NetMamba: Efficient Network Traffic Classification via Pre-training Unidirectional Mamba

Tongze Wang[1], Xiaohui Xie[2*] , Wenduo Wang[2], Chuyi Wang[2], Youjian Zhao[2], Yong Cui[2*]
[1]*Institute for Network Sciences and Cyberspace, Tsinghua University*
[2]*Department of Computer Science and Technology, Tsinghua University*

***Abstract*—Network traffic classification is a crucial research area aiming to enhance service quality, streamline network management, and bolster cybersecurity. To address the growing complexity of transmission encryption techniques, various machine learning and deep learning methods have been proposed. However, existing approaches face two main challenges. Firstly, they struggle with model inefficiency due to the quadratic complexity of the widely used Transformer architecture. Secondly, they suffer from inadequate traffic representation because of discarding important byte information while retaining unwanted biases. To address these challenges, we propose NetMamba, an efficient linear-time state space model equipped with a comprehensive traffic representation scheme. We adopt a specially selected and improved unidirectional Mamba architecture for the networking field, instead of the Transformer, to address efficiency issues. In addition, we design a traffic representation scheme to extract valid information from massive traffic data while removing biased information. Evaluation experiments on six public datasets encompassing three main classification tasks showcase NetMamba's superior classification performance compared to state-of-the-art baselines. It achieves an accuracy rate of nearly 99% (some over 99%) in all tasks. Additionally, NetMamba demonstrates excellent efficiency, improving inference speed by up to 60 times while maintaining comparably low memory usage. Furthermore, NetMamba exhibits superior few-shot learning abilities, achieving better classification performance with fewer labeled data. To the best of our knowledge, NetMamba is the first model to tailor the Mamba architecture for networking.**

***Index Terms*—NetMamba, Traffic Classification, Pre-training**

I. Introduction

Network traffic classification, which aims to identify potential threats within traffic or classify the category of traffic originating from different applications or services, has become an increasingly vital research area. This is crucial for ensuring cybersecurity, improving service quality and user experience, and enabling efficient network management. However, the widespread adoption of encryption techniques (e.g., TLS) and anonymous network technologies (e.g., VPN, Tor) has made the accurate analysis of complex traffic more challenging.

Researchers have proposed numerous approaches to address this issue, showing promising results yet facing severe limitations. Conventional machine learning methods [1]–[3], primarily relying on manually engineered features or statistical attributes, often fail to capture accurate traffic representations due to the absence of raw traffic data. In contrast, deep learning approaches [4]–[6] automatically extract features from raw byte-level data, leading to enhanced traffic classification capabilities. Nonetheless, these deep learning methods necessitate extensive labeled datasets, rendering the models susceptible to biases and impeding their adaptability to novel data distributions.

Recently, pre-training has emerged as a prevalent model training paradigm in natural language processing (NLP) [7] and computer vision (CV) [8]. Motivated by this trend, several Transformer-based pre-trained traffic models [9]–[11] have been developed to learn generic traffic representations from extensive unlabeled data and then fine-tune for specific downstream tasks using limited labeled traffic data. However, these existing models face two significant challenges: 1) Limited Model Efficiency: state-of-the-art methods in traffic analysis primarily use Transformer architecture, which employs a quadratic self-attention mechanism to calculate correlations within a sequence. This leads to substantial computational and memory costs on long sequences [12], [13]. Consequently, these models are unsuitable for real-time online traffic classification and cannot operate efficiently with the limited resources of typical network devices. 2) Inadequate Traffic Representation: current methodologies fail to adequately and accurately represent raw traffic data due to discarding crucial byte information and preserving unwanted biases. As a result, these unreliable schemes impair classification performance or even cause model failure in complex traffic scenarios.

To address these challenges, we propose NetMamba, an efficient linear-time state space model equipped with a comprehensive traffic representation scheme, aiming to accurately perform network traffic classification tasks with higher inference speed and lower memory usage.

To improve model efficiency, we use the Mamba architecture for the model backbone instead of the Transformer. Mamba [14], a liner-time state space model for sequence modeling, has achieved notable success across various domains, including natural language processing [15], computer vision [12] and graph understanding [16]. This suggests promising potential for applying Mamba to the network domain. However, adapting Mamba for efficient and robust network traffic analysis requires selecting the appropriate architecture from the existing heterogeneous and complex Mamba variants. By carefully testing different variants of Mamba, we found that the original unidirectional Mamba [14], without omnidirec-

This work is supported by the NSFC Project under Grant 62132009, Grant 62221003 and Grant 62394322.

* Corresponding Authors: Xiaohui Xie and Yong Cui

979-8-3503-5171-2/24/$31.00 ©2024 IEEE

tional scans or redundant blocks, is well-suited for efficiently learning latent patterns within sequential network traffic. To further enhance the model's performance and robustness, we incorporate positional embeddings and pre-training strategies specially designed for networking.

