, interactive actions are numbered according to their order in the communication process. Light blue denotes uplink packets, and light orange denotes downlink packets.

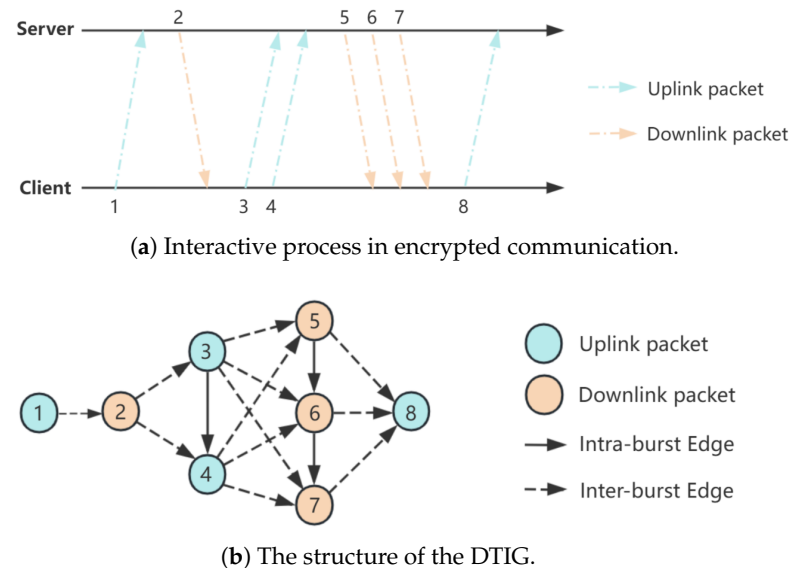


Figure 3. Correspondence between the interactive process and the constructed DTIG. The numbers (1–8) indicate the chronological order of the packets in the communication process.

3.4. BiGraphSAGE

To learn from the DTIG with explicit temporal edge attributes, we propose the BiGraphSAGE architecture. Standard GraphSAGE is a representative backbone in prior graph-based traffic models, but it aggregates only node features and does not directly handle edge attributes. To address this limitation, we designed the BiGraphSAGE as an edge-aware and bi-directional representation learning model for directed temporal graphs.

Unlike standard GNNs that perform message passing in one direction, the BiGraphSAGE aggregates context from both the causal (forward-time) and retrospective (backward-time) perspectives.

Let $l \in \{0, 1, \dots, L-1\}$ denote the layer index, where L is the total number of BiGraphSAGE layers.

3.4.1. Edge-Integrated Aggregation Mechanism

To model temporal rhythm, we extend GraphSAGE neighbor aggregation to incorporate edge features. For a target node i and its neighbor j connected by a directed edge $(v_j, v_i) \in \mathcal{E}$, we concatenate the neighbor feature $\mathbf{h}_j \in \mathbb{R}^{d_h}$ with the temporal edge embedding $\mathbf{e}_{ji} \in \mathbb{R}^2$. The model then aggregates these edge-conditioned contexts using mean pooling:

$$\mathbf{m}_{\mathcal{N}(i)} = \text{Mean}_{j \in \mathcal{N}(i)} \left(\mathbf{W}_{\text{neighbor}} [\mathbf{h}_j \parallel \mathbf{e}_{ji}] \right), \quad (8)$$

where $i \in \{1, 2, \dots, N\}$ is the index of the target node, j represents the index of a neighbor node such that $j \in \mathcal{N}(i)$ (i.e., $(v_j, v_i) \in \mathcal{E}$), and $\mathbf{W}_{\text{neighbor}} \in \mathbb{R}^{d_m \times (d_h+2)}$ maps the concatenated representation to a message space. The aggregated neighborhood context $\mathbf{m}_{\mathcal{N}(i)} \in \mathbb{R}^{d_m}$ is then concatenated with the target node's linearly transformed feature $\mathbf{W}_{\text{self}} \mathbf{h}_i$ (with $\mathbf{W}_{\text{self}} \in \mathbb{R}^{d_m \times d_h}$) and passed through a linear transformation to update its state:

$$\mathbf{h}_i^{(l+1)} = \sigma \left(\mathbf{W}_{\text{update}} [\mathbf{W}_{\text{self}} \mathbf{h}_i^{(l)} \parallel \mathbf{m}_{\mathcal{N}(i)}^{(l)}] \right), \quad (9)$$

where $\mathbf{W}_{\text{update}} \in \mathbb{R}^{d'_h \times 2d_m}$ ensures the final updated dimension is d'_h . When $\mathcal{N}(i) = \emptyset$, we set $\mathbf{m}_{\mathcal{N}(i)} = \mathbf{0}$. This mechanism lets the aggregation layer combine temporal interaction distance with neighboring packet semantics and thus model communication pacing directly.

3.4.2. Dual-Path Aggregation Strategy

We employ two parallel edge-aware GraphSAGE layers operating on different graph topologies:

1. **Forward Path (Causal View):** This path operates on the original directed edge set $\mathcal{E}$. It aggregates messages from past packets to update the current packet state. The neighborhood $\mathcal{N}_{in}(i)$ is defined on the original graph structure:

$$\vec{\mathbf{h}}_i^{(l+1)} = \sigma \left(\mathbf{W}_{up}^{fwd} [\mathbf{W}_{self}^{fwd} \vec{\mathbf{h}}_i^{(l)} \parallel \vec{\mathbf{m}}_{\mathcal{N}_{in}(i)}^{(l)}] \right). \quad (10)$$

2. **Backward Path (Retrospective View):** This path operates on the transposed edge set $\mathcal{E}^T = \{(v, u) \mid (u, v) \in \mathcal{E}\}$. It aggregates information from future responses back to earlier requests. Here, $\mathcal{N}_{out}(i)$ acts as the reversed neighbor set within the transposed graph. This path helps interpret encrypted handshakes whose roles depend strongly on subsequent server responses:

$$\overleftarrow{\mathbf{h}}_i^{(l+1)} = \sigma \left(\mathbf{W}_{up}^{bwd} [\mathbf{W}_{self}^{bwd} \overleftarrow{\mathbf{h}}_i^{(l)} \parallel \overleftarrow{\mathbf{m}}_{\mathcal{N}_{out}(i)}^{(l)}] \right). \quad (11)$$

3.4.3. Fusion and Classification

The node representations from both paths are aggregated into graph-level embeddings using a sum-pooling readout function:

$$\mathbf{z}_{fwd} = \sum_{v \in \mathcal{V}} \vec{\mathbf{h}}_v^{(L)}, \quad \mathbf{z}_{bwd} = \sum_{v \in \mathcal{V}} \overleftarrow{\mathbf{h}}_v^{(L)}. \quad (12)$$

We then fuse these complementary contexts and pass them through a Multi-Layer Perceptron (MLP) for class prediction:

$$\mathbf{z}_{final} = \text{ReLU}(\mathbf{W}_{fuse}[\mathbf{z}_{fwd} \parallel \mathbf{z}_{bwd}]), \quad (13)$$

where $\mathbf{W}_{fuse} \in \mathbb{R}^{d_{fuse} \times 2d_h^l}$ is the fusion matrix. Subsequently, we feed the fused representation $\mathbf{z}_{final}$ into an MLP to produce class logits, followed by a softmax layer to obtain the final class probabilities:

$$\hat{\mathbf{y}} = \text{Softmax}(\text{MLP}(\mathbf{z}_{final})). \quad (14)$$

This dual-path design allows the model to capture both the sequence of actions leading to a state and the consequences that follow it. As a result, it provides a more complete interaction context for fine-grained classification. For clarity, the end-to-end execution flow is summarized in Algorithm 1. The pseudo-code shows feature initialization, the bi-directional message-passing process across graph layers, and the final fusion stage used to derive classification probabilities.

