Tracegram: Framing Trace-Level Traffic Analysis with Temporally-Aware Multiple Instance Learning

Jian Qu[1, 2, †], Yuchen Zhang[1, 2, †], Jialong Zhang[1, 2], Jianfeng Li[1, 2], Xiaobo Ma[1, 2, 3, *]
[1]*MOE Key Laboratory for Intelligent Networks and Network Security, Xi'an Jiaotong University.*
[2]*Faculty of Electronic and Information Engineering, Xi'an Jiaotong University.*
[3]*Shaanxi Province Key Laboratory of Computer Network, Xi'an Jiaotong University.*

Abstract

Modern network behaviors span multiple flows and evolve over time, making temporal and co-occurrence contexts across flows essential for reliable traffic analysis. This need is especially critical in the security domain, where attacks progress through reconnaissance, delivery, command and control, and lateral movement over extended intervals and across multiple flows. Existing packet-level or single-flow approaches fragment this context and limit performance on trace-level classification, detection, and attribution. We introduce the trace as the analysis unit and present `Tracegram`, which formulates trace-level analysis as Multiple Instance Learning. `Tracegram` combines per-flow encoders with a temporally aware aggregation module to reason across flows, preserve long-range dependencies, and produce key-flow attribution signals that support analyst verification and forensics. Our validation spans theory and practice. We theoretically justify the MIL-based decomposition for trace-level traffic analysis and conduct extensive experiments on four public datasets across multiple tasks, showing better or comparable performance to state-of-the-art methods. Finally, case studies on APT traces from the DAPT dataset show that `Tracegram` highlights flows aligned with attack phases, enabling targeted investigation. The source code and datasets for Tracegram are publicly available at https://doi.org/10.5281/zenodo.17978903.

1 Introduction

Network behavior typically spans multiple flows over extended intervals. Reliable analysis must model event ordering, inter-arrival timing, and cross-flow co-occurrence. This need is particularly critical for security applications [42, 43]. Advanced persistent threats (APTs) and botnets rarely reveal intent in isolated packets or a single flow [2, 50]; instead they manifest as carefully coordinated and often stealthy activity across heterogeneous flows over hours to days [9]. For example, a data-exfiltration trace may begin with a brief DNS query to establish infrastructure, continue with seemingly benign HTTPS beaconing for command and control (C2), and later include lateral movement or bulk transfer [45, 46].

Most existing systems are packet-or flow-centric. Packet-level methods such as deep packet inspection (DPI), signatures, and per-packet features are precise when payloads are visible, but their effectiveness degrades under encryption and protocol churn. Flow-level learning scales and works with encrypted traffic via header and timing features, yet it treats flows independently and fragments cross-flow temporal structure [41, 54]. Less work aggregates flows using long sequence models or graph-based correlation [12]. These attempts reduce fragmentation but often impose substantial computational cost, depend on custom trace-construction pipelines, and still provide limited analyst-oriented explanations [73].

These limitations motivate analyzing related flows jointly. We use the term *trace* to denote a semantically coherent collection of related flows that should be considered together. Working at this level introduces three practical challenges.

C1: Temporal dependencies across flows. Evidence is spread across flows over time. The model must retain event ordering, inter-arrival timing, and cross-flow co-occurrence to separate benign events from coordinated attacks, often under encryption using only header and timing features [14, 33, 44].

C2: Scalability and efficiency. Traces can span hours or days and contain a number of heterogeneous flows [32]. Merging all activity into a long sequence is costly and brittle [88]. Methods should keep long-range dependencies while keeping latency and memory within practical limits at the trace-level.

C3: Key-flow attribution. Beyond a trace label, operations teams need flow-level evidence to show which flows drove the decision and why [55, 78]. Attribution should be analyst-oriented and temporally grounded so that analysts can perform targeted triage and reproducible forensics [81–83], even when supervision is available only at the trace-level [49, 63].

We present `Tracegram`, which formulates trace-level traffic analysis as Multiple Instance Learning (MIL) [10, 11, 84].

*Corresponding author: xma.cs@xjtu.edu.cn
†Equal contribution.

In this formulation, a trace is treated as a bag and its flows as instances. `Tracegram` encodes each flow and then aggregates across flows to perform trace-level classification, and provide flow-level attribution as an inherent interpretability benefit identifying key flows. The design is encoder-agnostic and operates under encryption by relying on header-and timing-based features. This formulation is natural for three reasons. It matches the way supervision is typically available at the trace level, and it accommodates variable-sized collections [25, 90]. It concentrates cross-flow reasoning in an aggregation module rather than collapsing all activity into one long sequence, which improves efficiency and maintainability while preserving long-range dependencies [58]. It also supports instance-level attribution over flows, aligning with operational workflows and supporting analyst verification and forensics.

Conventional MIL treats instances as an unordered set and therefore discards temporal cues that are decisive in security traces. It does not capture event order, inter-arrival timing, or time gaps between flows [21]. To address this limitation, `Tracegram` introduces a temporally aware aggregation module. The aggregator conditions interactions between flows on their relative order and their time gaps, so the model can separate benign co-occurrence from DNS, followed by HTTPS for C2, and then lateral movement. It combines per-flow representations with lightweight timing signals and captures both local neighborhoods and long-range dependencies [64]. It also decouples flow encoding from temporal reasoning, which keeps computation practical at the trace-level and supports encoder replacement without redesign [8, 80]. Together, these choices yield a trace-level model that preserves temporal structure, scales to long horizons, and produces actionable attributions.

We substantiate `Tracegram` with a formal theoretical analysis and experiments. Theoretically, under an instance-separability assumption, we derive a two-stage objective that learns flow encoding followed by aggregation. Under an orthogonality condition this decomposed objective is equivalent to the optimal end-to-end objective, clarifying when the MIL decomposition is well-founded. Empirically, we evaluate on four public datasets and multiple tasks against strong recent baselines. Results are comparable to or better than state-of-the-art (SOTA), and APT trace analyses show the produced attributions are phase-aligned and support targeted investigation.

To the best of our knowledge, this work is among the first to systematically formulate trace-level traffic classification. In particular, we implement `Tracegram`, the first system that adopts trace-level supervision while producing flow-level explanations via weakly supervised attribution. The code is available at: https://github.com/YuchenZhang-Academic/Tracegram. Our work has made the following contributions:

- We present `Tracegram`, a trace-level MIL framework for traffic analysis, and design a temporally aware aggregation module that models order and inter-arrival timing across flows, overcoming the unordered instance limitation of conventional MIL and enabling key-flow attribution.
- We provide a theoretical justification for the MIL decomposition for trace-level traffic analytics by deriving a two-stage objective under an instance-separability assumption, and we then formally prove its equivalence to the optimal end-to-end objective under an orthogonality condition.
- We evaluate on four public datasets across multiple tasks, demonstrating performance comparable to or better than the SOTA. Analyses of APT traces from the DAPT dataset further demonstrate phase-aligned attributions that can effectively support targeted forensic investigation.

2 Problem Setting and Formulation

2.1 Motivation and Challenges

The trend of end-to-end encryption has rendered traditional deep packet inspection ineffective, leading to the rise of traffic fingerprinting [75]. This technique infers network activities by analyzing traffic data, proving its value in scenarios like website fingerprinting and IoT device identification [68].

The core task is to model a behavioral trace to accurately classify the network activity. As shown in Figure 1, modern attacks are inherently multi-stage, where a complete trace might mix short-lived reconnaissance flows, low-and-slow encrypted C2 tunnels, and bursts of data exfiltration streams with benign traffic [60]. Consequently, any method that analyzes single flows in isolation is prone to failure, as the full attack narrative is only visible when analyzing the trace holistically.

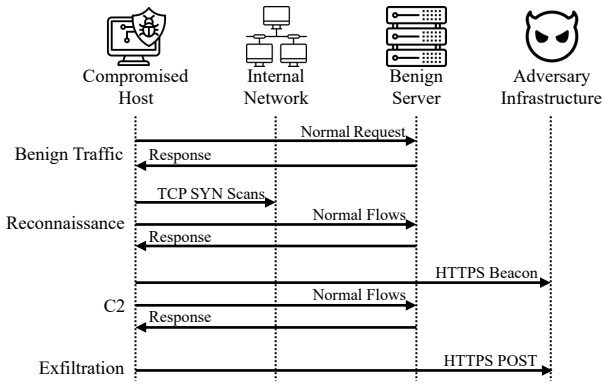


Figure 1: A trace example of a multi-stage attack.

However, a fatal contradiction emerges when applying SOTA solutions, such as large Transformer models, to this problem. These models are designed for single, linear sequences, not for multi-flow traces. Common workarounds, such as classifying flows individually (divide-and-conquer) or concatenating all packets into one long sequence, create a dilemma. The former suffers from critical information loss by

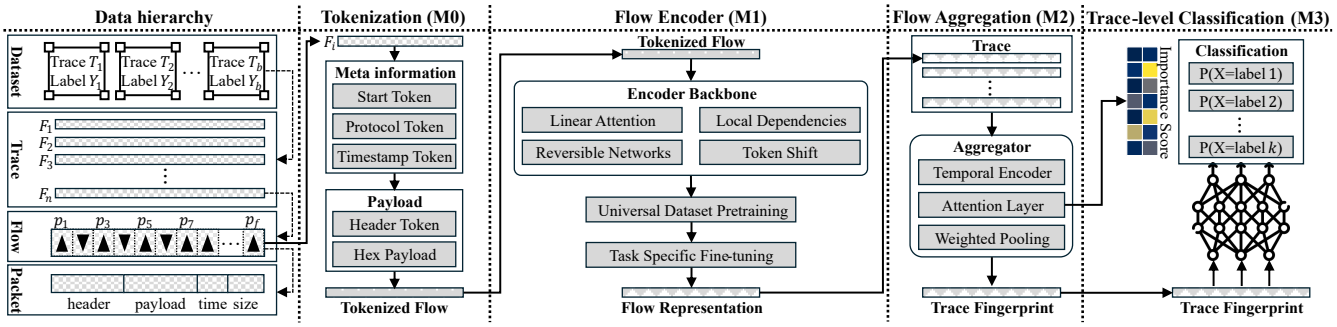


Figure 2: Overview of `Tracegram` framework.

ignoring inter-flow context, while the latter is computationally infeasible due to the $O(L^2)$ complexity of Transformers [30].

To break this impasse, our research goal is to design a novel framework that can natively process multi-flow traces, characterized by three key properties: *Principled*, *Scalable*, and *Interpretable* [59]. The framework must be *Principled*, grounded in MIL to holistically model traces without lossy preprocessing. It must be *Scalable*, circumventing the quadratic complexity of large models for efficient deployment. Critically, it must be *Interpretable*, providing actionable intelligence by identifying the key flows that determine the final classification [34].

2.2 Trace-Level MIL Formulation

For the purpose of rigorous analysis, we provide the following formal definitions: A packet p is a tuple $p = (\text{hdr}, \text{pld}, t, s)$. Here, hdr contains header information such as the 5-tuple (src IP, dst IP, src port, dst port, protocol), pld is the application-layer payload, t is its capture timestamp, and s is the packet size. A flow F_j is a sequence of f packets that share the same 5-tuple, i.e., $F_j = \langle p_1, p_2, \ldots, p_f \rangle$. A trace T is the ordered sequence of all packets from a complete network activity, consisting of n heterogeneous flows, $T = \bigcup_{j=1}^{n} F_j$.

To connect this problem with a powerful theoretical framework, we reformulate it from the perspective of MIL. We abstract a complete trace T as a bag B, where $B = \{F_1, F_2, \ldots, F_n\}$. Correspondingly, each individual flow F_j within the trace is treated as an instance $x_j \in B$.