To enhance traffic representation, we design a more comprehensive and reliable scheme. This scheme retains valuable packet content within both headers and payloads while eliminating unwanted biases through various methods, including packet anonymizing, byte allocation balancing and stride-based data cutting, thereby improving traffic classification capabilities.

Specifically, NetMamba initially extracts hierarchical flow information from raw traffic and converts it into a stride sequence, which serves as the model's input. Subsequently, NetMamba undergoes self-supervised pre-training on large unlabeled datasets using a masked autoencoder structure, which is designed to learn generic representations of traffic data through reconstructing masked strides. Finally, the decoder is replaced with a multi-layer perceptron head, and NetMamba is fine-tuned on limited labeled data to refine traffic representations and adapt to downstream traffic classification tasks. Extensive experiments conducted on publicly available datasets demonstrate the effectiveness and efficiency of NetMamba. In all classification tasks, NetMamba achieves an accuracy rate of nearly 99% (some over 99%), and, compared to existing baselines, it improves inference speed by up to 60 times while maintaining low GPU memory usage. Furthermore, NetMamba exhibits superior few-shot learning capabilities in comparison to other pre-training models, achieving better performance with fewer labeled data.

In summary, our work makes the following contributions:

(1) We propose NetMamba, the first state space model specifically designed for network traffic classification. Compared to existing Transformer-based methods, NetMamba demonstrates superior performance and inference efficiency.
(2) We develop a comprehensive representation scheme for network traffic data that preserves valuable traffic characteristics while eliminating unwanted biases.
(3) We conduct extensive experiments across a range of traffic classification tasks. An overall comparison, along with detailed evaluations—encompassing ablation studies, efficiency analyses, and few-shot learning investigations—is provided. These insights could illuminate paths for future research. Additionally, the code of NetMamba is publicly available [1].

II. Related Work

A. Transformer-based Traffic Classification

Due to its highly parallel architecture and robust sequence modeling abilities, Transformer has gained significant popularity and is extensively used for traffic understanding and generation tasks. For instance, MTT [17] employs a multi-task Transformer trained on truncated packet byte sequences to analyze traffic features in a supervised way. Recognizing the challenges associated with data annotation, MT-FlowFormer [18] introduces a Transformer-based semi-supervised framework for data augmentation and model improvement.

To leverage unlabeled data effectively, several pre-trained models have been proposed. Inspired by BERT's pre-training methodology in natural language processing, PERT [19] and ET-BERT [9] process raw traffic bytes using tokenization, apply masked language modeling to learn traffic representations, and fine-tune the models for downstream tasks. Similarly, YaTC [10] and FlowMAE [20] adopt the widely-used MAE pre-training approach from computer vision, which involves patch splitting for byte matrices, capturing traffic correlations through masked patch reconstruction, and subsequent fine-tuning.

Given the global interest in large language models, pre-trained traffic foundation models such as NetGPT [21] and Lens [11] have been developed to address traffic analysis and generation simultaneously. However, Transformer-based models face computational and memory inefficiencies because of the quadratic complexity of their core self-attention mechanism. This necessitates a more efficient and effective solution for online traffic classification.

B. Mamba-based Representation Learning

Representation learning is a branch of machine learning concerned with automatically learning and extracting meaningful representations or features from raw data. Since the advent of Mamba, an efficient and effective sequence model, numerous Mamba variants have emerged to enhance representation learning across diverse domain-specific data formats. For instance, in the realm of vision tasks requiring spatial awareness, custom-designed scan architectures like Vim [12] and VMamba [22] have been developed. In the domain of language modeling, DenseMamba [15] improves upon the original SSM by incorporating dense internal connections to boost performance. Handling graph data necessitates specialized solutions such as Graph-Mamba [16] and STG-Mamba [23], each employing tailored graph-specific selection mechanisms. Furthermore, various Mamba variants have proven effective in domains like signal processing [24], point cloud analysis [25], and multi-modal learning [26].

However, to date, there are no reports of Mamba's successful application in network traffic classification, highlighting the need for our research in this area.

C. Traffic Representation Schemes

In real-world scenarios, massive raw network traffic encompasses a wide range of data categories that vary in upper applications, carried protocols, or transmission purposes. Therefore, a robust representation scheme with appropriate granularity is crucial for accurate traffic understanding.

Traditional machine learning methods [1]–[3], [27], [28], constrained by limited model parameters and fitting capabilities, commonly resort to utilizing compressed statistical

[1]https://github.com/wangtz19/NetMamba

features at the packet or flow level, such as distributions of packet sizes or inter-arrival times. However, these features often suffer from excessive compression, resulting in the loss of vital information inherent in raw datagrams.

Recent advancements in deep learning have endeavored to utilize raw traffic bytes. However, as shown in Table I, these methods face limitations. They often neglect crucial information in packet headers and introduce unwanted biases by retaining biased IP addresses, ignoring byte balance, or using improper data-splitting techniques.