Algorithm 1 Forward procedure of the BiDT framework.

Input: Encrypted flow $\mathcal{F}_{meta} = \{(s_i, d_i, t_i)\}_{i=1}^N$; quantization scale s ; number of layers L

Output: Predicted class probabilities $\hat{\mathbf{y}}$

- 1: Construct the DTIG $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{X}, \mathbf{E}_{attr})$ from $\mathcal{F}_{meta}$
 - 2: Initialize node features by $\mathbf{h}_i^{(0)} \leftarrow \text{Embedding}(\min(s_i, S_{max}))$ for each $v_i \in \mathcal{V}$
 - 3: Construct the transposed edge set $\mathcal{E}^T = \{(v, u) \mid (u, v) \in \mathcal{E}\}$
 - 4: **for** $l = 0$ **to** $L - 1$ **do**
 - 5: **for** each node $i \in \mathcal{V}$ **do**
 - 6: Compute the forward aggregated message $\vec{\mathbf{m}}_{\mathcal{N}_{in}(i)}^{(l)}$
 - 7: Update the forward representation $\vec{\mathbf{h}}_i^{(l+1)}$
 - 8: Compute the backward aggregated message $\overleftarrow{\mathbf{m}}_{\mathcal{N}_{out}(i)}^{(l)}$
 - 9: Update the backward representation $\overleftarrow{\mathbf{h}}_i^{(l+1)}$
 - 10: **end for**
 - 11: **end for**
 - 12: Read out graph-level representations: $\mathbf{z}_{fwd} \leftarrow \sum_{v \in \mathcal{V}} \vec{\mathbf{h}}_v^{(L)}$, $\mathbf{z}_{bwd} \leftarrow \sum_{v \in \mathcal{V}} \overleftarrow{\mathbf{h}}_v^{(L)}$
 - 13: Fuse both views by $\mathbf{z}_{final} \leftarrow \text{ReLU}(\mathbf{W}_{fuse}[\mathbf{z}_{fwd} \parallel \mathbf{z}_{bwd}])$
 - 14: Obtain the prediction $\hat{\mathbf{y}} \leftarrow \text{Softmax}(\text{MLP}(\mathbf{z}_{final}))$
 - 15: **return** $\hat{\mathbf{y}}$
-

4. Experiments and Results

4.1. Experimental Setup

We conducted experiments to evaluate the effectiveness and generalizability of the BiDT. The evaluation included the primary VNAT benchmark and a validation experiment on ISCX-VPN.

4.1.1. Datasets and Preprocessing

To evaluate the BiDT, we used the VNAT dataset as the primary benchmark and ISCX-VPN as a validation dataset.

Dataset 1: Fine-Grained VNAT: We selected the VNAT [37] dataset as the primary benchmark for fine-grained classification. This dataset contains several common types of encrypted traffic collected in realistic network environments. To focus on the most challenging fine-grained setting, we selected 10 representative applications: Netflix, Vimeo, YouTube (Video Streaming); Skype-Chat, Voice over Internet Protocol (VoIP) (Communication); Secure Shell (SSH), Remote Desktop Protocol (RDP) (Remote Access); and Secure Copy Protocol (SCP), SSH File Transfer Protocol (SFTP), Rsync (File Transfer).

Dataset 2: ISCX-VPN: To provide additional cross-benchmark validation, we used the ISCX-VPN dataset [38]. This dataset contains multiple application types in both virtual private network (VPN) and non-VPN settings. Following the evaluation protocol in [9], we excluded browser-generated traffic and VPN-tunneled traffic and retained only the non-VPN subset. To ensure a fair cross-dataset comparison with Dataset 1, we re-extracted ISCX-VPN flows using the same Scapy-based preprocessing strategy, including TCP/UDP-only filtering; removal of Domain Name System (DNS) and service-discovery traffic (ports 53, 5353, and 5355), removal of local broadcast and multicast traffic; and discarding of flows with fewer than five packets. After preprocessing, the ISCX-VPN subset was reorganized into a 17-class fine-grained benchmark containing 5369 valid flows.

Preprocessing and Filtering: For both datasets, we applied a unified preprocessing pipeline. Unlike [9], which uses NFStream [39], we employed Scapy [40] for flow grouping and feature extraction. We chose Scapy because it supports flexible and precise custom header extraction. This allows us to implement strict filtering logic and process large PCAP files iteratively while avoiding the memory instability that may arise in stream-based parsers on large datasets. During extraction, we applied a set of heuristic rules through Scapy to remove common background noise and retained only traffic relevant to end-host interactions. Specifically, we filtered out non-IP/layer-2 (L2) management traffic (e.g., address resolution protocol (ARP)), DNS queries (ports 53 and 5353), local broadcast/multicast messages (e.g., Simple Service Discovery Protocol (SSDP), 239.255.255.250) and retained only TCP and UDP flows. We also discarded flows with fewer than five packets because they provide insufficient information for graph construction. Flows with fewer than five packets typically represent failed TCP handshakes, background keep-alive probes, or incomplete sessions that contain no actual application-level payload data. In our preprocessing, this rule filtered out approximately 5.39% of the raw flows in VNAT and 6.71% in ISCX-VPN. Removing these non-interactive stubs prevents the benchmark from being biased by trivial background noise and ensures that the evaluation rigorously focuses on the actual request–response interactions of the services. Table 1 summarizes the number of classes and valid flow samples obtained from each dataset after structural cleaning.

Table 1. Dataset statistics after preprocessing.

Dataset ID	Dataset	Label Number	Sample Number
1	VNAT	10	3549
2	ISCX-VPN	17	5369

4.1.2. Evaluation Metrics

We used the same evaluation metrics for the BiDT and all baselines. For multi-class classification, metrics were computed per class in a one-vs-rest manner, where true positives (*TP*) denote the number of correctly predicted positive instances, true negatives (*TN*) the number of correctly predicted negative instances, false positives (*FP*) the number of negative instances incorrectly predicted as positive, and false negatives (*FN*) the number of positive instances incorrectly predicted as negative.

Accuracy (*ACC*) is defined as

$$ACC = \frac{TP + TN}{TP + TN + FP + FN}. \quad (15)$$

For each class, we also computed recall (*RC*), precision (*PR*), and *F1 – score*:

$$RC = \frac{TP}{TP + FN}, \quad (16)$$

$$PR = \frac{TP}{TP + FP'} \quad (17)$$

$$F1 = \frac{2 \times PR \times RC}{PR + RC}. \quad (18)$$

Due to class imbalance (e.g., frequent YouTube versus rare VoIP flows in VNAT), we prioritized *Macro-F1*, which is computed by equally averaging the class-wise *F1*-scores obtained with the one-vs-rest setting. We used it as the primary indicator of overall fine-grained classification stability.