Under this formulation, the real-world annotation challenge maps precisely to the MIL setting. We can typically only obtain a bag-level label $Y_B = Y_T$, while the fine-grained instance-level labels y_j (the true labels of each flow) are unknown. Therefore, our learning objective is to learn a classifier that can accurately classify a new, unseen trace using only the known, coarse-grained bag-level labels for supervision. To achieve this, our MIL framework first encodes each flow F_j into a representation z_j, then aggregates the set $\{z_1, \ldots, z_n\}$ into a single trace representation z_T, on which the final classification is performed.

3 `Tracegram`: Architecture and Design

This section details the four modules of the `Tracegram` framework. These components collectively form a staged traffic analysis pipeline that adheres to the MIL paradigm. The pipeline sequentially comprises four modules: Packet Tokenization (M0), Flow(Instance) Encoder (M1), Flow(Instance) Aggregation (M2), and Trace-Level Classification (M3).

3.1 Framework Overview

Based on the design goals established previously to address the challenges of annotation and attribution, we propose `Tracegram`, a hierarchical, multi-stage traffic analysis framework. The overall architecture of the framework is illustrated in Figure 2. It strictly adheres to the core principles of MIL. The framework systematically decomposes the trace-level analysis task into a four-stage processing workflow: Packet Tokenization, Instance Encoding, Instance Aggregation, and Trace Classification. The two critical sub-tasks within this workflow are Instance Encoding and Instance Aggregation.

The first stage of the pipeline is Packet Tokenization. Its core responsibility is to convert raw packets into numerical inputs understandable by deep learning models. This preprocessing step serves as a bridge between the raw network traffic and the subsequent complex neural network models. The quality of its design directly determines the upper bound of information that the model can learn from the data.

The second stage of the pipeline is the Flow Encoder. In this stage, the model processes each individual flow within a Trace. The encoder takes a token sequence corresponding to a flow as input. Its objective is to distill the rich information within each flow and encode it into a compact, highly discriminative feature vector. We refer to this vector as the Instance Representation. This stage is critical. It independently and concurrently captures the intrinsic behavioral pattern of each flow, providing high-quality inputs for the subsequent aggregation and decision-making processes. The quality of these instance representations directly determines the theoretical performance ceiling of the entire framework.

The third stage of the pipeline is Flow Aggregation. This module receives an ordered sequence of instance vectors from the flow encoders. Its core task is to aggregate these instance representations to produce a single feature vector representing the global properties of the entire Trace. We term this vector the *Trace Fingerprint*. More importantly, the aggregation model is designed to identify and assign higher weights to the key instances that are decisive for the final classification. This design is fundamental to achieving the model's attribution.

The final stage of the pipeline involves applying the Trace Fingerprint to Downstream Tasks. This stage takes the versatile Trace Fingerprint from the aggregation module as input. For a specific task such as classification, a lightweight, task-specific head (typically consisting of one or more fully-connected layers) is appended. This head maps the fingerprint to the desired output space, for instance, by outputting the predicted probabilities for each class. By decoupling the generation of a general-purpose Trace Fingerprint from the task-specific head, our framework can adapt to a variety of traffic analysis tasks with great flexibility and efficiency.

Through this four-stage pipeline design, the `Tracegram` framework transforms a difficult long-sequence problem into a feasible Tokenization-Encoder-Aggregation Task paradigm. This design processes multiple independent, shorter flow sequences at the instance encoding stage. This approach fundamentally avoids the quadratic computational bottleneck associated with monolithic end-to-end models, thereby achieving *Scalable*. Furthermore, the introduction of the MIL paradigm provides a *Principled* solution for learning with only coarse-grained supervision. The aggregation module's inherent mechanism for identifying key instances also provides a solid foundation for the model's *Interpretable*. This directly addresses the core design goals established in the previous section.

Before detailing the specific pipeline modules, § 3.2 first defines the fundamental unit (Trace) of analysis and discusses strategies for its construction. Subsequently, § 3.3 describes Packet Tokenization. § 3.4 and § 3.5 will respectively detail the theoretical foundations and specific implementations of the two core modules: the Encoder and the Aggregation module. § 3.6 will explain how the Trace Fingerprint is applied to classification tasks. Finally, § 4 will theoretically prove the validity of applying the MIL method to the network traffic domain, demonstrating the equivalence between the decomposed sub-tasks and the original task.

3.2 Trace Construction and Constraints

In the context of MIL, the **Trace** corresponds to the bag, serving as the fundamental unit of analysis. Constructing a valid trace from continuous, raw network traffic is the first and most critical step. Unlike fixed datasets, real-world traffic is a continuous stream. Therefore, practitioners must define logical boundaries to segment this stream into discrete units.

Trace Construction Strategies. We categorize trace construction into three systematic approaches, each with distinct implications for analysis:

1) Time-based Windowing. In scenarios requiring continuous monitoring (e.g., encrypted traffic surveillance), traffic is often unbounded. A practical approach is to slice traffic associated with a specific target into fixed time windows (e.g., $\Delta t = 5s$). This method ensures real-time processing capability suitable for online deployment, though it carries the risk of splitting a single logical activity across two traces.

2) Interaction-based Grouping. For tasks focusing on specific user actions (e.g., webpage visits or command execution), traces should be defined by logical interaction boundaries. Practitioners can utilize delimiter flows (e.g., DNS queries) or idle time gaps to delimit these sessions. This approach maximizes semantic consistency within a trace, allowing the model to learn precise causal relationships, but relies on the availability of identifiable delimiters.

3) Entity-based Profiling. In device fingerprinting or forensics, the focus shifts to the cumulative behavior of a host over a period (e.g., a startup sequence). This strategy captures the long-term profile of an entity rather than transient actions, making it robust against short-term behavior fluctuations.

Practical Constraints and Parameter Selection. Deploying `Tracegram` involves addressing complex trade-offs between comprehensive coverage and computational efficiency:

1) Tolerance to Noise. Ideally, a trace should contain the target behavior. However, perfect segmentation is difficult in practice. Our MIL formulation is designed to tolerate dirty bags with a significant ratio of background noise, reducing the burden on pre-processing logic to perform exact segmentation. The validity of this tolerance is demonstrated in § 5.4.

2) Determining Sequence Length. Practitioners should determine the truncation threshold L_{max} by analyzing the Cumulative Distribution Function of flow lengths in their target environment. The decision involves a critical trade-off: aggressive truncation saves memory but discards tail packets that may contain vital payload signatures. Unlike standard Transformers limited by $O(L^2)$ complexity, `Tracegram`'s linear complexity ($O(L)$) alters this calculus. We recommend selecting a threshold covering the **long tail** (e.g., 99^{th} percentile). This ensures the retention of complete flow context—crucial for robust classification—while maintaining high throughput comparable to shorter truncations in traditional models.

3.3 Packet Tokenization (M0)

Packet tokenization is the starting point of the entire processing pipeline. Its core objective is to convert raw network traffic data into a numerical representation for neural networks. We adapt and extend the tokenization schemes from existing work [35, 47]. Our goal is to design a conversion process that preserves all potential classification information. The process begins by extracting individual flows from the raw `.pcap` file based on the 5-tuple. Each flow is then independently

processed into a concatenated sequence of tokens from its constituent packets. To ensure informational integrity, a token block for each packet is composed of three parts.

Meta-information tokens. This part includes a `[Start]` token to mark a packet's beginning and a token for the link-layer protocol, such as Ethernet. They provide clear structural boundaries and context for the model to parse long sequences.

Timestamp Tokens. Time is one of the most critical features in traffic fingerprinting analysis. We quantize and encode the inter-arrival time between the current packet and its predecessor. This relative time representation enables the model to learn fine-grained temporal patterns at a microscopic level.

Payload Tokens. This is the most information-dense part, encapsulating the packet's complete header fields and application-layer payload. To handle all content uniformly, we convert all bytes of a packet into a hexadecimal representation, where each hexadecimal character is treated as an individual token. This universal representation ensures that the token sequence can be precisely restored to the original packet, thus achieving a bijective and lossless transformation.

3.4 Flow Encoder (M1)

After packet tokenization, the pipeline proceeds to the instance encoding stage. This is the cornerstone of `Tracegram`. The core responsibility of the flow encoder is to learn a powerful mapping function. This function encodes each raw flow, a variable-length token sequence, into a fixed-dimensional flow representation that captures its intrinsic behavior patterns. The quality of the Instance Representation directly determines the theoretical upper limit of the framework's performance.

Backbone Architecture. The encoder backbone integrates several techniques to balance high performance and efficiency when processing long sequences like individual flows. Its core is a pre-trained architecture based on linear attention [26]. The traditional self-attention mechanism in Transformers has a complexity of $O(L^2)$. This makes it impractical for processing a single flow with thousands of packets. Linear Attention leverages the associative property of matrix multiplication to reduce computational and memory complexity to $O(L)$. This greatly enhances the model's ability to handle long instance sequences. To further improve performance and efficiency, the backbone also integrates other key technologies. (1) The Local Attention mechanism [76] enables the model to focus on capturing local dependencies in the packet sequence, which is crucial for understanding micro-burst traffic patterns. (2) The design of Reversible Networks [28] optimizes memory consumption, enabling the construction of deeper networks without sacrificing performance. (3) Token Shift [53] accelerates information flow between layers, promoting information fusion among adjacent tokens at a minimal computational cost and thus speeding up model convergence.

Two-Step Supervised Tuning. To maximize the representation capability of the flow encoder, `Tracegram` adopts a two-step supervised tuning process inspired by the Universal Sentence Encoder [7]. In the first stage, the backbone network is fine-tuned on a large-scale, diverse public dataset to build a Universal Flow Encoder that understands general traffic patterns. The second stage involves a second round of fine-tuning, task-specific fine-tuning, on the dataset of a specific downstream task. In this stage, only trace-level (bag-level) labels are available. We therefore adopt a common MIL training strategy. During training, we temporarily assign the label of each trace to all the flows (instances) it contains. This serves as the supervision signal to update the encoder's parameters. This process enables the model to adapt its general traffic knowledge and to focus on identifying the most critical and subtle instance features for the specific task.

3.5 Flow Aggregation (M2)

Once the flow encoder has processed all flows in a trace into high-dimensional feature vectors, the pipeline proceeds to the instance aggregation stage. The core responsibility of this module is to aggregate a series of instance representations. The goal is to generate a single feature vector, the trace fingerprint, which represents the global properties of the entire trace (bag). Its input is a tensor $\mathbf{X}$ with the shape $B \times L \times D$, where B is the batch size, L is the maximum number of flows in a trace, and D is the dimension of the instance representation.

Unlike many traditional MIL methods that treat instances as an unordered set. However, in network attack scenarios, the order of flows often contains important logical relationships. Therefore, our model first uses a Bidirectional Long Short-Term Memory (Bi-LSTM) network [89] to scan the instance sequence. This enables the model to consider both preceding instances (historical context) and subsequent instances (future context) when processing the i-th instance, thereby effectively capturing temporal dependencies among instances.

However, not all instances are equally important for the final decision. For example, in an attack trace, a C2 tunnel flow is far more critical than a normal DNS query flow. We thus introduce an attention mechanism after the Bi-LSTM layer. This mechanism learns a weight for each instance representation in the sequence. This weight directly reflects the importance of the instance in determining the properties of the entire trace. This mechanism provides an interpretability benefit, allowing `Tracegram` to attribute predictions to specific key flows. By analyzing the attention weights, we can identify what the model considers to be the key instances.

Finally, a single Trace Fingerprint is generated by a weighted summation of all instance representations. It fuses information from all instances, with an emphasis on critical ones. The detailed algorithmic procedure for this aggregation process is formalized in Algorithm 1.