To address these issues, we propose a novel network traffic representation scheme. Our approach remedies the aforementioned shortcomings, preserving hierarchical traffic information while effectively eliminating biases.

TABLE I
COMPARISON OF EXISTING REPRESENTATION SCHEMES

Method	Header	Payload	IA[1]	BB[2]	Splitting
PERT [19]	✗	✓	✗	✗	token
ET-BERT [9]	✗	✓	✗	✗	token
YaTC [10]	✓	✓	✗	✓	patch
FlowMAE [20]	✓	✓	✓	✗	patch
NetGPT [21]	✓	✓	✗	✗	token
Lens [11]	✓	✓	✓	✗	token
NetMamba	✓	✓	✓	✓	stride

[1] IA: IP Anonymizing removes all IP addresses.
[2] BB: Byte Balance sets fixed sizes for headers and payloads.

III. PRELIMINARIES

This section elaborates on basic definitions, terminologies, and components underlining the Mamba block which serves as the foundation of the proposed NetMamba.

A. State Space Models

As the key components of Mamba, State Space Models (SSMs) represent a contemporary category of sequence models within deep learning that share broad connections with Recurrent Neural Networks (RNNs) and Convolutional Neural Networks (CNNs). Drawing inspiration from continuous systems, SSMs are commonly structured as linear Ordinary Differential Equations (ODEs) which establish a mapping from an input sequence $x(t) \in \mathbb{R}^N$ to an output sequence $y(t) \in \mathbb{R}^N$ via an intermediate latent state $h(t) \in \mathbb{R}^N$:

$$\begin{aligned} h^{'}(t) &= \mathbf{A}h(t) + \mathbf{B}x(t) \\ y(t) &= \mathbf{C}h(t) \end{aligned} \tag{1}$$

where $\mathbf{A} \in \mathbb{R}^{N\times N}$ represents the evolution parameter, while $\mathbf{B} \in \mathbb{R}^{N\times 1}$ and $\mathbf{C} \in \mathbb{R}^{1\times N}$ are the projection parameters.

B. Discretization

Integrating raw SSMs with deep learning presents a significant challenge due to the discrete nature of typical real-world data, contrasting with the continuous-time characteristic of SSMs. To overcome this challenge, the zero-order hold (ZOH) technique is utilized for discretization, leading to the discrete version formulated as follows:

$$\begin{aligned} h_t &= \overline{\mathbf{A}}h_{t-1} + \overline{\mathbf{B}}x_t \\ y_t &= \mathbf{C}h_t \end{aligned} \tag{2}$$

where $\overline{\mathbf{A}} = \exp(\Delta\mathbf{A})$ and $\overline{\mathbf{B}} \approx \Delta\mathbf{B}$ represent the discretized parameters, with Δ denoting the discretization step size. This recurrent formulation is characterized by linear time complexity.

C. Selection Mechanism

While designed for sequence modeling, SSMs exhibit subpar performance when content-aware reasoning is required, primarily due to their time-invariant nature. Specifically, the parameters $\overline{\mathbf{A}}$, $\overline{\mathbf{B}}$, and $\mathbf{C}$ remain constant across all input tokens within a sequence. To address this issue, Mamba [14] introduces the selection mechanism, enabling the model to select pertinent information from the context dynamically. This adaptation involves transforming the SSM parameters $\overline{\mathbf{B}}$, $\mathbf{C}$, and Δ into functions of the input x. Additionally, a GPU-friendly implementation is devised to facilitate efficient computation of the selection mechanism, leading to a notable reduction in memory I/O operations and eliminating the need to store intermediate states.

IV. NETMAMBA FRAMEWORK

This section overviews the framework of NetMamba (see Figure 1), providing a comprehensive blueprint for the detailed design presented in § V and § VI. Initially, NetMamba extracts hierarchical information from raw binary traffic and converts it into stride-based representation. Inspired by the Masked AutoEncoders (MAE) pre-training model in computer vision, NetMamba employs a dual-stage training approach. Specifically, self-supervised pre-training is utilized to acquire traffic representation, while supervised fine-tuning is employed to tailor the model for downstream traffic understanding tasks.

1) Traffic Representation Phase: To enhance domain knowledge within networks, NetMamba adopts a stride-based methodology to represent key content within network traffic. Initially, traffic data is segmented into distinct flows, categorized by their 5-tuple attributes: Source IP, Destination IP, Source Port, Destination Port, and Protocol. Fixed-sized segments of header and payload bytes are then extracted for each packet within a flow. To collect more comprehensive traffic information without compromising model efficiency due to excessively long packet sequences, we follow approaches outlined in prior studies [9], [10], which involve selectively utilizing specific packets within a flow. Specifically, bytes from the initial packets of each flow are aggregated into a unified byte array, integrating information across byte, packet, and flow levels for a comprehensive view of traffic characteristics.