We also used the False Positive Rate (*FPR*) to monitor whether the model produces excessive spurious positive predictions. *FPR* is computed per class as

$$FPR = \frac{FP}{FP + TN}. \quad (19)$$

A low *Macro-FPR* together with a high *Macro-F1* indicates that the classifier maintains both recall and specificity, which is especially important in imbalanced fine-grained traffic settings.

4.1.3. Comparison Baselines

To evaluate the proposed framework fairly, we compared it against several state-of-the-art baselines from different methodological paradigms. All these methods rely strictly on traffic metadata (e.g., packet lengths, directions, or timestamps) rather than payload inspection. The baselines are as follows:

- IBGC [9] builds an interactive behavior graph for communication behaviors and adopts a GraphSAGE-based subgraph sampling model to realize high-precision encrypted traffic classification with outstanding performance on similar traffic distinction.
- GRAIN [41] leverages seven statistical features based on packet payload length and adopts a classifier chain with two cascaded RF classifiers to realize granular multi-label classification of encrypted traffic at application name and inter- and intra-application service levels.
- GraphDApp [5] models encrypted decentralized application (DApp) traffic with packet length and direction metadata as TIGs and adopts a GNN-based classifier with multilayer perceptrons (MLPs) for accurate DApp encrypted traffic classification in closed and open-world scenarios.
- SmartDetector [7] extracts packet length, direction and IAT to build the Semantic Attribute Matrix, adopts traffic-specific data augmentation, pre-trains an encoder via contrastive learning on unlabeled data, and fine-tunes with few labeled samples to robustly detect obfuscated malicious encrypted traffic.
- ANASETC [42] integrates traffic burst features with Neural Architecture Search, designs the ETNasnet search space with parameter sharing, extracts burst features via bidirectional gated recurrent units (bi-GRU) encoding, and adopts a reinforcement learning strategy to automatically generate high-performance architectures for accurate encrypted traffic classification.

4.1.4. Implementation Details

All experiments were conducted using PyTorch Geometric 2.6.1 on an NVIDIA GeForce RTX 4060 graphics processing unit (GPU). We employed a Stratified Block Split strategy to divide the dataset into 70% for training, 15% for validation, and 15% for testing, which effectively prevents data leakage between contiguous flows.

For our proposed BiDT framework, the model was trained using the Adam optimizer with an initial learning rate of 1×10^{-3} for a maximum of 100 epochs. To mitigate over-

fitting, we applied an early stopping mechanism with a patience of 10 epochs. The initial hyperparameter settings were drawn from related prior studies, and the final optimal configuration was determined through grid search, as detailed in Table 2.

To ensure a strictly fair and rigorous comparison, all baseline models were evaluated under identical experimental conditions. Specifically, every baseline was trained and evaluated on the exact same VNAT data splits (70-15-15) for a maximum of 100 epochs, utilizing the same early stopping strategy (patience of 10 epochs) as the BiDT. Furthermore, we did not merely execute the baseline methods with their default parameters. Instead, their initial hyperparameters were set according to the recommendations in their respective original papers. Subsequently, we performed grid-search tuning on the validation set to determine its final hyperparameters, ensuring that every comparative method could achieve its best possible performance on the VNAT dataset.

Table 2. Hyperparameter configuration.

Hyperparameters	Range	Value
Flow Length (N)	[10 ... 80]	40
IAT Quantization Scale (s)	[1, 10 ... 10^6]	10^4
GNN Layers	[1, 2, 3, 4]	3
Hidden Dimension	[32, 64, 128, 256]	64
Dropout	[0.025 ... 0.5]	0.05
Batch Size	[16, 32, 64, 128]	128
Learning Rate	[0.0001 ... 0.01]	0.001
Training Epochs	[50 ... 200]	100
Optimizer	[Adam, SGD]	Adam
Activation Function	[Tanh, ReLU]	ReLU

4.2. Hyperparameter Sensitivity and Implementation

4.2.1. Impact of Flow Sequence Length

Since our BiDT framework relies on capturing the rhythm of interactions, the number of packets N used to construct the graph is a critical hyperparameter. Insufficient packets may fail to encompass the initial negotiation and characteristic burst transitions (e.g., request–response exchanges and early data bursts), while excessive packets introduce noise and computational overhead.

We evaluated the model on the VNAT dataset by varying $N \in \{10, 20, 30, 40, 50, 60, 70, 80\}$, fixing all other hyperparameters (GraphSAGE-mean, $s = 10^4$, random seed 42). The results are illustrated in Figure 4. Overall accuracy rises steeply from 85.71% at $N = 10$ to 96.61% at $N = 30$ then plateaus at ~ 98.6 – 98.9% for $N \geq 40$. A similar trend is observed for per-class recall and precision, and the Macro-FPR drops sharply from 1.72% ($N = 10$) to below 0.20% ($N \geq 40$). Notably, protocol-level classes sensitive to control-phase timing—namely Rsync, SCP, and SFTP—require at least $N = 30$ packets to surpass zero recall, while session-oriented classes such as Skype-Chat and SSH saturate as early as $N = 20$. Beyond $N = 40$, performance gains are marginal, but memory consumption grows linearly. Thus, $N = 40$ provides the optimal trade-off, effectively covering the protocol negotiation and the first major data exchange phase.

4.2.2. Sensitivity to IAT Quantization Granularity

The temporal edge feature $\mathbf{e}_{ij} = [\mathbf{e}_{ij}^{log}, \mathbf{e}_{ij}^{zero}]$ is governed by the quantization scale s , which maps raw IATs (in seconds) to integer bins. We evaluated seven scales: $s \in \{10^6, 10^5, 10^4, 10^3, 10^2, 10, 1\}$, corresponding to 1 μ s, 10 μ s, 100 μ s, 1 ms, 10 ms, 100 ms, and 1 s granularity, respectively.

As shown in Figure 5, while overall performance metrics largely stay within a high range, the per-class results expose a clear dichotomy in sensitivity to quantization precision. Notably, file-transfer protocols (e.g., SCP, SFTP, Rsync) and remote shells (SSH) remain robustly stable near 100% F1-score across all scales, indicating their structural and size-based features inherently dominate the classification.

Conversely, adaptive streaming behaviors (particularly Netflix) and finely paced interactive/signaling flows (e.g., VoIP and RDP) exhibit dramatic sensitivity to IAT scaling. This sensitivity is driven by two main effects:

(1) Over-fragmentation at fine scales: Excessively fine resolutions, particularly at $10\ \mu\text{s}$ ($s = 10^5$), excessively split normal hardware jitter into distinct discrete bins. This shatters the consistency of high-speed intra-burst arrivals, causing performance drops in timing-sensitive classes like Netflix ($F1$ drops to 82.0%) and VoIP (66.7%).

(2) Temporal smoothing at coarse scales: Conversely, coarse granularities ($\geq 1\ \text{ms}$) amalgamate distinctly paced inter-burst intervals. Smoothing out these critical sub-millisecond footprint differences leads to severe misclassification in interactive pacing, heavily degrading VoIP and causing RDP to plummet (e.g., $F1$ drops to 50.0% at 100 ms).

Balancing these factors, we identify parameter $s = 10^4$ (100 μs) as the optimal quantization set point. This choice attains the peak overall performance, suggesting a practical temporal boundary at which low-level system noise is filtered out while the macro-rhythm of complex streaming remains preserved.