Algorithm 1 M2: Flow Aggregation with Attention

Require: Sequence of Flow Embeddings $H = \{h_1, ..., h_n\}$ from M1

Step 1: Temporal Context Encoding

$H' \leftarrow \text{BiLSTM}(H)$ {Process sequence to capture dependencies}

Step 2: Attention Weight Computation

$U \leftarrow \tanh(H' \cdot W_\omega + b_\omega)$ {Learn hidden representation}

$\alpha \leftarrow \text{softmax}(U \cdot u_\omega^T)$ {Compute normalized importance scores}

Step 3: Weighted Aggregation

$z \leftarrow \sum_{i=1}^{n} \alpha_i h'_i$ {Generate Trace Fingerprint}

return Trace Fingerprint z, Attention Weights α

3.6 Trace-Level Classification (M3)

The final stage of the pipeline performs Trace-Level Classification. The Trace Fingerprint, generated by the Flow Aggregation module, encapsulates the global properties of the entire trace. To evaluate its effectiveness, we attach a lightweight classification head (typically fully-connected layers) to the end of the framework. This head maps the highly condensed Trace Fingerprint to the label space of the specific classification task. By decoupling the generation of the Trace Fingerprint from the classifier, our framework ensures that the learned representations are robust, while the lightweight head effectively reduces the risk of overfitting on limited datasets.

4 Theoretical Justification of the MIL Decomposition

This section gives a high-level justification for decomposing trace analysis into Flow Encoding (M1) and Flow Aggregation (M2), grounded in information theory. Detailed mathematical proofs and derivations are provided in Appendix A.2.

4.1 Instance Separability and Hierarchical Optimization

Directly modeling a bag B with many heterogeneous instances to maximize the likelihood $L(Y_B|B;\theta)$ is computationally intractable. However, in traffic analysis, the label Y_T of a trace T often depends on specific key flows (e.g., a C2 heartbeat). We thus propose the **Instance Separability Assumption**: The label of a trace depends primarily on its key flows, which are separable from background flows in the feature space.

This assumption allows us to decompose the optimization into a hierarchical objective:

$$\begin{cases} \hat{\theta} = \langle \hat{\theta}_{enc}, \hat{\theta}_{agg} \rangle, \\ \hat{\theta}_{enc} = \arg\max_{\theta_{enc}} \prod_{j=1}^{n} L(Y_B|F_j;\theta_{enc}), \\ \hat{\theta}_{agg} = \arg\max_{\theta_{agg}} L(Y_B|B;\hat{\theta}_{enc},\theta_{agg}). \end{cases} \tag{1}$$

Here, θ_{enc} learns to distinguish flows (M1), and θ_{agg} learns to identify key flows within the bag (M2).

4.2 Information-Theoretic View

We justify this decomposition using mutual information. The total information in a trace T relevant to label Y_B can be decomposed using the chain rule:

$$I(Y_B;T) = I(Y_B;\mathcal{C}) + I(Y_B;\mathcal{T}_{\text{inter}}|\mathcal{C}) \tag{2}$$

where $\mathcal{C}$ represents the **Intra-Instance Information** (independent content of flows), and $\mathcal{T}_{\text{inter}}$ represents the **Inter-Instance Information** (temporal and contextual relations).

Our framework's two stages map to these components:

- **Flow Encoder (M1)** maximizes $I(Y_B;\mathcal{C})$ by extracting intrinsic features from individual flows, temporarily ignoring inter-flow relationships.
- **Flow Aggregation (M2)** maximizes $I(Y_B;\mathcal{T}_{\text{inter}}|\mathcal{C})$ by modeling the sequence and timing relationships among flows.

4.3 Assumption Validity and Robustness

It is important to clarify that the *Instance Separability Assumption* and the associated *Orthogonality Criterion* (detailed in Appendix A.2) are idealized assumptions. In real-world traffic, flows are not always strictly independent; a trace's malicious nature may emerge from the interaction of multiple flows (high $I(Y_B;\mathcal{T}_{\text{inter}}|\mathcal{C})$).

However, these assumptions are not strict prerequisites. The M2 Aggregator is explicitly designed to capture the inter-instance dependencies ($\mathcal{T}_{\text{inter}}$) that M1 treats as independent. Our robustness experiments (§ 5.4) confirm this: even when traces contain significant noise or mixed behaviors (violating strict separability), the model maintains high performance. This suggests the framework effectively approximates the optimal solution even when ideal conditions are relaxed.

5 Performance Evaluation

In this section, we systematically evaluate the `Tracegram` framework through a series of experiments. The evaluation is designed to answer three core research questions.

Table 1: Neural network parameter settings for different modules in Tracegram.

Module	Function	Neural Networks	Network Layers	Output Size
M1	Packet Vector Sequence Compression	CNN	Conv1d(in_channels=3, out_channels=10, kernel_size=5)	$n \times 10$
			MaxPool1d(kernel_size=3, stride=3, padding=1)	$\approx \text{int}(n/3) \times 10$
			Conv1d(in_channels=10, out_channels=50, kernel_size=5);	$\approx \text{int}(n/3) \times 10$
			MaxPool1d(kernel_size=3, stride=3, padding=1)	$\approx \text{int}(n/9) \times 10$
			Conv1d(in_channels=50, out_channels=100, kernel_size=5);	$\approx \text{int}(n/9) \times 10$
			MaxPool1d(kernel_size=3, stride=3, padding=1)	$\approx \text{int}(n/27) \times 10$
	Flow Encoder	Linear Attention	BiLSTM(input_size=100, hidden_size=100, num_layers=1)	Len(flows)×100
			size(Q)= 50×100;	50×100
			Conv1d(50→50, kernel_size=1);	50×100
			Conv1d(50→10, kernel_size=1)	10×100
M2	Flow Aggregation	BiLSTM & Attention	BiLSTM(input_size=100, hidden_size=100, num_layers=1)	Len(flows)×100
			Size(Q)= 50×100	50×100
			Conv1d(50→50, kernel_size=1)	50×100
			Conv1d(50→10, kernel_size=1)	10×100
M3	Trace-to-label Classification	Fully Connected	Linear(in=1000, out=100)	100
			Linear(in=100, out=50)	50
			Linear(in=50, out=50)	50
			Linear(in=50, out=*class number*)	*number of classes*

- **RQ1.** How does the classification performance of the Tracegram framework compare to SOTA methods based on deep learning or handcrafted features on end-to-end trace classification tasks?
- **RQ2.** To what extent do the key architectural designs of Tracegram contribute to its performance and robustness?
- **RQ3.** Can the MIL paradigm introduced by the Tracegram framework offer attribution by accurately localizing key malicious instances within a complex, real-world attack trace containing significant background noise?

5.1 Datasets

We evaluate Tracegram on four datasets, strictly applying the trace construction strategies defined in § 3.2 to suit each scenario.

User Activities (UAV) [29]. Representing a continuous monitoring scenario, we apply Time-based Windowing to this dataset. Continuous traffic is sliced into non-overlapping 5-second traces to simulate real-time classification needs.

IoT Device Identification (IDI) [48]. Containing 27 device types, we employ Entity-based Profiling. Each trace captures the distinct activation phase (startup process) of a device, providing a comprehensive behavioral profile.

Intrusion Detection (ISD) [24]. Distinguishing malware from benign traffic, we use Interaction-based Grouping. Flows are aggregated from specific benign usage sessions or malware execution sessions to form complete traces.

Keyword Searching (KWS) [56]. For this fine-grained task (50 keywords), we also utilize Interaction-based Grouping. A trace corresponds strictly to the sequence of encrypted flows triggered by a single keyword search event.

Constraints. Adhering to the trade-offs discussed in § 3.2, we set the maximum sequence length L_{max} to **12,032 tokens**. This value targets $\geq 99^{th}$ percentile coverage based on our CDF analysis, prioritizing context retention. Detailed dataset statistics are provided in Appendix A.3.

5.2 Experimental Setup

We evaluate Tracegram using a rigorous experimental protocol to ensure fair comparison and reproducibility across diverse network environments.

Implementation Details. Our model is implemented using PyTorch [22] and trained on a single NVIDIA RTX 3080Ti GPU (10 GB VRAM), highlighting the framework's memory efficiency. The architecture balances capacity and speed: the pre-trained backbone contains $\approx 2.0 \times 10^8$ parameters, while the task-specific components (Instance Encoder decoder and Trace Classifier) add only $\approx 2.5 \times 10^6$ parameters combined. During fine-tuning, we employ the AdamW optimizer [38] (LR=1×10^{-3}, batch size=16) with early stopping. Datasets are partitioned into training, validation, and test sets using an 8:1:1 ratio. Regarding the trace constraints discussed in § 3.2, we set the maximum sequence length L_{max} to 12,032 tokens. Detailed architectural parameters are summarized in Table 1.

Baselines. For a comprehensive comparison, we benchmark Tracegram against several SOTA methods:

- **K-Means+RF** [29]: A classic baseline method based on feature engineering and traditional classification models.
- **Hybrid Attention** [56]: An advanced trace-level classifier that serves as our primary deep learning baseline.
- **Transformer Baselines**: We adapt SOTA flow-level pre-

Table 2: Performance Comparison of Tracegram with Various Classification Methods.

Method	UAV [29]			KWS [56]			IDI [48]			ISD [24]		
	PR	RC	F1	PR	RC	F1	PR	RC	F1	PR	RC	F1
K-Means + RF [29]	0.9717	0.9672	0.9682	0.5569	0.5123	0.5184	0.7754	0.7451	0.7458	0.8561	0.8276	0.9057
RF& CCR-ELM [15, 17]	0.9410	0.9354	0.9382	0.9055	0.8881	0.8967	0.8892	0.8708	0.9100	0.9751	0.9382	0.9610
Hybrid Attention [56]	0.9307	0.9201	0.9280	**0.9804**	0.9705	**0.9773**	0.9442	0.9389	0.9401	0.9918	0.9836	0.9840
ET-BERT* [35]	0.9575	0.9592	0.9582	0.9620	0.9608	0.9615	0.9890	0.9875	0.9885	0.9855	0.9870	0.9862
TrafficFormer* [87]	0.9655	0.9640	0.9648	0.9715	**0.9730**	0.9723	0.9915	0.9908	0.9912	0.9902	0.9910	0.9905
Tracegram (Ours)	**0.9803**	**0.9700**	**0.9751**	0.9536	0.9588	0.9539	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**

* Adapted from flow-level methods using the bag-of-flows strategy described in §5.2

trained models, specifically **ET-BERT** [35] and **TrafficFormer** [87], for the trace-level task.

- We also include methods reporting SOTA performance on specific datasets, such as **RF** [15] and **CCR-ELM** [17].

Since the Transformer baselines are natively designed for single flows, we adapt them using a *bag-of-flows* strategy: each flow is encoded independently, and embeddings are aggregated via mean-pooling to form a fixed-length trace representation for the linear classifier. This baseline effectively represents the capability of SOTA flow encoders lacking our temporal aggregation logic. Specific configurations for all baselines are detailed in Appendix A.6.

5.3 Tracegram *vs.* Baseline Defenses

To answer RQ1, evaluating Tracegram against SOTA methods, we conducted comprehensive comparative experiments. The detailed performance metrics are presented in Table 2.

Performance on Structural Tasks (IDI, ISD, UAV). The results demonstrate Tracegram's dominance on tasks defined by flow composition and temporal structure. On the IDI and ISD tasks, Tracegram achieved a perfect F1-score of **1.000**. While the adapted Transformer baselines (ET-BERT and TrafficFormer) also performed strongly (F1 $>$ 0.98), they failed to achieve perfect classification. This indicates that explicitly modeling flow *composition* via our aggregation module is more effective for device and intrusion identification than the simple mean-pooling used by the baselines. On the more challenging UAV dataset, Tracegram achieved an F1-score of **0.9751**, surpassing all baselines, including the SOTA TrafficFormer (0.9648). This confirms that capturing the inter-arrival timing and order of flows—capabilities inherent to our M2 module—is critical for distinguishing complex user activities.

Comparison on Fine-grained Content Task (KWS). On the KWS task, which relies on identifying subtle patterns within encrypted payloads, the Transformer baselines exhibited a slight advantage. TrafficFormer achieved the highest F1-score of **0.9723**, compared to 0.9539 for Tracegram. This result acknowledges a trade-off: the quadratic self-attention ($O(L^2)$) in Transformers is exceptionally capable of extracting fine-grained payload features. However, Tracegram still significantly outperforms traditional methods and the previous Hybrid Attention baseline, achieving a competitive accuracy with a much lower linear complexity ($O(L)$).