This byte array forms the foundation for segmenting non-overlapping flow strides. It preserves semantic relationships between adjacent bytes, effectively mitigating biases introduced by conventional patch-splitting methods, as well as addressing out-of-vocabulary issues commonly associated with

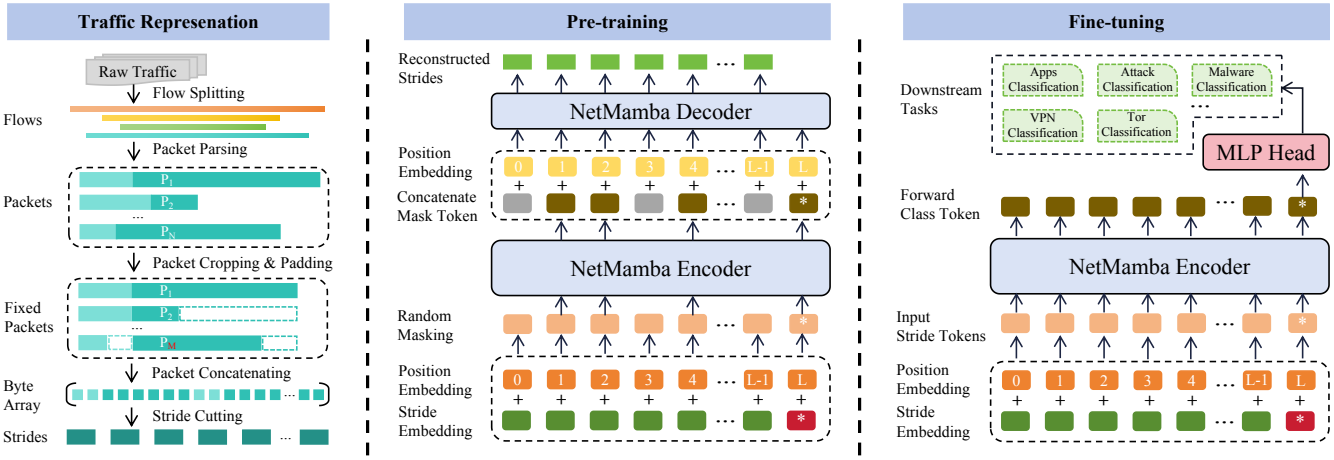

Fig. 1. Overview of NetMamba Framework

tokenization processes. Further design intricacies regarding traffic representation are elucidated in § V.

2) Pre-training Phase: To acquire generic encodings of network domain knowledge based on flow stride representations, NetMamba undergoes pre-training using extensive unlabeled network traffic data. Specifically, NetMamba utilizes a masked autoencoder (MAE) architecture, incorporating multiple unidirectional Mamba blocks in both its encoder and decoder, as detailed in § VI-A2.

During pre-training, flow strides undergo several sequential steps: concatenation with a trailing class token, mapping into stride embeddings, addition of positional embeddings, and random masking. The encoder focuses solely on visible strides, grasping inherent relationships and generating an output traffic representation. The decoder then reconstructs the masked strides using the encoder's output and dummy tokens. Pre-training is optimized by minimizing the reconstruction loss for the masked strides, ensuring the model learns robust traffic patterns. Detailed insights into the pre-training strategy are provided in § VI-B.

3) Fine-tuning Phase: For accurately capturing traffic patterns and understanding downstream task requirements, NetMamba undergoes fine-tuning using labeled traffic data. During this phase, the decoder of NetMamba is replaced by a multi-layer perceptron (MLP) head to facilitate classification tasks. With the removal of the reconstruction task, all embedded flow strides become visible to the encoder. As the unidirectional Mamba block processes sequence information in a front-to-back manner, the trailing class token, after being processed by the encoder, aggregates the overall traffic characteristics. Subsequently, NetMamba forwards only this class token to the MLP-based classifier.

Post pre-training, NetMamba's encoder exhibits significant adaptability when fine-tuned with limited labeled data, enabling efficient transition to various downstream tasks such as application classification and attack detection. Additional details on the fine-tuning process are provided in § VI-C.

V. Traffic Representation

This section provides detailed information about the traffic representation scheme used by NetMamba. The key hyper-parameters are listed in Table II.