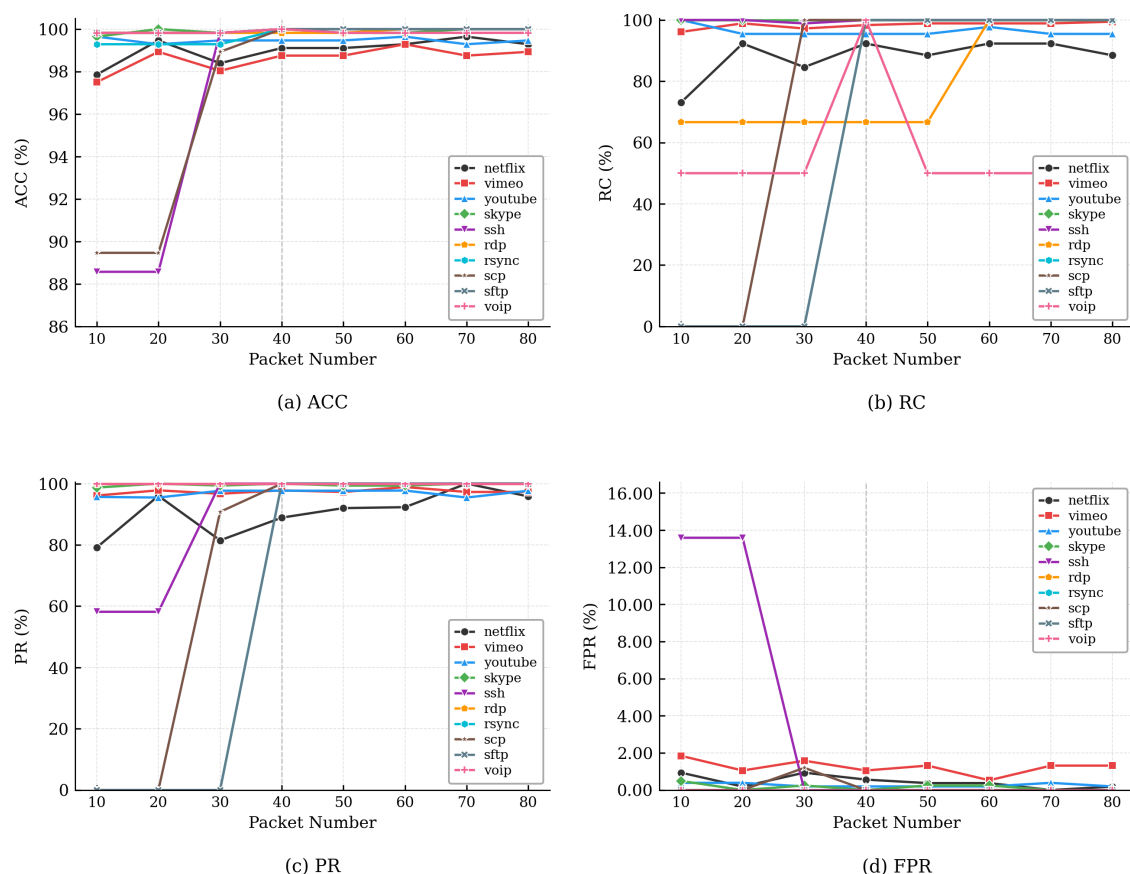


Figure 4. Per-class sensitivity of the BiDT framework to flow sequence length $N \in \{10, 20, \dots, 80\}$, with all other hyperparameters fixed. Each panel shows one metric for all 10 traffic classes: (a) ACC (%), (b) RC (%), (c) PR (%), and (d) FPR (%). The dashed vertical line marks the adopted default $N = 40$, which achieves the best accuracy–efficiency trade-off. Protocol-level classes (Rsync, SCP, and SFTP) require $N \geq 30$ for non-zero recall, while session-level classes (Skype-Chat and SSH) saturate by $N = 20$.

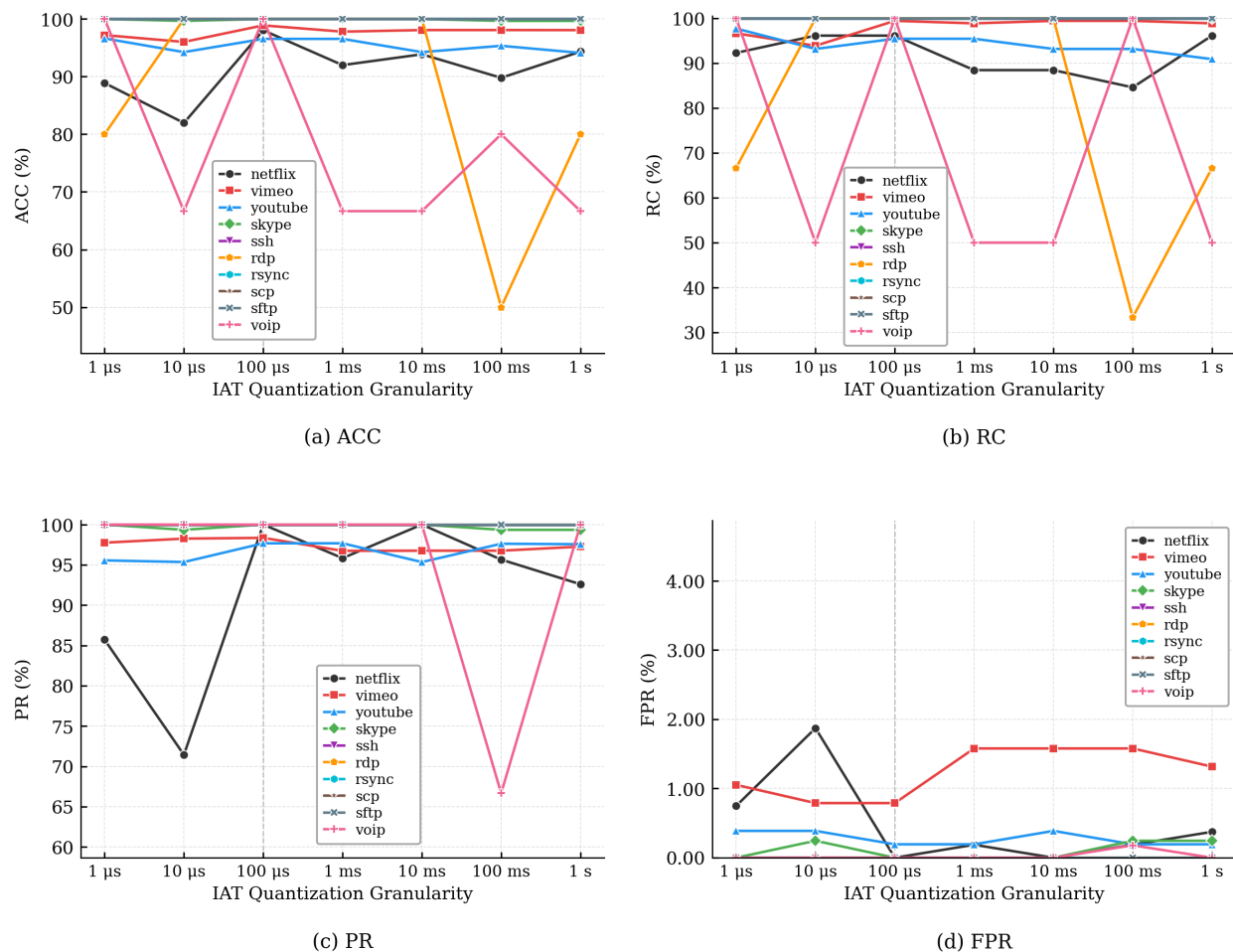


Figure 5. Per-class sensitivity of the BiDT framework to IAT quantization granularity across seven scales (1 μ s–1 s). Each panel shows one metric for all 10 traffic classes: (a) ACC (%), (b) RC (%), (c) PR (%), and (d) FPR (%). The dashed vertical line marks the adopted default $s = 10^4$ (100 μ s granularity), which achieves the best overall performance among all evaluated scales. ACC, PR, and RC remain broadly stable; FPR stays consistently low, with a slight rise at 10 μ s ($s = 10^5$) where excessive resolution fragments IAT bins.