Summary. Overall, Tracegram offers a superior trade-off. It matches or exceeds SOTA performance on structural and temporal tasks (UAV, IDI, ISD) while maintaining high efficiency, making it a more practical choice for large-scale trace analysis than computationally heavy Transformers.

Answer to RQ1

Tracegram significantly outperforms handcrafted methods across all tasks. Compared to SOTA DL models, it achieves superior performance on structural tasks (F1=1.000 on IDI/ISD) and competitive results on payload-heavy tasks, proving its effectiveness against both feature-based and end-to-end baselines.

5.4 Ablations and Robustness Analysis

This section seeks to fully answer RQ2 through a series of ablation studies and sensitivity analyses. We investigate the underlying rationale behind the framework's key designs and its robustness against the influence of real-world factors.

Ablation studies of key components. We first evaluate the impact of the choice of instance encoder on the final performance, as it is the cornerstone of the framework for learning high-quality instance representations. As shown in Table 3, we compared our linear-attention-based encoder with a series of representative SOTA traffic encoders, ranging from a classic CNN to various advanced Transformer architectures.

The experimental results reveal several key trends. Transformer-based encoders outperform the classic CNN baseline across all tasks. This provides strong evidence for the inherent advantage of the Transformer architecture in capturing complex, global dependencies within a flow. Among the standard Transformer models, Bigram and Datagram2Token

Table 3: Performance Comparison with Different Flow Encoders

Encoder Method	UAV			KWS			IDI			ISD		
	PR	RC	F1	PR	RC	F1	PR	RC	F1	PR	RC	F1
CNN [31]	0.7600	0.7500	0.7550	0.6900	0.6730	0.6814	0.8200	0.8270	0.8235	0.8450	0.8374	0.8412
Bigram [18]	0.9432	0.9398	0.9415	0.9185	0.9236	0.9211	0.9751	0.9697	0.9724	0.9823	0.9887	0.9855
Datagram2Token [35]	0.9401	0.9463	0.9432	0.9234	0.9157	0.9195	0.9765	0.9810	0.9788	0.9880	0.9842	0.9861
T5 Encoder [72]	0.9601	0.9576	0.9588	0.9455	0.9509	0.9482	0.9901	0.9949	0.9925	0.9918	0.9951	0.9934
MAE [86]	0.9595	0.9625	0.9610	0.9429	0.9401	0.9415	0.9912	0.9890	0.9901	0.9948	0.9965	0.9956
`Tracegram` (Ours)	**0.9803**	**0.9700**	**0.9751**	**0.9536**	**0.9588**	**0.9539**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**

show similar performance, implying different traffic tokenization strategies can have a subtle impact on specific tasks.

Furthermore, the T5 Encoder and MAE, which employ more advanced pre-training paradigms, outperform standard Transformer methods on most metrics. This demonstrates the effectiveness of more complex pre-training tasks that are specifically designed for traffic characteristics.

However, despite the strong performance of these advanced baseline methods, the flow encoder used by `Tracegram` achieves the best performance across all four datasets. Notably, on the IDI and ISD datasets, it achieves a perfect F1-score of 1.000, a result unmatched by any other method. This comprehensive lead provides powerful evidence for the superiority of our encoder's architecture. Theoretically, it solves the quadratic complexity problem of traditional Transformers through its linearized attention mechanism. Practically, it demonstrates an exceptional ability to capture the key discriminative features of a single flow (instance). This capability provides the highest quality input for the subsequent aggregation and classification stages, thus laying a solid foundation for the success of our entire `Tracegram` framework.

The experiments systematically replaced our flow encoder with the following five methods for comparison:

- CNN [31]: Treats network flow payloads as images or sequences and uses sliding convolutional kernels to extract local pattern features. It is computationally efficient but struggles to capture long-range dependencies.
- Bigram [18]: Preprocesses flow payloads into a sequence of byte pairs and uses the multi-head self-attention mechanism of a Transformer Encoder to capture global context and long-range dependencies. Its main drawback is the quadratic computational complexity.
- Datagram2Token [35]: Aggregates packets within a flow into BURST sequences by transmission direction to preserve interaction patterns. It then converts the payloads within each BURST into bi-gram tokens for encoding, effectively the structure and transport characteristics of traffic.
- T5 Encoder [57]: Uses the encoding method from the Text-to-Text Transfer Transformer. Building on this, Lens [72] pre-trains the model on a multi-task objective deeply customized for traffic characteristics, including tasks like Packet Order Prediction, to encode network flows.
- MAE [19]: A self-supervised learning paradigm for traffic encoding by YaTC [86]. It first represents a flow as an MFR matrix. During pre-training, it randomly masks a large portion of the matrix and trains an encoder to reconstruct the complete original flow from the small visible part.

Next, we validate the necessity of the two-stage supervised tuning strategy, particularly the task-specific fine-tuning stage. The results in Table 4 reveal a striking difference. Without fine-tuning, the model performed adequately on tasks like UAV and IDI. However, on the highly challenging KWS task, its F1-score was only 0.3952, indicating substantial underperformance in classification. After applying task-specific fine-tuning, performance on the KWS task surged to an F1-score of 0.9539, and performance on the UAV task also improved significantly. This result clearly demonstrates that task-specific fine-tuning is a critical step for unlocking the model's full potential. It allows the model to adapt its general traffic knowledge learned during pre-training and focus on the subtle features of a specific downstream task.

Finally, we explore the importance of application-layer payload information. The four experimental setups in Table 5 provide a broad perspective. Under the optimal fine-tuned configuration (+Fine-Tuning +Payload *vs.* +Fine-Tuning -Payload), removing the payload caused a significant performance drop on the UAV, KWS, and IDI tasks. This confirms that the payload is a key source of discriminative information. An interesting observation emerged in the non-fine-tuned setting (-Fine-Tuning +Payload *vs.* -Fine-Tuning -Payload). Removing the payload actually improved performance on the KWS task. This may indicate that the model without fine-tuning cannot correctly utilize the complex patterns in the payload and may even be negatively affected by the noise within it. In contrast, the fine-tuning process teaches the model how to effectively extract task-relevant signals from the payload.

Robustness and scalability analysis. We evaluate the `Tracegram` framework from two perspectives. We assess its robustness against random, non-adaptive noise when its core assumption is challenged, and its computational scalability when processing large-scale data. We first evaluate the robust-

Table 4: Impact of Task-Specific Fine-tuning (F1-score)

Method	UAV	KWS	IDI	ISD
+ Fine-tuning	**0.9751**	**0.9539**	**1.000**	**1.000**
- Fine-tuning	0.8814	0.3952	0.9313	**1.000**

Table 5: Impact of Application Layer Payloads (F1-score)

Method	UAV	KWS	IDI	ISD
+ Fine-Tuning + Payload	**0.9751**	**0.9539**	**1.000**	**1.000**
+ Fine-Tuning - Payload	0.8190	0.6270	0.8750	**1.000**
- Fine-Tuning + Payload	0.8814	0.3952	0.9313	**1.000**
- Fine-Tuning - Payload	0.7870	0.7940	0.8250	**1.000**

ness of the framework when its core Instance Separability Assumption is systematically violated. We directly challenge this assumption by injecting random noise into the training data during the fine-tuning stage. See Appendix A.5 for details on the noise used. For each trace in the training set, we randomly replace a certain percentage of its original flows with contaminating flows from other traces with different labels. The results are presented in Table 7. The data shows that our model exhibits a high degree of robustness. On the IDI and ISD tasks, which have distinct features, the model's performance is almost unaffected even with a noise ratio as high as 50%. On the more challenging UAV and KWS tasks, performance degrades only slightly and gradually when the noise ratio is below 40%. This result provides strong evidence that our method does not rely excessively on a perfect theoretical assumption. Instead, it remains robust in non-ideal network environments with significant background noise.

Efficiency and Latency Analysis. To demonstrate the practical feasibility of `Tracegram` for high-throughput deployment, we conducted a rigorous efficiency evaluation comparing it against the Transformer baselines (ET-BERT and TrafficFormer). All efficiency benchmarks were conducted on a workstation equipped with a single NVIDIA GeForce RTX 3080 Ti GPU (12GB VRAM), running Ubuntu 20.04. To ensure a fair and reproducible comparison, the Transformer baselines were configured with the standard BERT-base architecture using standard library implementations [35, 87] without specific kernel optimizations. We measured the Per-Trace Inference Latency and Peak GPU Memory Usage with a batch size of 1 to simulate a real-time streaming detection scenario. Evaluations were performed on traces with representative flow length distributions derived from the evaluation datasets. Reported results are averaged over 1,000 inference runs following 50 warm-up iterations to eliminate initialization overhead and jitter.

The comparative results are summarized in Table 6.

As shown in Table 6, the adapted Transformer baselines incur significantly higher computational overhead. This is inherent to their architecture: processing a trace with multiple flows requires running the heavy 12-layer self-attention mechanism ($O(L^2)$ complexity) repeatedly or on concatenated sequences, leading to high latency (e.g., 124.5 ms for ET-BERT). In contrast, `Tracegram` leverages a lightweight linear-attention encoder and a hierarchical aggregation design, achieving a theoretical complexity of $O(L)$. Empirically, this results in a **4.17× speedup** in inference latency and a substantial **80.4% reduction** in peak memory usage compared to ET-BERT. This efficiency is critical for real-world defense, allowing `Tracegram` to operate on resource-constrained edge devices where deploying full-scale Transformers would be computationally prohibitive.

Table 6: Efficiency Comparison: Inference Latency and Memory Cost per Trace.

Model	Complexity	Latency (ms)	Peak Mem (MB)
ET-BERT* [35]	$O(L^2)$	124.5	2450
TrafficFormer* [87]	$O(L^2)$	118.2	2280
`Tracegram`	**O(L)**	**29.8**	**480**

* Adapted to the trace-level task using a bag-of-flows strategy.

Hyperparameter sensitivity analysis. To investigate the model's performance sensitivity to key hyperparameters, we studied the effect of varying *Flow* truncation lengths on classification performance. The truncation length determines the maximum amount of contextual information that the flow encoder (M1) module can observe for each *flow* (instance). We set five different truncation points between 512 and 12,032 tokens and conducted a comprehensive evaluation across the four tasks. The experimental results are shown in Table 8.

We can draw several key observations from the experimental results. For tasks with relatively distinct features, such as ISD, the truncation length has a negligible impact. Even with the shortest setting of 512 *tokens*, the model achieves a perfect F1-score of 1.0. This indicates that the key malicious patterns are concentrated at the beginning of the *flow*, making short sequences sufficient for accurate identification.

For the more challenging UAV and KWS tasks, the relationship between model performance and truncation length is non-monotonic. In the UAV task, performance peaks at a length of 4,096 (0.975), while for the KWS task, the best performance occurs at a length of 2,048 (0.954). When the sequence length exceeds this optimal range, performance actually shows a slight decline. We infer two possible reasons for this. Longer sequences might introduce irrelevant noise to the classification task, such as generic connection-closing signals at the end of a session, which could interfere with the model's judgment. Alternatively, this phenomenon might be related to the maximum sequence length used during the pre-training phase of our *encoder*, where excessively long inputs could prevent the model from generating optimal representations.