TABLE II
Summary of Hyper-Parameter Notations in NetMamba

Notation	Description
M	Number of packets selected from a single flow
N_h	Number of header bytes selected from a single packet
N_p	Number of payload bytes selected from a single packet
L_b	Length of the byte array for a single flow
L_s	Length of the consecutive bytes for a flow stride
N_s	Length of the stride sequence for a single flow
$\mathtt{D}_{\text{enc}}/\mathtt{D}_{\text{dec}}$	Hidden state dimension of NetMamba encoder or decoder
$\mathtt{E}_{\text{enc}}/\mathtt{E}_{\text{dec}}$	Expanded state dimension of NetMamba encoder or decoder
$\mathtt{N}$	Dimension of state space models in NetMamba
$\mathtt{B}$	Batch size of the input token sequence
$\mathtt{L}$	Length of the original input token sequence
r	Ratio of masked stride tokens
$\mathtt{L}_{\text{vis}}$	Length of the visible input token sequence

1) Flow Splitting: Formally, given network traffic comprising multiple packets, we segment it into various flows, with each flow consisting of packets that belong to a specific protocol and are transmitted between two ports on two hosts. Packets within the same flow encapsulate significant interaction information between the two hosts. This information includes the establishment of a TCP connection, data exchanged during communication, and the overall transmission status. These flow-level features are pivotal in characterizing application behaviors and enhancing the efficiency of traffic classification processes.

2) Packet Parsing: For each flow, all packets are processed through several sequential operations to preserve valuable

information and eliminate unnecessary interference. When narrowing down the scope for analyzing traffic data related to specific applications or services, we exclude all packets carried by non-IP protocols, such as Address Resolution Protocol (ARP) and Dynamic Host Configuration Protocol (DHCP). Considering the critical information contained within both the packet (e.g., the total length field) and the payload (text content for upper-level protocols), we choose to retain these elements. Furthermore, to mitigate biases introduced by identifiable information, all packets are anonymized through the removal of Ethernet headers and the masking of IP addresses.

3) Packet Cropping & Padding, and Concatenating: Given the variability in packet size within the same flow and the fluctuation in both header length (including the IP header and any upper-layer headers) and payload length within individual packets, problematic scenarios often arise. For instance, the first long packet can occupy the entire limited model input array, or excessively long payloads can dominate the byte information within shorter headers. Therefore, it is essential to standardize packet sizes by assigning uniform sizes to all packets and fixed lengths to both packet headers and payloads. Specifically, we select the first M packets from a single flow, setting the header length to N_h bytes and the payload length to N_p bytes. Any packet exceeding this length will be cropped, while shorter packets will be padded to meet these specifications.

Eventually, all bytes of initial M packets are concatenated into an unified array $[b_1, b_2, \ldots, b_{L_b}]$ where $L_b = M \times (N_h + N_p)$ represents the array length and b_i denotes the i-th byte.

4) Stride Cutting: Given the significant computational and memory demands posed by a byte array with L_b (typically greater than 1000) elements, it becomes imperative to explore further compression techniques to enhance the efficiency of model training and inference. Traditional methods often involve reshaping the byte array into a square matrix and employing two-dimensional patch splitting, a practice borrowed from computer vision. However, this technique unintentionally introduces biases by grouping vertically adjacent bytes that are semantically unrelated, as they are not naturally contiguous in the sequential traffic data.

Inspired by patching methods used in time-series forecasting, we adopt a 1-dimensional stride cutting approach on the original array, aligning with the sequential nature of network traffic and preserving inter-byte correlations. Specifically, we divide the byte array into non-overlapping strides of size $1 \times L_s$, resulting in a total number of strides $N_s = L_b/L_s$. Each stride $\mathbf{s}_i \in \mathbb{R}^{1\times L_s}$ is defined as $[b_{L_s\times i}, b_{L_s\times i+1}, \ldots, b_{L_s\times(i+1)-1}]$ for $0 \le i < N_s$. This strategy aims to mitigate biases while retaining essential sequential information in the data.

Takeaway. *Our traffic representation scheme effectively retains crucial information from both packet headers and payloads, while eliminating unwanted biases through techniques such as IP anonymization, byte balancing, and stride cutting. For a detailed evaluation, please refer to § VII-D.*

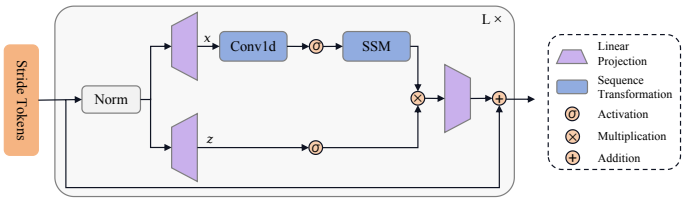

Fig. 2. NetMamba Encoder (Decoder)

VI. Model Details

This section details the NetMamba model architecture, along with the pre-training and fine-tuning strategies.