4.3. Main Results

4.3.1. Performance on VNAT Dataset

Table 3 reports the main results on the challenging VNAT fine-grained benchmark across eight metrics. To ensure fair comparison, we reproduced all baseline methods and evaluated them on the same VNAT splits. The results show that the BiDT performs better than the compared baselines overall. Specifically, the BiDT achieves the highest accuracy of 98.57%, precision of 99.00%, and Macro-F1-score of 95.10%, together with the lowest FPR of 0.1942%. These results suggest that the model maintains high specificity while also separating highly similar application traffic effectively.

Table 3. Performance comparison on the VNAT 10-class dataset.

Method	ACC (%)	PR (%)	RC (%)	F1 (%)	FPR (%)	AUC	Train (s)	Infer (ms)
GraphDApp ([5])	83.75	54.09	42.59	42.09	2.2536	0.8807	41.0	130.6
GRAIN ([41])	94.29	84.65	77.06	79.80	0.7753	0.9930	1.1	7.7
IBGC ([9])	94.46	89.89	86.48	86.81	0.7602	0.9606	128.4	198.6
SmartDetector ([7])	97.32	96.43	92.91	93.77	0.3126	0.9977	264.9	860.2
ANASETC ([42])	98.04	96.95	93.69	94.37	0.2214	0.9995	61.5	134.7
BiDT (Ours)	98.57	99.00	92.95	95.10	0.1942	0.9984	10.8	8.3

Traditional machine-learning architectures (GRAIN [41]) and standard GNN models (GraphDApp [5] and IBGC [9]) remain limited on this fine-grained task, with accuracies ranging from 83.75% to 94.46%. In particular, IBGC [9] emphasizes undirected correlations and lacks explicit temporal edge modeling, which may explain its 86.81% F1-score on this benchmark. Our approach also remains competitive against recent temporal and sequence-based baselines. SmartDetector [7], a 2D-CNN-based method, reaches 97.32% accuracy, but its F1-score on minority classes is lower (93.77%), and its inference cost is much higher (860.2 ms). ANASETC [42] reaches a close 98.04% accuracy and is slightly better in recall (93.69% vs. BiDT's 92.95%) and area under the curve (AUC) (0.9995 vs. 0.9984). However, the BiDT still provides stronger F1 stability and a lower FPR.

To ensure the robustness of our results and confirm that the improvements were not due to random training variance, we evaluated the models across multiple independent runs with different random seeds. The BiDT consistently achieved an accuracy of $98.57\% \pm 0.14\%$ and a Macro-F1 of $95.10\% \pm 0.18\%$. This tight variance confirms that its performance advantage over the closest baseline, ANASETC (which achieved $98.04\% \pm 0.21\%$ accuracy), is statistically stable and significant.

Beyond predictive performance, the BiDT also shows favorable computational efficiency. Because early stopping is used during training, the reported Training Time mainly reflects the time required to reach convergence and should be treated as a secondary reference. In deterministic inference, the BiDT requires only 8.3 ms. This latency is roughly two orders of magnitude lower than that of more complex deep learning baselines, such as SmartDetector and ANASETC, and remains close to the lightweight statistical baseline GRAIN (7.7 ms). According to the categorization of real-time analysis capability in [43], this level of inference latency already satisfies the requirements of practical real-time deployment. These results indicate that embedding temporal dynamics directly into graph topology can provide an efficient alternative to heavier recurrent or sequence-based feature extractors.

4.3.2. Fine-Grained Analysis on VNAT

Figure 6 illustrates the confusion matrix. One notable result of our method is the perfect separation (100% F1) of protocol pairs that are traditionally confused.

Our model clearly distinguishes between SCP, SFTP, and Rsync (100% F1). Although these protocols serve similar purposes and all operate over encrypted channels, the temporal edge features appear to capture differences in their transmission rhythms and control signaling latencies.

As expected, distinguishing between streaming services remains the most challenging fine-grained task because of their near-identical hypertext transfer protocol (HTTP)-based adaptive bitrate streaming mechanisms. Even so, the model still achieves strong separation within this group, with the precision of Netflix, Vimeo, and YouTube reaching the 95–98% range. It also separates this group clearly from other traffic types and preserves stable category-level boundaries.

4.4. Ablation Study

To quantify the contribution of each architectural component, we conducted a component-wise ablation study by removing one design element at a time while keeping the others fixed. We evaluated four ablated variants on the VNAT 10-class benchmark: (V1) w/o Discrete Embed replaces the learnable packet-size embedding with a 5-dimensional raw feature vector; (V2) w/o Temporal Edge zeros all IAT edge attributes while preserving the graph topology; (V3) w/o Backward Path disables the backward GNN stack, reducing the model to a forward-only GraphSAGE; and (V4) w/o Directed

Edges symmetrizes the edge set by adding the reverse of every directed edge, yielding an undirected graph. The results are reported in Table 4.