In the IDI task, we observed a notably different and inter-

Table 7: Model Performance under Varying Noise Ratios

Noise Ratio	UAV			KWS			IDI			ISD		
	PR	RC	F1	PR	RC	F1	PR	RC	F1	PR	RC	F1
0.00%	**0.9803**	**0.9700**	**0.9751**	**0.9536**	**0.9588**	**0.9539**	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**
10.00%	0.9716	0.8561	0.8802	0.9284	0.9284	0.8725	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
20.00%	0.9639	0.8534	0.8745	0.8767	0.8699	0.8631	1.0000	1.0000	1.0000	0.9750	0.9988	0.9866
30.00%	0.9680	0.8534	0.8770	0.8860	0.8922	0.8842	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
40.00%	0.9613	0.8506	0.8719	0.8600	0.8508	0.8448	0.9340	0.9583	0.9440	0.9750	0.9988	0.9866
50.00%	0.8515	0.8358	0.8384	0.6973	0.7046	0.6927	0.9069	0.9236	0.9035	0.9988	0.9737	0.9859

Table 8: Impact of Flow Truncation Token Length (F1-score)

Truncation Length	UAV	KWS	IDI	ISD
512	0.876	0.870	0.908	**1.000**
1024	0.834	0.880	0.817	**1.000**
2048	0.848	**0.954**	0.924	**1.000**
4096	**0.975**	0.907	0.900	**1.000**
12032	0.881	0.911	**1.000**	**1.000**

esting trend. Performance generally improved with increasing length, reaching a perfect F1-score of 1.0 at the longest setting of 12,032. This suggests that for identifying certain IoT devices, their unique behavioral fingerprints may be distributed over a longer time series. Therefore, more complete contextual information is required for precise differentiation.

In summary, the optimal *Flow* truncation length is task-dependent. It requires a balance between capturing sufficient discriminative information and avoiding the introduction of noise. This finding also validates the importance of targeted hyperparameter tuning for different downstream tasks.

Answer to RQ2

`Tracegram`'s key designs are vital for performance and robustness. The linear-attention encoder outperforms SOTA encoders, and task-specific fine-tuning boosts KWS F1-score from 0.3952 to 0.9539. The decoupled architecture is more scalable than end-to-end models.

5.5 Attribution through MIL

This section aims to provide a clear and direct answer to RQ3 by demonstrating the attribution that the MIL paradigm brings to `Tracegram` through visualization.

Representation Space Visualization. To visually evaluate the powerful representation learning capabilities of the `Tracegram` framework, we visualized the Trace Fingerprints generated by our model for the four public datasets. We used two mainstream dimensionality reduction techniques for this visualization: t-SNE and UMAP. As shown in Figure 3, each colored point represents a single trace. The figure visually demonstrates that our framework learns highly discriminative representations. This capability is fundamental to achieving both accurate classification and subsequent attribution.

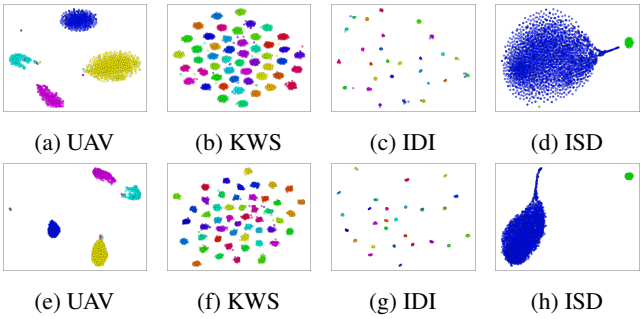
The figures clearly show that `Tracegram` successfully separates traces of different classes in the two-dimensional feature space across all four datasets. Particularly for the IDI and ISD tasks in Figure 3c, 3d, 3g, and 3h, the clusters for different classes are highly cohesive internally. Furthermore, the boundaries between these clusters are distinct, with almost no overlap. This observation aligns perfectly with the perfect classification results (F1-score of 1.0) reported in our performance evaluation. It explains from a feature space perspective why the model achieves such excellent performance.

On the more challenging UAV and KWS tasks in Figure 3a, 3b, 3e, and 3f, the visualization results consistently exhibit clear and well-defined clustering structures, despite the larger number of classes and greater task complexity. The clusters are more compact and have some minor but noticeable overlaps at the boundaries. This reflects the inherent difficulty of these tasks. Even so, the vast majority of samples are still correctly partitioned into their respective class clusters. This result provides strong evidence that our framework can learn sufficiently powerful trace-level representations, even for fine-grained classification tasks with subtle inter-class differences.

In summary, these visualization results provide strong, intuitive evidence for the effectiveness of `Tracegram`. They show that our framework is not a simple black box. Instead, it successfully maps complex, high-dimensional raw traffic data into a well-structured, class-separable low-dimensional feature space. This powerful representation learning capability is the fundamental prerequisite for achieving high-precision classification (RQ1) and meaningful attribution (RQ3).

Case Study: Locating Key Instances in APT Trace. To definitively validate `Tracegram`'s attribution and its ability to locate key malicious instances, we conducted a case study on the highly challenging DAPT dataset. This dataset is designed to simulate real-world Advanced Persistent Threat (APT) scenarios. It features an extremely low signal-to-noise ratio, where attack traffic is submerged in a massive volume of benign background traffic. This poses a particularly severe test for any model's threat localization capabilities.

We selected a typical, multi-stage APT case from the dataset for in-depth analysis. We then visualized the internal attention scores output by the Aggregation Model. As shown in Figure 4, a heatmap intuitively displays the basis for the



(a) UAV (b) KWS (c) IDI (d) ISD

(e) UAV (f) KWS (g) IDI (h) ISD

Figure 3: Trace representations from `Tracegram` using t-SNE (top row) and UMAP (bottom row) across the four evaluation datasets. Each colored point represents a trace.

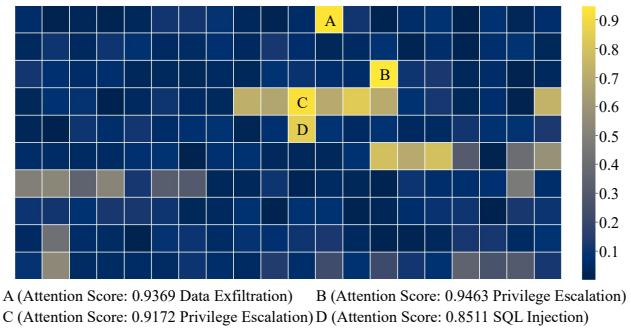


Figure 4: Locating Key Malicious Flows via Attention Mechanism

model's decisions. The vast majority of dark blue squares in the figure represent benign background flows. Their attention scores are all close to zero, which indicates that our model has successfully learned to ignore the overwhelming majority of the noise. Crucially, the model precisely focuses extremely high attention weights on a few known flows that represent critical steps of the attack. For example, the Privilege Escalation flow labeled B received the highest score of 0.9463, followed closely by the Data Exfiltration flow A (0.9369) and another Privilege Escalation flow C (0.9172). Additionally, the SQL Injection flow (labeled D), an early step in the attack chain, also received a high score of 0.8511. This finding provides strong evidence that `Tracegram` is not merely a simplistic black-box pattern matcher. Instead, it has genuinely learned, captured, and understood the nuanced internal logic of a complex attack. Without the need for fine-grained label supervision, it can precisely locate the most threatening core stages of an attack from a massive amount of noise, similar to the reasoning process of a security analyst. This capability transforms a raw alert into actionable intelligence.

Quantitative Validation on Attribution. Beyond the qualitative case study, we statistically validated the reliability of our attention mechanism across the DAPT dataset. We measured the *Attribution Recall*, calculating the percentage of ground-truth malicious flows covered by the top $\alpha\%$ of instances ranked by attention weights. Results demonstrate that the top **10%** of flows consistently cover over **95%** of the total attack behaviors. This implies that `Tracegram` effectively allows analysts to filter out 90% of the background traffic while retaining nearly all critical forensic evidence, significantly reducing the manual investigation workload.

Answer to RQ3

MIL yields flow-level attribution directly aligned with known cyber attack stages. On a DAPT APT case, the highest attention focuses on Privilege Escalation and Data Exfiltration flows (≈ 0.95 and ≈ 0.94), while background flows receive near-zero weight, providing concrete starting points for deeper investigation.

6 Discussion

Scope and Assumptions The core scenario for `Tracegram` is trace-level traffic analysis in environments with high encryption. Our methodology is based on a fundamental premise. Defenders can observe the header and temporal metadata sufficient to construct a trace, while labels are only available at the trace level, especially during training. Such metadata is routinely available in enterprise networks and backbone infrastructures, making this assumption realistic in both academic studies and operational deployments [13]. We acknowledge that trace construction is a non-trivial prerequisite [65]. The construction process may introduce noise, such as including irrelevant flows or missing certain flows. Therefore, a key design goal is to maintain robustness when processing these inevitably imperfect traces. This requirement is particularly critical in high-throughput backbone settings where thousands of concurrent sessions must be processed [69]. Another goal is to maintain acceptable latency and memory overhead, avoiding infinitely long sequences. This framework is applicable to various trace-level classification scenarios, including intrusion detection, APT activity analysis, website fingerprinting, and app service classification [36].

Limits of the Decomposition Our theoretical section supports the two-stage MIL objective under the Instance Separability Assumption. It also proves its equivalence to an ideal end-to-end objective under the Orthogonality Criterion. Note that these ideal conditions may be weakened in practice. For example, instance separability decreases when the feature distributions of flows in traces from different classes overlap greatly [77, 85]. Our model of decomposing a trace into independent instances for aggregation faces challenges when a trace's malicious nature is an emergent property [40]. This occurs when the property arises from complex, synergistic interactions among multiple flows over extended periods. Fur-

thermore, high redundancy within a trace, such as a large number of similar background flows, can dilute the contribution of a single key flow [1]. Such redundancy not only complicates the learning dynamics, but also mirrors the reality of backbone traffic where benign flows vastly outnumber malicious ones. To mitigate these effects, our aggregator incorporates a time-aware capability and leverages an attention mechanism to identify key signals. These design choices enhance the model's stability and its understanding of context, but they cannot completely eliminate all potential failure modes. In practical applications, we recommend using model calibration and confidence evaluation to identify these boundary cases and avoid overconfident misclassifications [70].

Robustness and Limitations. Our robustness experiments (§ 5.4) primarily evaluate resilience against *random noise* (e.g., packet loss, random background flow injection) typically found in non-adversarial network environments. We explicitly acknowledge that *adversarial perturbations*—such as carefully crafted padding to mimic benign statistics, or timing manipulation designed to evade the aggregator—are a limitation of this work. While the attention mechanism offers some resistance to simple decoy traffic, defense against adaptive, sophisticated adversarial attackers remains a significant open problem. Future work should explore incorporating adversarial training or formal verification methods to enhance system robustness against such targeted attacks.

Deployment and Generalization In deployment, `Tracegram` can support streaming analysis through sliding windows or micro-batches [61]. Due to our hierarchical and decoupled architecture, its memory and computational overheads are manageable. This avoids the bottlenecks associated with processing entire long sequences [62]. The model's output consists of a trace label and a list of flows, timestamped and ranked by importance. This *label + evidence* output format is designed to integrate seamlessly into a security analyst's workflow for event triage, investigation, and forensics [5, 85]. We must emphasize that the model's attribution is based on correlation, not causal proof. This distinction is important, since forensic analysts may otherwise overinterpret attribution results as causal evidence in legal or compliance contexts [6]. We therefore recommend logging attribution scores and decision thresholds to support the reproducibility of results. When generalizing to new network environments, the model may be affected by shifts in network policies and topology [16]. Since our method relies primarily on header and temporal features, its portability is already enhanced. For further performance optimization, modest fine-tuning on the target domain can typically restore most of the performance. Before deployment, validating the model with several different trace construction strategies can reduce the risk of overfitting to a specific session heuristic [67]. This can also more clearly reveal its generalization capabilities. Overall, these practices contribute to a smoother transition from research prototypes to field-ready systems, minimizing unexpected failures in production [27].