A. NetMamba Architecture

1) Stride Embedding: Given the stride array, we initially perform a linear projection on each stride $\mathbf{s}_i$ to a vector with size $\mathtt{D}_{\text{enc}}$ and incorporate position embeddings $\mathbf{E}^{\text{pos}}_{\text{enc}} \in \mathbb{R}^{N_s\times \mathtt{D}_{\text{enc}}}$ as shown below:

$$\mathbf{X}_0 = [\mathbf{s}_1\mathbf{W}; \mathbf{s}_2\mathbf{W}; \cdots; \mathbf{s}_{N_s}\mathbf{W}; \mathbf{x}_{\text{cls}}] + \mathbf{E}^{\text{pos}}_{\text{enc}} \tag{3}$$

where $\mathbf{W} \in \mathbb{R}^{L_s\times \mathtt{D}_{\text{enc}}}$ represents the learnable projection matrix. Inspired by ViT [29] and BERT [7], we introduce a class token to represent the entire stride sequence, denoted as $\mathbf{x}_{\text{cls}}$. Since the unidirectional Mamba processes sequence information from front to back, we opt to append the class token to the end of the sequence for enhanced information aggregation.

2) NetMamba Block : Recently, several variants of Mamba have been proposed to accommodate domain-specific data formats and task requirements. For instance, Vim [12] incorporates bidirectional Mamba blocks for spatial-aware understanding of vision tasks, Graph-Mamba [16] introduces a graph-dependent selection mechanism for graph learning, while MiM-ISTD [30] customizes a cascading Mamba structure for extracting hierarchical visual information. We argue that the original unidirectional Mamba design [14], tailored for sequence modeling, is well-suited for representation learning in sequential network traffic, offering increased efficiency through the elimination of omnidirectional scans and redundant blocks. We carefully test different Mamba variants, demonstrating that the selected unidirectional Mamba is more suitable for processing network traffic. Please refer to the ablation studies for more details.

Hence, we implement the NetMamba encoder and decoder using unidirectional Mamba blocks, as illustrated in Figure 2. The operational process of the NetMamba block forward pass is outlined in Algorithm 1. For a given input token sequence $\mathbf{X}_{t-1}$ with a batch size $\mathtt{B}$ and sequence length $\mathtt{L}$ from the $(t-1)$-th NetMamba block, we begin by normalizing it and then projecting it linearly into $\mathbf{x}$ and $\mathbf{z}$, both with dimension size of $\mathtt{E}$. We subsequently apply causal 1-D convolution to $\mathbf{x}$, resulting in $\mathbf{x}'$. Based on $\mathbf{x}'$, we compute the input-dependent step size $\mathbf{\Delta}$, as well as the projection parameters $\mathbf{B}$ and $\mathbf{C}$ having a dimension size of $\mathtt{N}$. We then discretize $\overline{\mathbf{A}}$ and $\overline{\mathbf{B}}$ using $\mathbf{\Delta}$. Following this, we calculate $\mathbf{y}$ employing

a hardware-aware SSM. Finally, $\mathbf{y}$ is gated by $\mathbf{z}$ and added residually to $\mathbf{X}_{t-1}$, resulting in the output token sequence $\mathbf{X}_t$ for the t-th NetMamba block.

Algorithm 1 NetMamba Block Forward Pass

Input: $\mathbf{X}_{t-1} : (\mathtt{B}, \mathtt{L}, \mathtt{D})$
Output: $\mathbf{X}_t : (\mathtt{B}, \mathtt{L}, \mathtt{D})$
$\mathbf{X}'_{t-1} : (\mathtt{B}, \mathtt{L}, \mathtt{D}) \leftarrow \mathbf{Norm}(\mathbf{X}_{t-1})$ *// normalize input sequence*
$\mathbf{x} : (\mathtt{B}, \mathtt{L}, \mathtt{E}) \leftarrow \mathbf{Linear}^{\mathbf{x}}(\mathbf{X}'_{t-1})$
$\mathbf{z} : (\mathtt{B}, \mathtt{L}, \mathtt{E}) \leftarrow \mathbf{Linear}^{\mathbf{z}}(\mathbf{X}'_{t-1})$
$\mathbf{x}' : (\mathtt{B}, \mathtt{L}, \mathtt{E}) \leftarrow \mathbf{SiLU}(\mathbf{Conv1d}(\mathbf{x}))$
$\mathbf{B} : (\mathtt{B}, \mathtt{L}, \mathtt{N}) \leftarrow \mathbf{Linear}^{\mathbf{B}}(\mathbf{x}')$ *// input-dependent*
$\mathbf{C} : (\mathtt{B}, \mathtt{L}, \mathtt{N}) \leftarrow \mathbf{Linear}^{\mathbf{C}}(\mathbf{x}')$ *// input-dependent*
$\mathbf{\Delta} : (\mathtt{B}, \mathtt{L}, \mathtt{E}) \leftarrow \log(1 + \exp(\mathbf{Linear}^{\mathbf{\Delta}}(\mathbf{x}') + \mathbf{Parameter}^{\mathbf{\Delta}}))$
// softplus ensures positive step size, input-dependent
$\overline{\mathbf{A}} : (\mathtt{B}, \mathtt{L}, \mathtt{E}, \mathtt{N}) \leftarrow \mathbf{\Delta} \bigotimes \mathbf{Parameter}^{\mathbf{A}}$ *// discritize*
$\overline{\mathbf{B}} : (\mathtt{B}, \mathtt{L}, \mathtt{E}, \mathtt{N}) \leftarrow \mathbf{\Delta} \bigotimes \mathbf{B}$ *// discritize*
$\mathbf{y} : (\mathtt{B}, \mathtt{L}, \mathtt{E}) \leftarrow \mathbf{SSM}(\overline{\mathbf{A}}, \overline{\mathbf{B}}, \mathbf{C})(\mathbf{x}')$ *// hardware-aware scan*
$\mathbf{y}' : (\mathtt{B}, \mathtt{L}, \mathtt{E}) \leftarrow \mathbf{y} \bigodot \mathbf{SiLU}(\mathbf{z})$ *// self-gating*
$\mathbf{X}_t : (\mathtt{B}, \mathtt{L}, \mathtt{D}) \leftarrow \mathbf{Linear}^{\mathbf{X}}(\mathbf{y}') + \mathbf{X}_{t-1}$ *// residual connection*
Return: $\mathbf{X}_t$ *// output sequence*