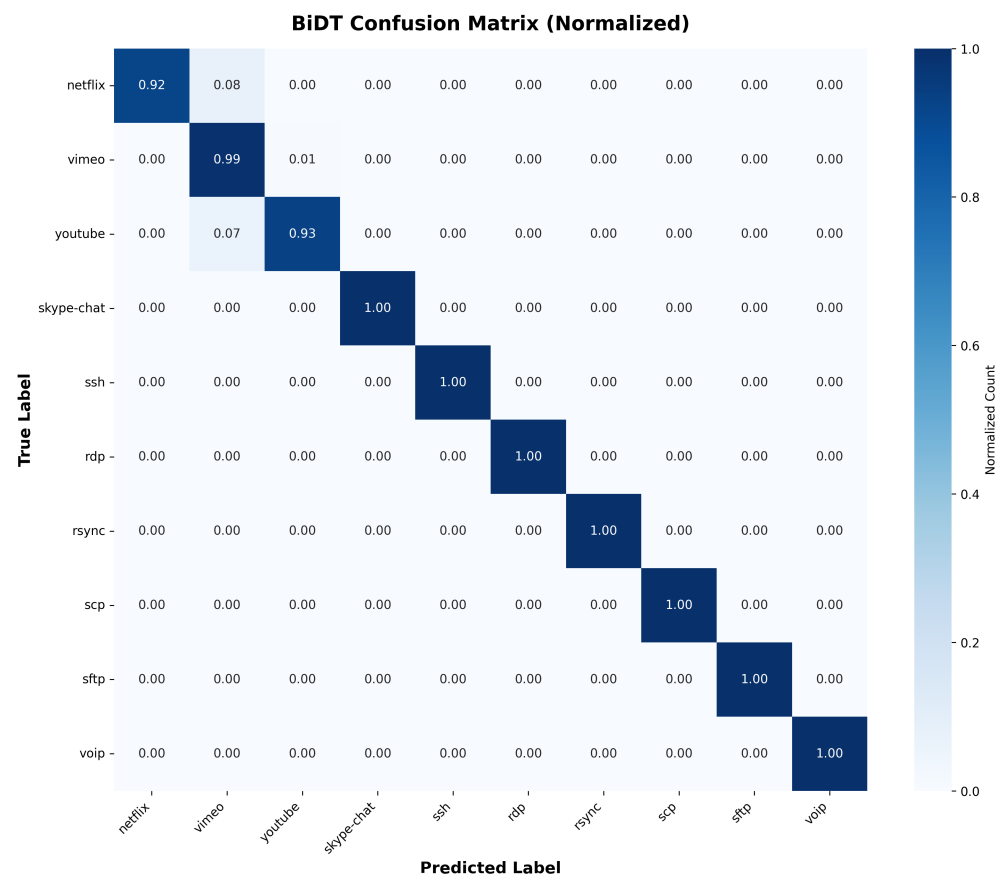


Figure 6. The confusion matrix of the BiDT on the VNAT 10-class dataset, demonstrating perfect separation for most protocol classes and strong isolation even among inherently similar streaming services.

Table 4. Ablation study on the VNAT 10-class dataset.

Model Variant	ACC (%)	Macro F1 (%)	Δ ACC	Δ F1
Full Model (BiDT)	98.57	95.10	—	—
V1: <i>w/o</i> Discrete Embed	91.79	70.40	−6.78	−24.70
V2: <i>w/o</i> Temporal Edge	95.21	91.12	−3.36	−3.98
V3: <i>w/o</i> Backward Path	93.21	88.89	−5.36	−6.21
V4: <i>w/o</i> Directed Edges	96.39	93.14	−2.18	−1.96

- Effect of Discrete Packet-Size Embedding (V1): Removing the learnable size embedding yields the largest performance drop: −6.78% accuracy and −24.70% *Macro-F1*. The degradation is particularly severe for protocol-specific classes—Rsync *F1* falls from 100% to 40%, and SFTP collapses entirely to 0%—because these protocols are chiefly distinguished by their characteristic packet-length distributions. This confirms that the discrete embedding is the single most critical component for fine-grained file-transfer protocol separation.
- Effect of Temporal Edge Features (V2): Zeroing the IAT attributes causes a −3.98% *Macro-F1* drop and a −3.36% drop in accuracy. Although broad class boundaries are maintained, temporal information provides the additional granularity needed to separate protocol variants with similar transfer volumes but different pacing behaviors (e.g., Netflix vs. VoIP).

- Effect of Backward Aggregation Path (V3): Disabling the backward GNN stack yields a -6.21% Macro-F1 degradation and a -5.36% drop in accuracy. This consistent degradation confirms that retrospective flow context—response-acknowledgment patterns captured in the reverse direction—provides measurable benefit.
- Effect of Directed Edges (V4): Symmetrizing the graph (undirected) reduces overall accuracy by -2.18% and Macro-F1 by -1.96% . Edge directionality primarily benefits classes whose control-phase messages (client $\rightarrow$ server vs. server $\rightarrow$ client) have asymmetric temporal patterns; without direction, the model cannot disambiguate initiator from responder roles in the early handshake.

Overall, the discrete packet-size embedding is the strongest contributor to fine-grained protocol discrimination, followed by temporal edge features and edge directionality, while the backward path provides additional complementary benefit. The results indicate that all three metadata components (packet size, IAT, and direction) are necessary for strong 10-class encrypted traffic classification.

4.5. Generalizability Validation

To examine the generalizability of the BiDT, we conducted an additional validation experiment on the non-VPN subset of ISCX-VPN. This setting allows us to test the model under different network environments and collection conditions.

The BiDT achieves strong performance on this validation dataset. The overall classification results are summarized in Table 5. The model shows strong classification ability across different non-VPN application types.

Table 5. Generalizability validation on the ISCX-VPN dataset.

Dataset	ACC (%)	PR (%)	RC (%)	Macro F1 (%)
ISCX-VPN	95.23	93.24	92.38	91.67

Classification performance for complex categories such as Video Streaming and File Transfer remains stable. Although some individual application types show minor fluctuations, the overall accuracy stays at a high level. This suggests that the BiDT maintains stable classification performance in different dataset and network environments.

Overall, the BiDT maintains high multi-class classification accuracy on both the primary benchmark and the validation experiment. This consistency suggests that the proposed design remains effective across benchmarks for encrypted traffic classification.

5. Discussion

Although the BiDT shows strong performance in fine-grained encrypted traffic classification, several aspects merit further discussion in order to clarify its practical implications and current limitations.

The Role of Interaction Rhythm: One clear observation from our results is that topology alone is often insufficient to distinguish functionally similar encrypted protocols, such as SCP and SFTP, or closely related adaptive streaming behaviors. By embedding quantized IATs into directed edges, the BiDT moves the representation from static communication structure toward interaction rhythm. Our sensitivity analysis further suggests that $100\ \mu\text{s}$ quantization captures this rhythm effectively. It filters out operating system jitter while preserving the millisecond-level request-response pacing associated with application behavior.

Real-World Deployment Feasibility and Scalability: Practical traffic classification systems require low latency, strong privacy preservation, and the ability to scale to real-world network volumes. The BiDT addresses these requirements effectively. First, it relies

only on three payload-agnostic metadata attributes: size, direction, and time, avoiding CPU-intensive DPI. Second, regarding scalability to larger traffic volumes, the framework's computational complexity per flow is strictly bounded. Because the DTIG targets the first $N = 40$ packets of a flow, the graph size and inference memory cost remain constant ($O(1)$) regardless of whether the flow lasts for seconds or hours. This early-classification capability prevents the buffering of long sessions, yielding a deterministic inference time of roughly 8.3 ms per flow. Such lightweight characteristics suggest that the DTIG and BiGraphSAGE are well-suited for scaling to high-throughput environments or inline QoS routing.

Limitations and Adversarial Vulnerabilities: The model's reliance on temporal and packet-size metadata also introduces several limitations. First, as reflected in our initial observations with minority classes, this data-driven GNN approach can be sensitive to severe dataset imbalance and limited training samples without specialized loss re-weighting or minority oversampling strategies. Second, because the BiDT depends strongly on packet size patterns and transmission pacing, it may be vulnerable to deliberate obfuscation strategies, such as random packet padding or artificial delay injection. These operations could alter the metadata that the model uses most heavily. Finally, although the framework models standard 5-tuple internet protocol (IP) flows effectively, modern multiplexed protocols such as QUIC may carry multiple logical streams within a single UDP connection. Recovering clean interaction graphs from such multiplexed traffic without explicit logical demultiplexing remains a non-trivial problem.

6. Conclusions

In this paper, we present the BiDT, a framework for fine-grained encrypted traffic classification. To address the limitations of existing graph-based methods in modeling interaction directionality and temporal dynamics, we introduce the DTIG, which embeds IATs explicitly into directed edges. We also design a BiGraphSAGE model to learn these structural and temporal representations from both causal and retrospective views.