7 Related Work

Encrypted Traffic Classification. Traditional ML [74, 91] and recent deep learning methods [37, 39, 66, 79] have improved classification accuracy but remain strictly limited to single-flow analysis, failing to capture the cross-flow contexts defining complete behaviors. While they significantly improved classification accuracy, these methods remain limited to analyzing a single flow. They cannot capture the cross-flow contextual information that defines a complete user behavior. Recent work has started exploring the analysis of complete traces to overcome this bottleneck. While emerging trace-level methods [15, 17, 29, 56] attempt to bridge this gap, they predominantly rely on heuristic aggregation or engineering-driven architectures, lacking the rigorous theoretical grounding required for reliable attribution. In contrast, our framework is the first trace analysis method based on a principled orthogonal decomposition. We provide a rigorous theoretical foundation. This not only justifies our hierarchical approach but also enables us to formally analyze its performance boundaries. Our work therefore goes beyond previous heuristic aggregations and purely architectural innovations.

Pre-trained Models in Traffic Analysis. The pre-training paradigm has recently been introduced into the field of network traffic analysis. This was inspired by its tremendous success in natural language processing. Models like PERT [18], ET-BERT [35], TrafficFormer [87], NetGPT [47], and LENS [72] have emerged. These Transformer-based models are pre-trained on massive traffic datasets and achieve SOTA performance on single-flow classification tasks. However, a deep analysis of these works reveals a fundamental architectural limitation. Their design assumes that the input is a single linear sequence. In practice, this means their unit of processing is a single flow. These models must revert to the traditional approach when faced with complex, real-world traces composed of multiple concurrent flows. They inevitably lose cross-flow contextual information. For instance, some works [35, 72] try to indirectly learn packet order or intra-flow relationships through complex pre-training tasks. This approach implicitly highlights the information loss that arises from linearizing inherently multi-dimensional interactions. Departing from this single-sequence paradigm, our work introduces principled orthogonal decomposition into the pre-training paradigm, fundamentally addressing the inability of existing models to process heterogeneous traces. It aims to advance pre-training in network traffic analysis from single-flow level analysis to trace-level analysis.

Graph and Long-Sequence Models. Beyond flow-level analysis, recent research has explored modeling complex dependencies using Graph Neural Networks (GNNs) and long-sequence Transformers. GNN-based methods [20, 51, 52] construct traffic interaction graphs to capture topological structures, where nodes represent flows or hosts and edges represent communication channels. While effective at captur-

ing spatial correlations, they often treat flows as static nodes, struggling to efficiently model the fine-grained temporal evolution of packets within each flow. Long-sequence models, such as TrafficFormer [87] and variants of Longformer [4], attempt to process concatenated packets from all flows as a single long sequence to capture global context. However, they typically suffer from quadratic computational complexity ($O(L^2)$) or high memory costs, making them computationally prohibitive for processing entire traces that may contain thousands of packets. In contrast, `Tracegram` employs an MIL formulation that decouples local flow encoding from global temporal aggregation. This design achieves linear complexity ($O(L)$) while effectively preserving both intra-flow features and inter-flow temporal contexts, offering a superior trade-off between efficiency and performance.

8 Conclusion

We present `Tracegram`, a principled framework that pioneers a temporally-aware weakly supervised paradigm to resolve annotation bottlenecks and opacity inherent in traditional traffic analysis. By decoupling flow encoding from temporally aware aggregation, our model achieves perfect F1-scores of 1.0000 on the IDI and ISD datasets and sets a new SOTA on the challenging UAV task with 0.9751. To the best of our knowledge, this work is among the first to systematically formulate trace-level traffic classification. In particular, `Tracegram` is the first to adopt trace-level supervision while producing flow-level explanations via weakly supervised attribution. This develops a scalable and interpretable framework for traffic analysis systems, seamlessly bridging coarse-grained labels and fine-grained explanations to convert analytical insights into actionable, evidence-driven intelligence.

Acknowledgments

This work was supported in part by National Natural Science Foundation (U23A20332, 62272381, 623B2081, 62202405), Natural Science Basic Research Program of Shaanxi Province (2023-JC-JQ-50), Key Research and Development Program of Shaanxi (2024PT-ZCK-91), the Fundamental Research Funds for the Central Universities, Postdoctoral Science Foundation (2024T170722, 2023M732791), of China.

Ethical Considerations

We evaluated the ethical implications of `Tracegram`, focusing first on privacy. We utilized existing public datasets (UAV, IDI, ISD, KWS, DAPT) to avoid new data collection risks. Although utilizing tokenized encrypted payloads implies a side-channel privacy risk, we mitigate this by strictly classifying machine behaviors (e.g., malware C2, APTs) rather than profiling or deanonymizing individual human users. Regarding dual-use potential, we acknowledge that traffic fingerprinting capabilities could theoretically be repurposed for censorship or surveillance. However, we argue that the defensive necessity is paramount. As attackers increasingly hide within encrypted traffic, defenders require capabilities to detect stealthy threats without invasive decryption. The defensive benefits of empowering operators outweigh the potential misuse risks. Finally, regarding responsible disclosure, `Tracegram` is strictly defensive. To mitigate misuse, our code license explicitly restricts usage to defensive monitoring and research. Furthermore, our attributions represent statistical correlations, not causal forensics. Operationally, analysts must treat these as leads requiring verification to prevent unjust consequences from false positives.

Open Science

To facilitate reproducibility, we release the full implementation of `Tracegram` at Zenodo[1] and GitHub[2]. While licensing restrictions prevent us from redistributing the original datasets, they are publicly available from their respective authors: UAV[3], IDI[4], ISD[5], KWS[6], and DAPT[7]. Additionally, all baseline methods employed in our comparative analysis were sourced from their respective official open-source repositories to ensure fair comparison. Synthetic examples are included to facilitate testing. All code is released under an open-source license and will be updated with a non-anonymized link upon acceptance in compliance with USENIX policies.

References

[1] Mahmoud Abbasi, Sebastian Lopez Florez, Amin Shahraki, Amir Taherkordi, Javier Prieto, and Juan M Corchado. Class imbalance in network traffic classification: An adaptive weight ensemble-of-ensemble learning method. *IEEE Access*, 2025.

[2] Mehdi Asadi, Mohammad Ali Jabraeil Jamali, Arash Heidari, and Nima Jafari Navimipour. Botnets unveiled: A comprehensive survey on evolving threats and defense strategies. *Transactions on Emerging Telecommunications Technologies*, 35(11), 2024.

[3] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[1] https://doi.org/10.5281/zenodo.17978903
[2] https://github.com/YuchenZhang-Academic/Tracegram
[3] https://data.mendeley.com/datasets/5pmnkshffm/3
[4] https://github.com/andypitcher/IoT_Sentinel
[5] https://data.mendeley.com/datasets/nbs66kvx6n/1
[6] https://www.kaggle.com/datasets/fuukaa/network-multiflow-fingerprinting-datasets?datasetId=3270419
[7] https://www.kaggle.com/datasets/sowmyamyneni/dapt2020

[4] Iz Beltagy, Matthew E Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.

[5] Sonam Bhardwaj and Mayank Dave. Sql injection attack detection, evidence collection, and notifying system using standard intrusion detection system in network forensics. In *Proc. IEM-ICDC*, 2021.

[6] Yogita Borse, Deepti Patole, Gaurav Chawhan, Geeta Kukreja, Harsh Parekh, and Rishabh Jain. Advantages of blockchain in digital forensic evidence management. In *Proc. ICAST*, 2021.

[7] Daniel Cer, Yinfei Yang, Sheng-yi Kong, Nan Hua, Nicole Limtiaco, Rhomni St John, Noah Constant, Mario Guajardo-Cespedes, Steve Yuan, Chris Tar, et al. Universal sentence encoder. *arXiv preprint arXiv:1803.11175*, 2018.

[8] Haochen Chen, Bryan Perozzi, Yifan Hu, and Steven Skiena. Harp: Hierarchical representation learning for networks. In *Proc. AAAI*, volume 32, 2018.

[9] EE Chinwe and CHISOM ELIZABETH Alozie. Adversarial tactics, techniques, and procedures (ttps): A deep dive into modern cyber attacks. *ICONIC RESEARCH AND ENGINEERING JOURNALS*, 8(7), 2025.

[10] Thomas G Dietterich, Richard H Lathrop, and Tomás Lozano-Pérez. Solving the multiple instance problem with axis-parallel rectangles. *Artificial intelligence*, 89(1-2), 1997.

[11] Hailun Ding, Juan Zhai, Yuhong Nan, and Shiqing Ma. {AIRTAG}: Towards automated attack investigation by unsupervised learning with log texts. In *Proc. USENIX Security*, 2023.

[12] Jiayu Ding, Shuming Ma, Li Dong, Xingxing Zhang, Shaohan Huang, Wenhui Wang, Nanning Zheng, and Furu Wei. Longnet: Scaling transformers to 1,000,000,000 tokens. *arXiv preprint arXiv:2307.02486*, 2023.

[13] Yasir Ali Farrukh, Syed Wali, Irfan Khan, and Nathaniel D Bastian. Senet-i: An approach for detecting network intrusions through serialized network traffic images. *Engineering Applications of Artificial Intelligence*, 126, 2023.

[14] Gibran Gomez, Platon Kotzias, Matteo Dell'Amico, Leyla Bilge, and Juan Caballero. Unsupervised detection and clustering of malicious tls flows. *Security and Communication Networks*, 2023(1), 2023.

[15] Salma Abdalla Hamad, Wei Emma Zhang, Quan Z Sheng, and Surya Nepal. Iot device identification via network-flow based fingerprinting and learning. In *Proc. IEEE TrustCom*, 2019.

[16] Fujun Han, Peng Ye, Shukai Duan, and Lidan Wang. Ada-id: Active domain adaptation for intrusion detection. In *Proc. ACM MM*, 2024.

[17] Nasimul Hasan, Zhenxiang Chen, Chuan Zhao, Yuhui Zhu, and Cong Liu. Iot botnet detection framework from network behavior based on extreme learning machine. In *Proc. IEEE INFOCOM*, 2022.

[18] Hong Ye He, Zhi Guo Yang, and Xiang Ning Chen. Pert: Payload encoding representation from transformer for encrypted traffic classification. In *Proc. ITU Kaleidoscope*, 2020.

[19] Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. Masked autoencoders are scalable vision learners. In *Proc. IEEE/CVF CVPR*, 2022.

[20] Ting-Li Huoh, Yan Luo, Peilong Li, and Tong Zhang. Flow-based encrypted network traffic classification with graph neural networks. *IEEE Transactions on Network and Service Management*, 20(2), 2022.

[21] Maximilian Ilse, Jakub Tomczak, and Max Welling. Attention-based deep multiple instance learning. In *Proc. ICML*, 2018.

[22] Sagar Imambi, Kolla Bhanu Prakash, and GR Kanagachidambaresan. Pytorch. In *Programming with TensorFlow: solution for edge computing applications*. 2021.

[23] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proc. ICML*, 2015.

[24] Dorde Jovanovic and Pavle Vuletic. Etf iot botnet dataset, 2021. doi: 10.17632/nbs66kvx6n.1.

[25] Muhammad Monjurul Karim, Zhaozheng Yin, and Ruwen Qin. An attention-guided multistream feature fusion network for early localization of risky traffic agents in driving videos. *IEEE Transactions on Intelligent Vehicles*, 9(1), 2023.

[26] Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention. In *Proc. ICML*, 2020.

[27] Ansam Khraisat and Ammar Alazab. A critical review of intrusion detection systems in the internet of things: techniques, deployment strategy, validation strategy, attacks, public datasets and challenges. *Cybersecurity*, 4(1), 2021.

[28] Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. *arXiv preprint arXiv:2001.04451*, 2020.

[29] Víctor Labayen, Eduardo Magana, Daniel Morató, and Mikel Izal. Online classification of user activities using machine learning on network traffic. *Computer Networks*, 181, 2020.

[30] Franck Le, Mudhakar Srivatsa, Raghu Ganti, and Vyas Sekar. Rethinking data-driven networking with foundation models: challenges and opportunities. In *Proc. HotNets*, 2022.

[31] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11), 2002.