B. NetMamba Pre-training

1) Random Masking: Given the embedded stride tokens $\mathbf{X}_0 \in \mathbb{R}^{\mathtt{L} \times \mathtt{D}_{\text{enc}}}$, a portion of strides is randomly sampled while the remaining ones are removed. For a predefined masking ratio $r \in (0, 1)$, the length of visible tokens is determined as $\mathtt{L}_{\text{vis}} = \lceil (1 - r)\mathtt{L} \rceil$. The visible tokens are then sampled as follows:

$$\mathbf{X}_0^{\text{vis}} = \mathbf{Shuffle}(\mathbf{X}_0)[1 : \mathtt{L}_{\text{vis}},\ :\ ] \in \mathbb{R}^{\mathtt{L}_{\text{vis}} \times \mathtt{D}_{\text{enc}}} \tag{4}$$

where the **Shuffle** operation permutes the token sequence randomly. Notably, we ensure that the trailing class token remains unmasked throughout this process since its role in aggregating overall sequence information necessitates its preservation at all times.

The primary objective behind random masking is the elimination of redundancy. This approach creates a challenging task that resists straightforward solutions through extrapolation from neighboring strides alone. Additionally, the reduction in input length diminishes computational and memory costs, offering an opportunity for more efficient model training.

2) Masked Pre-training: The NetMamba encoder is tasked with capturing latent inter-stride relationships using the visible tokens, whereas the NetMamba decoder's objective is to reconstruct masked strides utilizing both the encoder output tokens and mask tokens. Each mask token represents a shared, trainable vector indicating the presence of a missing stride. Additionally, new positional embeddings are added to provide location information to the mask tokens.

The formal forward process of NetMamba pre-training can be outlined as follows:

$$\begin{aligned}
\mathbf{X}_{\text{enc}}^{\text{out}} &= \mathbf{MLP}(\mathbf{Encoder}(\mathbf{X}_0^{\text{vis}})) \in \mathbb{R}^{\mathtt{L}_{\text{vis}} \times \mathtt{D}_{\text{dec}}} \\
\mathbf{X}_{\text{dec}}^{\text{in}} &= \mathbf{Unshuffle}(\mathbf{Concat}(\mathbf{X}_{\text{enc}}^{\text{out}}, \mathbf{X}_{\text{mask}})) + \mathbf{E}_{\text{dec}}^{\text{pos}} \\
\mathbf{X}_{\text{dec}}^{\text{out}} &= \mathbf{Decoder}(\mathbf{X}_{\text{dec}}^{\text{in}})
\end{aligned} \tag{5}$$

where the **Unshuffle** operation restores the original sequence order, and $\mathbf{E}_{\text{dec}}^{\text{pos}} \in \mathbb{R}^{\mathtt{L} \times \mathtt{D}_{\text{dec}}}$ represents decoder-specific positional embeddings. Subsequently, the mean square error (MSE) loss for self-supervised reconstruction is calculated as shown below:

$$\begin{aligned}
\mathbf{y}_{\text{real}} &= \mathbf{Shuffle}(\mathbf{X}_0)[\mathtt{L}_{\text{vis}} + 1 : \mathtt{L},\ :\ ] \\
\mathbf{y}_{\text{rec}} &= \mathbf{Shuffle}(\mathbf{X}_{\text{dec}}^{\text{out}})[\mathtt{L}_{\text{vis}} + 1 : \mathtt{L},\ :\ ] \\
\mathcal{L}_{\text{rec}} &= \mathbf{MSE}(\mathbf{y}_{\text{real}}, \mathbf{y}_{\text{rec}})
\end{aligned} \tag{6}$$

where $\mathbf{y}_{\text{real}}$ represents the ground-truth mask tokens, and $\mathbf{y}_{\text{rec}}$ signifies the predicted ones.