The experimental results show that the BiDT achieves an outstanding 98.57% accuracy on a challenging 10-class benchmark, successfully separating inherently similar protocols such as SCP and SFTP with near-perfect precision. Furthermore, by bounding the input sequence to the first $N = 40$ packets, the framework maintains an $O(1)$ computational complexity per flow. This early-classification strategy avoids the buffering of long sessions, yielding a deterministic and lightweight inference time of approximately 8.3 ms. These findings confirm that explicit modeling of interaction rhythm significantly reduces ambiguity between functionally similar applications while satisfying the low-latency requirements of real-world deployment.

In future work, we plan to address model sensitivity to severe class imbalance and investigate more resilient representations under deliberate adversarial perturbation conditions, such as random packet padding or delay injection. Additionally, we will explore advanced graph construction strategies for explicitly demultiplexing modern protocols such as QUIC.

Author Contributions: Conceptualization, J.Y. and H.S.; methodology, J.Y. and Z.D.; software, J.Y. and Y.H.; validation, Y.H.; writing—original draft preparation, J.Y.; writing—review and editing, H.S. and Y.H.; supervision, H.S. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by the National Natural Science Foundation of China (Grant Nos. 62572187 and 62472168). This research is also supported by the Fundamental Research Funds for the Central Universities.

Data Availability Statement: The original contributions presented in this study are included in the article. Further inquiries can be directed to the corresponding authors.

Acknowledgments: We acknowledge the use of Qwen 2.5 and Qwen 3 (<https://www.qianwen.com/chat/>, accessed from 20 February 2025 to 20 February 2026) to improve the organization and academic writing of this document. No portion of this work was produced exclusively by any AI tools.

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

References

1. Liu, C.; He, L.; Xiong, G.; Cao, Z.; Li, Z. Fs-net: A flow sequence network for encrypted traffic classification. In *Proceedings of the IEEE INFOCOM 2019—IEEE Conference On Computer Communications*; IEEE: New York, NY, USA, 2019; pp. 1171–1179.
2. Van Ede, T.; Bortolameotti, R.; Continella, A.; Ren, J.; Dubois, D.J.; Lindorfer, M.; Choffnes, D.; Van Steen, M.; Peter, A. Flowprint: Semi-supervised mobile-app fingerprinting on encrypted network traffic. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, San Diego, CA, USA, 23–26 February 2020; Volume 27.
3. Durumeric, Z.; Ma, Z.; Springall, D.; Barnes, R.; Sullivan, N.; Bursztein, E.; Bailey, M.D.; Halderman, J.A.; Paxson, V. The Security Impact of HTTPS Interception. In *Proceedings of the NDSS Symposium*, San Diego, CA, USA, 26 February–1 March 2017.
4. Anderson, B.; McGrew, D. Identifying encrypted malware traffic with contextual flow data. In *Proceedings of the 2016 ACM Workshop on Artificial Intelligence and Security*; ACM: New York, NY, USA, 2016; pp. 35–46.
5. Shen, M.; Zhang, J.; Zhu, L.; Xu, K.; Du, X. Accurate decentralized application identification via encrypted traffic analysis using graph neural networks. *IEEE Trans. Inf. Forensics Secur.* **2021**, *16*, 2367–2380. [[CrossRef](#)]
6. Rezaei, S.; Liu, X. How to achieve high classification accuracy with just a few labels: A semi-supervised approach using sampled packets. *arXiv* **2018**, arXiv:1812.09761.
7. Shen, M.; Wu, J.; Ye, K.; Xu, K.; Xiong, G.; Zhu, L. Robust detection of malicious encrypted traffic via contrastive learning. *IEEE Trans. Inf. Forensics Secur.* **2025**, *20*, 4228–4242. [[CrossRef](#)]
8. Li, W.; Zhang, X.Y.; Bao, H.; Shi, H.; Wang, Q. ProGraph: Robust network traffic identification with graph propagation. *IEEE/ACM Trans. Netw.* **2022**, *31*, 1385–1399. [[CrossRef](#)]
9. Li, Y.; Chen, X.; Tang, W.; Zhu, Y.; Han, Z.; Yue, Y. Interaction matters: Encrypted traffic classification via status-based interactive behavior graph. *Appl. Soft Comput.* **2024**, *155*, 111423. [[CrossRef](#)]
10. Pham, T.D.; Ho, T.L.; Truong-Huu, T.; Cao, T.D.; Truong, H.L. Mappgraph: Mobile-app classification on encrypted network traffic using deep graph convolution neural networks. In *Proceedings of the 37th Annual Computer Security Applications Conference*; ACM: New York, NY, USA, 2021; pp. 1025–1038.
11. Taylor, V.F.; Spolaor, R.; Conti, M.; Martinovic, I. Appscanner: Automatic fingerprinting of smartphone apps from encrypted network traffic. In *Proceedings of the 2016 IEEE European Symposium on Security and Privacy (EuroS&P)*; IEEE: New York, NY, USA, 2016; pp. 439–454.
12. Novo, C.; Morla, R. Flow-based detection and proxy-based evasion of encrypted malware C2 traffic. In *Proceedings of the 13th ACM Workshop on Artificial Intelligence and Security*; ACM: New York, NY, USA, 2020; pp. 83–91.
13. Diao, Z.; Qiao, M.; Wang, X.; Zhang, G.; Liang, W.; Chen, J.; Pei, C.; Li, Y.; Li, Z.; Xie, G. Not All Data are What You Need: A Data-Efficient Training Method Using Heterogeneous Hardware. *IEEE Trans. Knowl. Data Eng.* **2026**, 1–14. [[CrossRef](#)]
14. Lin, X.; Xiong, G.; Gou, G.; Li, Z.; Shi, J.; Yu, J. Et-bert: A contextualized datagram representation with pre-training transformers for encrypted traffic classification. In *Proceedings of the ACM Web Conference 2022*; ACM: New York, NY, USA, 2022; pp. 633–642.
15. Shapira, T.; Shavitt, Y. FlowPic: A generic representation for encrypted traffic classification and applications identification. *IEEE Trans. Netw. Serv. Manag.* **2021**, *18*, 1218–1232. [[CrossRef](#)]
16. Jin, Z.; Duan, K.; Chen, C.; He, M.; Jiang, S.; Xue, H. FedETC: Encrypted traffic classification based on federated learning. *Heliyon* **2024**, *10*, e35962. [[CrossRef](#)]
17. Liang, C.; Diao, Z.; Wang, X.; Huo, Y.; Li, K.; He, D.; Liang, W. FedAHP: Federated Learning with Adaptive Hot Parameter Identification and Personalized Anchoring for multi-agent collaboration. *J. Ind. Inf. Integr.* **2026**, *51*, 101087. [[CrossRef](#)]
18. 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* **2020**, *1*, 57–81. [[CrossRef](#)]
19. Wu, Z.; Pan, S.; Chen, F.; Long, G.; Zhang, C.; Yu, P.S. A comprehensive survey on graph neural networks. *IEEE Trans. Neural Netw. Learn. Syst.* **2020**, *32*, 4–24. [[CrossRef](#)]
20. Wang, S.; Wang, Z.; Zhou, T.; Sun, H.; Yin, X.; Han, D.; Zhang, H.; Shi, X.; Yang, J. Threatrace: Detecting and tracing host-based threats in node level through provenance graph learning. *IEEE Trans. Inf. Forensics Secur.* **2022**, *17*, 3972–3987. [[CrossRef](#)]
21. He, H.Y.; Yang, Z.G.; Chen, X.N. PERT: Payload encoding representation from transformer for encrypted traffic classification. In *Proceedings of the 2020 ITU Kaleidoscope: Industry-Driven Digital Transformation (ITU K)*; IEEE: New York, NY, USA, 2020; pp. 1–8.
22. Doriguzzi-Corin, R.; Millar, S.; Scott-Hayward, S.; Martinez-del Rincon, J.; Siracusa, D. LUCID: A practical, lightweight deep learning solution for DDoS attack detection. *IEEE Trans. Netw. Serv. Manag.* **2020**, *17*, 876–889. [[CrossRef](#)]