[32] Jianfeng Li, Zheng Lin, Xiaobo Ma, Jianhao Li, Jian Qu, Xiapu Luo, and Xiaohong Guan. Dnsscope: Fine-grained dns cache probing for remote network activity characterization. In *Proc. IEEE INFOCOM*, 2024.

[33] Jianfeng Li, Zheng Lin, Jian Qu, Shuohan Wu, Hao Zhou, Yangyang Liu, Xiaobo Ma, Ting Wang, Xiapu Luo, and Xiaohong Guan. Robust app fingerprinting over the air. *IEEE/ACM Transactions on Networking*, 2024.

[34] Xuhong Li, Haoyi Xiong, Xingjian Li, Xuanyu Wu, Xiao Zhang, Ji Liu, Jiang Bian, and Dejing Dou. Interpretable deep learning: Interpretation, interpretability, trustworthiness, and beyond. *Knowledge and Information Systems*, 64(12), 2022.

[35] Xinjie Lin, Gang Xiong, Gaopeng Gou, Zhen Li, Junzheng Shi, and Jing Yu. Et-bert: A contextualized datagram representation with pre-training transformers for encrypted traffic classification. In *Proc. WWW*, 2022.

[36] Sebastian Lins, Konstantin D Pandl, Heiner Teigeler, Scott Thiebes, Calvin Bayer, and Ali Sunyaev. Artificial intelligence as a service: classification and research directions. *Business & Information Systems Engineering*, 63(4), 2021.

[37] Ya Liu, Xiao Wang, Bo Qu, and Fengyu Zhao. Atvitsc: A novel encrypted traffic classification method based on deep learning. *IEEE Transactions on Information Forensics and Security*, 2024.

[38] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[39] Mohammad Lotfollahi, Mahdi Jafari Siavoshani, Ramin Shirali Hossein Zade, and Mohammdsadegh Saberian. Deep packet: A novel approach for encrypted traffic classification using deep learning. *Soft Computing*, 24(3), 2020.

[40] Minzhao Lyu, Hassan Habibi Gharakheili, and Vijay Sivaraman. A survey on enterprise network security: Asset behavioral monitoring and distributed attack detection. *IEEE Access*, 12, 2024.

[41] Xiaobo Ma, Jian Qu, Feitong Chen, Wenmao Liu, Jianfeng Li, Jing Tao, Hongshan Jiao, Mawei Shi, and Zhi-Li Zhang. One host with so many ips! on the security implications of dynamic virtual private servers. *IEEE Communications Magazine*, 59(2), 2021.

[42] Xiaobo Ma, Jian Qu, Jianfeng Li, John CS Lui, Zhenhua Li, and Xiaohong Guan. Pinpointing hidden iot devices via spatial-temporal traffic fingerprinting. In *Proc. IEEE INFOCOM*, 2020.

[43] Xiaobo Ma, Jian Qu, Jianfeng Li, John CS Lui, Zhenhua Li, Wenmao Liu, and Xiaohong Guan. Inferring hidden iot devices and user interactions via spatial-temporal traffic fingerprinting. *IEEE/ACM Transactions on Networking*, 30(1), 2021.

[44] Xiaobo Ma, Jian Qu, Mawei Shi, Bingyu An, Jianfeng Li, Xiapu Luo, Junjie Zhang, Zhenhua Li, and Xiaohong Guan. Website fingerprinting on encrypted proxies: A flow-context-aware approach and countermeasures. *IEEE/ACM Transactions on Networking*, 32(3), 2023.

[45] Kaleb McDowell, Ellen Novoseller, Anna Madison, Vinicius G Goecks, and Christopher Kelshaw. Re-envisioning command and control. In *Proc. ICMCIS*, 2024.

[46] Timothy Mcintosh, Teo Susnjak, Tong Liu, Dan Xu, Paul Watters, Dongwei Liu, Yaqi Hao, Alex Ng, and Malka Halgamuge. Ransomware reloaded: Re-examining its trend, research and mitigation in the era of data exfiltration. *ACM Computing Surveys*, 57(1), 2024.

[47] Xuying Meng, Chungang Lin, Yequan Wang, and Yujun Zhang. Netgpt: Generative pretrained transformer for network traffic. *arXiv preprint arXiv:2304.09513*, 2023.

[48] Markus Miettinen, Samuel Marchal, Ibbad Hafeez, Nadarajah Asokan, Ahmad-Reza Sadeghi, and Sasu Tarkoma. Iot sentinel: Automated device-type identification for security enforcement in iot. In *Proc. IEEE ICDCS*, 2017.

[49] Seyed Mohammad Mehdi Mirnajafizadeh, Ashwin Raam Sethuram, David Mohaisen, DaeHun Nyang, and Rhongho Jang. Enhancing network attack detection with distributed and {In-Network} data collection system. In *Proc. USENIX Security*, 2024.

[50] Noor Hazlina Abdul Mutalib, Aznul Qalid Md Sabri, Ainuddin Wahid Abdul Wahab, Erma Rahayu Mohd Faizal Abdullah, and Nouar AlDahoul. Explainable deep learning approach for advanced persistent threats (apts) detection in cybersecurity: A review. *Artificial Intelligence Review*, 57(11), 2024.

[51] Zulu Okonkwo, Ernest Foo, Zhe Hou, Qinyi Li, and Zahra Jadidi. Encrypted network traffic classification with higher order graph neural network. In *Proc. ACISP*, 2023.

[52] Bo Pang, Yongquan Fu, Siyuan Ren, Ye Wang, Qing Liao, and Yan Jia. Cgnn: traffic classification with graph neural network. *arXiv preprint arXiv:2110.09726*, 2021.

[53] Bo Peng, Eric Alcaide, Quentin Anthony, Alon Albalak, Samuel Arcadinho, Stella Biderman, Huanqi Cao, Xin Cheng, Michael Chung, Matteo Grella, et al. Rwkv: Reinventing rnns for the transformer era. *arXiv preprint arXiv:2305.13048*, 2023.

[54] Matineh Pooshideh, Amin Beheshti, Yuankai Qi, Helia Farhood, Mike Simpson, Nick Gatland, and Mehdi Soltany. Presentation attack detection: a systematic literature review. *ACM Computing Surveys*, 57(1), 2024.

[55] Yuhang Qiu, Honghui Chen, Xingbo Dong, Zheng Lin, Iman Yi Liao, Massimo Tistarelli, and Zhe Jin. Ifvit: Interpretable fixed-length representation for fingerprint matching via vision transformer. *IEEE Transactions on Information Forensics and Security*, 2024.

[56] Jian Qu, Xiaobo Ma, Jianfeng Li, Xiapu Luo, Lei Xue, Junjie Zhang, Zhenhua Li, Li Feng, and Xiaohong Guan. An input-agnostic hierarchical deep learning framework for traffic fingerprinting. In *Proc. USENIX Security*, 2023.

[57] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140), 2020.

[58] Zeyu Ren, Shuihua Wang, and Yudong Zhang. Weakly supervised machine learning. *CAAI Transactions on Intelligence Technology*, 8(3), 2023.

[59] Cynthia Rudin, Chaofan Chen, Zhi Chen, Haiyang Huang, Lesia Semenova, and Chudi Zhong. Interpretable machine learning: Fundamental principles and 10 grand challenges. *Statistic Surveys*, 16, 2022.

[60] Bushra Sabir, Faheem Ullah, M Ali Babar, and Raj Gaire. Machine learning for detecting data exfiltration: A review. *ACM Computing Surveys (CSUR)*, 54(3), 2021.

[61] Mozamel M Saeed. A real-time adaptive network intrusion detection for streaming data: a hybrid approach. *Neural Computing and Applications*, 34(8), 2022.

[62] Sajal Saha, Anwar Haque, and Greg Sidebottom. Analyzing the impact of outlier data points on multi-step internet traffic prediction using deep sequence models. *IEEE Transactions on Network and Service Management*, 20(2), 2023.

[63] Yam Sharon, David Berend, Yang Liu, Asaf Shabtai, and Yuval Elovici. Tantra: Timing-based adversarial network traffic reshaping attack. *IEEE Transactions on Information Forensics and Security*, 17, 2022.

[64] Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. Self-attention with relative position representations. *arXiv preprint arXiv:1803.02155*, 2018.

[65] Meng Shen, Kexin Ji, Zhenbo Gao, Qi Li, Liehuang Zhu, and Ke Xu. Subverting website fingerprinting defenses with robust traffic representation. In *Proc. USENIX Security*, 2023.

[66] Meng Shen, Jinhe Wu, Ke Ye, Ke Xu, Gang Xiong, and Liehuang Zhu. Robust detection of malicious encrypted traffic via contrastive learning. *IEEE Transactions on Information Forensics and Security*, 2025.

[67] Meng Shen, Ke Ye, Xingtong Liu, Liehuang Zhu, Jiawen Kang, Shui Yu, Qi Li, and Ke Xu. Machine learning-powered encrypted network traffic analysis: A comprehensive survey. *IEEE Communications Surveys & Tutorials*, 25(1), 2022.

[68] Chuan Sheng, Wei Zhou, Qing-Long Han, Wanlun Ma, Xiaogang Zhu, Sheng Wen, and Yang Xiang. Network traffic fingerprinting for iiot device identification: A survey. *IEEE Transactions on Industrial Informatics*, 2025.

[69] Jiyoung Song, Hyeri Choi, Seung Kwon Koh, Dohyun Park, James Yu, Habin Kang, Youngtaek Kim, Duck Cho, and Noo Li Jeon. High-throughput 3d in vitro tumor vasculature model for real-time monitoring of immune cell infiltration and cytotoxicity. *Frontiers in Immunology*, 12, 2021.

[70] Chih-Li Sung and Rui Tuo. A review on computer model calibration. *Wiley Interdisciplinary Reviews: Computational Statistics*, 16(1), 2024.

[71] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[72] Qineng Wang, Chen Qian, Xiaochang Li, Ziyu Yao, and Huajie Shao. Lens: A foundation model for network traffic. *arXiv preprint arXiv:2402.03646*, 2024.

[73] Xu Wang, Zimu Zhou, Fu Xiao, Kai Xing, Zheng Yang, Yunhao Liu, and Chunyi Peng. Spatio-temporal analysis and prediction of cellular traffic in metropolis. *IEEE Transactions on Mobile Computing*, 18(9), 2018.

[74] Nigel Williams, Sebastian Zander, and Grenville Armitage. A preliminary performance comparison of five machine learning algorithms for practical ip traffic flow classification. *ACM SIGCOMM Computer Communication Review*, 36(5), 2006.

[75] Xi Xiao, Xiang Zhou, Zhenyu Yang, Le Yu, Bin Zhang, Qixu Liu, and Xiapu Luo. A comprehensive analysis of website fingerprinting defenses on tor. *Computers & Security*, 136, 2024.

[76] Wenhan Xiong, Barlas Oğuz, Anchit Gupta, Xilun Chen, Diana Liskovich, Omer Levy, Wen-tau Yih, and Yashar Mehdad. Simple local attentions remain competitive for long-context tasks. *arXiv preprint arXiv:2112.07210*, 2021.

[77] Mengfan Xue, Shishen Jia, Ling Chen, Hailiang Huang, Lijuan Yu, and Wentao Zhu. Ct-based copd identification using multiple instance learning with two-stage attention. *Computer Methods and Programs in Biomedicine*, 230, 2023.

[78] Limin Yang, Zhi Chen, Chenkai Wang, Zhenning Zhang, Sushruth Booma, Phuong Cao, Constantin Adam, Alexander Withers, Zbigniew Kalbarczyk, Ravishankar K Iyer, et al. True attacks, attack attempts, or benign triggers? an empirical measurement of network alerts in a security operations center. In *Proc. USENIX Security*, 2024.

[79] Haozhen Zhang, Le Yu, Xi Xiao, Qing Li, Francesco Mercaldo, Xiapu Luo, and Qixu Liu. Tfe-gnn: A temporal fusion encoder using graph neural networks for fine-grained encrypted traffic classification. In *Proc. WWW*, 2023.