23. Finsterbusch, M.; Richter, C.; Rocha, E.; Muller, J.A.; Hanssgen, K. A survey of payload-based traffic classification approaches. *IEEE Commun. Surv. Tutor.* **2013**, *16*, 1135–1156. [\[CrossRef\]](#)
24. Lotfollahi, M.; Jafari Siavoshani, M.; Shirali Hossein Zade, R.; Saberian, M. Deep packet: A novel approach for encrypted traffic classification using deep learning. *Soft Comput.* **2020**, *24*, 1999–2012. [\[CrossRef\]](#)
25. Panchenko, A.; Lanze, F.; Pennekamp, J.; Engel, T.; Zinnen, A.; Henze, M.; Wehrle, K. Website fingerprinting at internet scale. In *Proceedings of the NDSS*; Internet Society: Reston, VA, USA, 2016; Volume 1, p. 23477.
26. Xu, S.J.; Geng, G.G.; Jin, X.B.; Liu, D.J.; Weng, J. Seeing traffic paths: Encrypted traffic classification with path signature features. *IEEE Trans. Inf. Forensics Secur.* **2022**, *17*, 2166–2181. [\[CrossRef\]](#)
27. Wang, W.; Zhu, M.; Wang, J.; Zeng, X.; Yang, Z. End-to-end encrypted traffic classification with one-dimensional convolution neural networks. In *Proceedings of the 2017 IEEE International Conference on Intelligence and Security Informatics (ISI)*; IEEE: New York, NY, USA, 2017; pp. 43–48.
28. Sirinam, P.; Imani, M.; Juarez, M.; Wright, M. Deep fingerprinting: Undermining website fingerprinting defenses with deep learning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*; ACM: New York, NY, USA, 2018; pp. 1928–1943.
29. Zhao, R.; Zhan, M.; Deng, X.; Li, F.; Wang, Y.; Wang, Y.; Gui, G.; Xue, Z. A novel self-supervised framework based on masked autoencoder for traffic classification. *IEEE/ACM Trans. Netw.* **2024**, *32*, 2012–2025. [\[CrossRef\]](#)
30. Wang, B.; Wang, B.; Wei, Z.; Zhao, S.; Chen, S.; Li, Z.; Wang, M. MFSI: Multi-flow based service identification for encrypted network traffic. *Comput. Netw.* **2025**, *265*, 111283. [\[CrossRef\]](#)
31. Zhang, H.; Yue, H.; Xiao, X.; Yu, L.; Li, Q.; Ling, Z.; Zhang, Y. Revolutionizing encrypted traffic classification with mh-net: A multi-view heterogeneous graph model. In *Proceedings of the AAAI Conference on Artificial Intelligence*; AAAI: Washington, DC, USA, 2025; Volume 39, pp. 1048–1056.
32. Zhang, H.; Yu, L.; Xiao, X.; Li, Q.; Mercaldo, F.; Luo, X.; Liu, Q. Tfe-gnn: A temporal fusion encoder using graph neural networks for fine-grained encrypted traffic classification. In *Proceedings of the ACM Web Conference 2023*; ACM: New York, NY, USA, 2023; pp. 2066–2075.
33. Zhao, J.; Cui, Z.; Fu, J.; Shen, M.; Li, Q. A Unified Framework for Robust Encrypted Malicious Traffic Detection in Adverse Environments via Graph Structure Learning. *IEEE Trans. Netw. Sci. Eng.* **2025**, *13*, 245–261. [\[CrossRef\]](#)
34. Huoh, T.L.; Luo, Y.; Li, P.; Zhang, T. Flow-based encrypted network traffic classification with graph neural networks. *IEEE Trans. Netw. Serv. Manag.* **2022**, *20*, 1224–1237. [\[CrossRef\]](#)
35. Han, X.; Xu, G.; Zhang, M.; Yang, Z.; Yu, Z.; Huang, W.; Meng, C. DE-GNN: Dual embedding with graph neural network for fine-grained encrypted traffic classification. *Comput. Netw.* **2024**, *245*, 110372. [\[CrossRef\]](#)
36. Shen, M.; Ye, K.; Liu, X.; Zhu, L.; Kang, J.; Yu, S.; Li, Q.; Xu, K. Machine learning-powered encrypted network traffic analysis: A comprehensive survey. *IEEE Commun. Surv. Tutor.* **2022**, *25*, 791–824. [\[CrossRef\]](#)
37. Jorgensen, S.; Holodnak, J.; Dempsey, J.; de Souza, K.; Raghunath, A.; Rivet, V.; DeMoes, N.; Alejos, A.; Wollaber, A. Extensible machine learning for encrypted network traffic application labeling via uncertainty quantification. *IEEE Trans. Artif. Intell.* **2023**, *5*, 420–433. [\[CrossRef\]](#)
38. Gil, G.D.; Lashkari, A.H.; Mamun, M.; Ghorbani, A.A. Characterization of encrypted and VPN traffic using time-related features. In *Proceedings of the 2nd International Conference on Information Systems Security and Privacy (ICISSP 2016)*; SciTePress: Setúbal, Portugal, 2016; pp. 407–414.
39. Aouini, Z.; Pekar, A. NFStream: A flexible network data analysis framework. *Comput. Netw.* **2022**, *204*, 108719. [\[CrossRef\]](#)
40. Biondi, P.; Lalet, P.; Potter, G.; Valadon, G.; Weiss, N. Scapy: The Python-Based Interactive Packet Manipulation Program & Library. 2026. Available online: <https://github.com/secdev/scapy> (accessed on 2 March 2026).
41. Zaki, F.; Afifi, F.; Abd Razak, S.; Gani, A.; Anuar, N.B. GRAIN: Granular multi-label encrypted traffic classification using classifier chain. *Comput. Netw.* **2022**, *213*, 109084. [\[CrossRef\]](#)
42. Zhang, H.; Chen, Z.; Xia, W.; Xiong, G.; Gou, G.; Li, Z.; Huang, G.; Li, Y. ANASETC: Automatic Neural Architecture Search for Encrypted Traffic Classification. In *Proceedings of the ICASSP 2025—2025 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*; IEEE: New York, NY, USA, 2025; pp. 1–5.
43. Feng, Y.; Li, J.; Mirkovic, J.; Wu, C.; Wang, C.; Ren, H.; Xu, J.; Liu, Y. Unmasking the internet: A survey of fine-grained network traffic analysis. *IEEE Commun. Surv. Tutor.* **2025**, *27*, 3672–3709. [\[CrossRef\]](#)

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.