[80] Jialong Zhang, Frederico Araujo, Teryl Taylor, and Marc Philippe Stoecklin. Detecting adversarial attacks through decoy training, 2022. US Patent 11,501,156.

[81] Jialong Zhang, Zhongshu Gu, Jiyong Jang, Dhilung Kirat, Marc Stoecklin, Xiaokui Shu, and Heqing Huang. Scarecrow: Deactivating evasive malware via its own evasive logic. In *Proc. DSN*, 2020.

[82] Jialong Zhang, Zhongshu Gu, Jiyong Jang, Hui Wu, Marc Ph Stoecklin, Heqing Huang, and Ian Molloy. Protecting intellectual property of deep neural networks with watermarking. In *Proc. AsiaCCS*, 2018.

[83] Jialong Zhang, Jiyong Jang, Guofei Gu, Marc Ph Stoecklin, and Xin Hu. Error-sensor: mining information from http error traffic for malware intelligence. In *Proc. RAID*, 2018.

[84] Yuchen Zhang, Zeyu Gao, Kai He, Chen Li, and Rui Mao. From patches to wsis: A systematic review of deep multiple instance learning in computational pathology. *Information Fusion*, 2025.

[85] Yuchen Zhang, Zhen Lu, Jianglin Zhou, Yi Sun, Wuci Yi, Juan Wang, Tianjing Du, Dongning Li, Xinyan Zhao, Yifei Xu, et al. Cdsnet: An automated method for assessing growth stages from various anatomical regions in lateral cephalograms based on deep learning. *Journal of the World Federation of Orthodontists*, 2024.

[86] Ruijie Zhao, Mingwei Zhan, Xianwen Deng, Yanhao Wang, Yijun Wang, Guan Gui, and Zhi Xue. Yet another traffic classifier: A masked autoencoder based traffic transformer with multi-level flow representation. In *Proc. AAAI*, volume 37, 2023.

[87] Guangmeng Zhou, Xiongwen Guo, Zhuotao Liu, Tong Li, Qi Li, and Ke Xu. Trafficformer: an efficient pretrained model for traffic data. In *Proc. IEEE S&P*, 2025.

[88] Haoyi Zhou, Shanghang Zhang, Jieqi Peng, Shuai Zhang, Jianxin Li, Hui Xiong, and Wancai Zhang. Informer: Beyond efficient transformer for long sequence time-series forecasting. In *Proc. AAAI*, volume 35, 2021.

[89] Peng Zhou, Wei Shi, Jun Tian, Zhenyu Qi, Bingchen Li, Hongwei Hao, and Bo Xu. Attention-based bidirectional long short-term memory networks for relation classification. In *Proc. ACL*, 2016.

[90] Hengye Zhu, Mengxiang Liu, Binbin Chen, Xin Che, Peng Cheng, and Ruilong Deng. Honeyjudge: A plc honeypot identification framework based on device memory testing. *IEEE Transactions on Information Forensics and Security*, 19, 2024.

[91] Denis Zuev and Andrew W Moore. Traffic classification using a statistical approach. In *Proc. PAM*, 2005.

A Appendix

A.1 Attention Mechanism in `Tracegram`

The input to an attention mechanism typically consists of three components: a query vector, a set of key vectors, and a set of value vectors. The core idea involves computing attention weights by comparing the query with a series of keys. These weights are subsequently used to perform a weighted sum of

the corresponding value vectors to produce the final output. The general form is summarized in Equation (3)

$$\text{Attention}(Query, Key, Value) = \sum_{i=1}^{L_x} a_i Value_i, \tag{3}$$

where L_x is the number of value vectors, and a_i represents the similarity between the query and the i-th key:

$$a_i = \text{Similarity}(Query, Key_i) \tag{4}$$

To apply this mechanism to traffic analysis, we adapt the scaled dot-product self-attention mechanism proposed by Vaswani et al. [71]. Its specific form is as follows:

$$\begin{cases} \text{Attention} = \text{Softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V, \\ Q = XW^Q, \\ K = XW^K, \\ V = XW^V. \end{cases} \tag{5}$$

Here, X is a matrix of input vectors (CPVs). For example, assume a flow is represented by 10 CPVs after compression, and each CPV has a dimension of 100. The input matrix would then be $X \in \mathbb{R}^{10\times 100}$. W^Q, W^K, and W^V are three trainable parameter matrices learned during the training process. Q, K, and V correspond to the *Query*, *Key*, and *Value* in Equation (3), respectively. The scaling factor d_k (the dimension of the key vectors) is used for gradient stabilization.

The key difference between our method and standard self-attention is that the query Q in our approach is a trainable constant vector that is independent of the input X. This crucial modification significantly reduces the computational complexity from $O(n^2)$ in standard self-attention to $O(n)$, where n is the number of rows in the input matrix X (i.e., the number of Compressed Packet Vectors, CPVs).

A.2 Detailed Theoretical Proofs

This appendix details the mathematical derivation of the equivalence between our hierarchical objective and the optimal end-to-end objective under the Orthogonality Criterion.

Orthogonality Criterion. We define a stricter probabilistic form of the Instance Separability Assumption. Given a class label Y_B, the joint posterior probability of the flows $F_1, \ldots, F_n$ in a trace B factorizes as:

$$P(F_1, \ldots, F_n | Y_B; \theta_{enc}) = \prod_{j=1}^{n} P(F_j | Y_B; \theta_{enc}). \tag{6}$$

This criterion implies that, conditioned on the label, the information contributed by each flow is independent.

Proof of Equivalence. Under this criterion, the optimization objective of a theoretically optimal end-to-end model (maximizing $P(B|Y_B)$) can be rewritten as:

$$\hat{\theta} = \arg\max_{\theta} \prod_{(B,Y_B)} \left(\prod_{j=1}^{n} P(F_j | Y_B; \theta) \right). \tag{7}$$

This is mathematically equivalent to our stage-one optimization objective for the Flow Encoder (M1):

$$\hat{\theta}_{enc} = \arg\max_{\theta_{enc}} \prod_{(B,Y_B)} \prod_{j=1}^{n} P(F_j | Y_B; \theta_{enc}). \tag{8}$$

Thus, under the ideal Orthogonality Criterion, optimizing the decoupled encoder is equivalent to optimizing the end-to-end objective.

A.3 Dataset Characteristics

Table 9 shows dataset characteristics, including the number of classes, dataset sizes (i.e., number of traces), etc.

A.4 Batch Normalization used in `Tracegram`

Batch Normalization is a technique widely used in deep neural networks to accelerate training and improve model stability [23]. Its core idea is to normalize the data within each mini-batch. The computation process is as follows:

$$\begin{cases} \mu_B \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i, \\ \sigma_B^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_B)^2, \\ \hat{x}_i \leftarrow \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \varepsilon}}, \\ y_i \leftarrow \gamma \hat{x}_i + \beta. \end{cases} \tag{9}$$

The method operates on the m inputs x_i within a mini-batch by first computing their mean μ_B and variance σ_B^2. Each input x_i is then normalized to $\hat{x}_i$ by subtracting the mean and dividing by std; a small constant ε is added to the denominator for numerical stability. Finally, a linear scale-and-shift transformation is applied to the normalized $\hat{x}_i$ using learnable parameters, γ and β. This last step is crucial as it allows the model to learn to restore the original input's feature distribution, thereby preserving the network's expressive power.

However, Batch Normalization has limitations when processing variable-length sequence inputs. This is because computing consistent mean and variance across sequences of varying lengths is difficult. Considering this factor, our model uses Batch Normalization only in the final fully-connected layers handling fixed-size vectors. For variable-length flow vectors, we instead adopt Layer Normalization [3].

A.5 Perturbation method

To rigorously evaluate the robustness of our framework against real-world network noise and potential adversarial manipulations, we designed a comprehensive perturbation method. Rather than applying a single type of noise, we probabilistically mix five types of perturbations to create a more challenging and realistic testbed. This approach is controlled by a single parameter, the *noise ratio*, which ranges from 0.00% to 50.00% in our experiments.

Table 9: The characteristics of the datasets used in our experiments.

Dataset	Classes	Traces	Number of flows per trace				Flow length (packets)			
			Average	Minimum	Maximum	Std. Dev.	Average	Minimum	Maximum	Std. Dev.
UAV	4	3,705	5.02	0	266	16.57	554.13	1	50,332	2984.48
KWS	50	5,000	31.30	6	68	8.29	36.39	1	8,378	180.74
IDI	27	550	27.34	1	215	44.58	12.20	1	297	25.75
ISD	2	4,402	10.47	0	578	22.7	166.42	1	2,252,416	12,831.47

For a given trace in the dataset and a specified *noise ratio* p, we randomly perturb based on this probability. Specifically, for each potential target of perturbation (e.g., a packet or a flow), we first determine whether to apply noise with probability p. If the decision is to add noise, we then randomly select one of the following five perturbation techniques to apply. This ensures the injected noise is diverse and unpredictable. The five base perturbation methods are as follows:

- **Packet Loss:** We randomly drop individual packets from a flow. For a given flow, each packet independently has a probability p of being removed during transmission.
- **Flow Loss:** We randomly drop entire flows from a trace. Each flow within a trace independently has a probability p of being entirely removed during transmission.
- **Packet Padding:** We randomly select packets with probability p and pad them to a fixed maximum length. This alters the size-based features of the traffic.
- **Dummy Packets:** We randomly select packets with probability p and insert a dummy packet immediately after them. The dummy packet copies a randomly chosen packet, simulating background traffic at the packet level.
- **Dummy Flows:** We randomly select flows with probability p and insert a dummy flow immediately after them. The dummy flow copies a randomly chosen flow, simulating background traffic at the flow level.

By combining these methods under a unified *noise ratio*, we can systematically increase the level of disruption to the original data, allowing us to precisely measure the performance degradation and thereby assess the model's true robustness.

A.6 Baseline Methods

To provide a comprehensive performance comparison, we benchmark `Tracegram` against several SOTA methods. These baselines cover both classic machine learning approaches based on feature engineering and advanced deep learning models.

- **ET-BERT** [35]: A pioneering pre-training framework that adapts the BERT architecture to encrypted traffic classification. It employs a Datagram2Token representation that converts raw packet bytes into bi-gram tokens. ET-BERT is pre-trained on large-scale unlabeled traffic using two self-supervised tasks: the *Masked BURST Model (MBM)* to capture byte-level context within a burst, and the *Same-origin BURST Prediction (SBP)* to learn flow-level transmission structures.
- **TrafficFormer** [87]: A recent SOTA pre-trained model designed for efficient traffic analysis. It improves upon prior work by introducing a fine-grained pre-training task called *Same Origin-Direction-Flow (SODF)*, which explicitly models the direction and order of packets within a flow. Additionally, TrafficFormer implements a *Random Initialization Field Augmentation (RIFA)* strategy during fine-tuning to mitigate the impact of randomized header fields (e.g., TCP sequence numbers).
- **K-Means + RF** [29]: A three-layer system that combines unsupervised and supervised learning for classifying user activities. It first uses K-Means for unsupervised trace clustering and then employs a Random Forest (RF) classifier on the resulting clusters.
- **Hybrid Attention** [56]: An advanced trace-based classification model and serves as our primary deep learning baseline. It introduces an input-agnostic hierarchical framework that can abstract heterogeneous traffic features (from packets to flows to traces) into homogeneous vectors, making them suitable for standard neural network classifiers.
- **RF** [15]: A handcrafted feature-based approach relies on handcrafted features. It extracts a wide range of behavioral and flow-based features from packet headers and payloads (Ethernet, IP, TCP, and UDP) and uses a Random Forest (RF) model for classification.
- **CCR-ELM** [17]: Combines a state transition matrix with a Class-specific Cost Regulation Extreme Learning Machine. It first extracts features to build a state transition matrix for each flow and then uses the CCR-ELM for the final classification.