C. NetMamba Fine-tuning

For downstream tasks, all encoder parameters, including embedding modules and Mamba blocks, are loaded from pre-training. To conduct classification on labeled traffic data, the decoder is replaced with an MLP head. Given that all stride tokens are visible, fine-tuning of NetMamba is performed in a supervised manner as detailed below:

$$\begin{aligned}
\mathbf{X} &= \mathbf{Encoder}(\mathbf{X}_0) \in \mathbb{R}^{\mathtt{L} \times \mathtt{D}_{\text{enc}}} \\
\hat{\mathbf{y}} &= \mathbf{MLP}(\mathbf{Norm}(\mathbf{X}[\mathtt{L},\ :]))
\end{aligned} \tag{7}$$

Here, $\mathbf{f}$ denotes the trailing class token, and $\hat{\mathbf{y}} \in \mathbb{R}^{\mathtt{C}}$ represent the prediction distribution, where $\mathtt{C}$ is the number of traffic categories. The classification process is then optimized by minimizing the cross-entropy loss between the prediction distribution $\hat{\mathbf{y}}$ and the ground-truth label $\mathbf{y}$:

$$\mathcal{L}_{\text{cls}} = \mathbf{CrossEntropy}(\hat{\mathbf{y}}, \mathbf{y}) \tag{8}$$

Takeaway. *The unidirectional Mamba architecture is well-suited for processing sequential network traffic data. To acquire generic network domain knowledge, NetMamba is pre-trained by reconstructing masked strides. For adaptation to specific downstream tasks, NetMamba is fine-tuned by minimizing prediction loss.*

VII. Evaluation

A. Experimental Setup

1) Datasets: To assess the effectiveness and generalization abilities of NetMamba, we conducted experiments using six publicly available real-world traffic datasets encompassing three main classification tasks.

1. **Encrypted Application Classification**: This task aims to classify application traffic under various encryption protocols. Specifically, the CrossPlatform (Android) [31] and CrossPlatform (iOS) [31] contain 254 and 253 applications respectively. Additionally, we use Tor traffic data from 8 communication categories in ISCXTor2016 [32] and VPN traffic data from 7 communication categories in ISCXVPN2016 [33].
2. **Attack Traffic Classification**: This task aims to identify potential attack traffic, such as Denial of Service (DoS) attacks and brute force attacks. We construct 6 data categories using CICIoT2022 [34].

3. **Malware Traffic Classification**: This task aims to distinguish between traffic generated by malware and benign traffic. We use all 20 data categories from the USTC-TFC2016 dataset [35].

2) Comparison Methods: To comprehensively evaluate NetMamba, we conducted comparisons with various open-source baselines and state-of-the-art techniques, as outlined below:

1. Classical machine learning methods such as **AppScaner** [3] and **FlowPrint** [2] that rely on statistical features for traffic classification.
2. Deep learning approaches like **FS-Net** [4] and **TFE-GNN** [6] that utilize packet lengths or raw bytes to perform traffic analysis in a supervised manner.
3. Transformer-based models such as **ET-BERT** [9] and **YaTC** [10] that capture traffic representations during pre-training and subsequently fine-tune for specific tasks with limited labeled data. In particular, we implement **YaTC(OF)** by substituting packet-level and flow-level attention with a global attention module, which expedites model inference while removing its original memory optimization.
4. Transformer variants within the NetMamba backbone, including **NT-Vanilla** and **NT-Linear**. The former replaces Mamba blocks in NetMamba with vanilla Transformer blocks [36] featuring quadratic complexity, while the latter adopts Linear Transformer blocks [37] with linear complexity.

3) Implementation Details: At the pre-training stage, we set the batch size to $\mathtt{B} = 128$ and train models for 150,000 steps. The initial learning rate is set to 1.0×10^{-3} with the AdamW optimizer, alongside a linear learning rate scaling policy. Additionally, a masking ratio of $r = 0.9$ is employed for randomly masking strides.

For fine-tuning, we adjust the batch size to $\mathtt{B} = 64$ and set the learning rate to 2.0×10^{-3}. Each dataset is partitioned into training, validation, and test sets following an 8:1:1 ratio. All models are trained for 120 epochs on the training data, with checkpoints saving the best accuracy on the validation set, subsequently evaluated on the test set.

The NetMamba architecture features an encoder composed of 4 Mamba blocks and a decoder composed of 2 Mamba blocks. More hyper-parameter details can be found in Table III.

The proposed model is implemented using PyTorch 2.1.1, with all experiments conducted on a Ubuntu 22.04 server equipped with CPU of Intel(R) Xeon(R) Gold 6240C CPU @ 2.60GHz, GPU of NVIDIA A100 (40GB $\times$ 4).

4) Evaluation Metrics: We assess the performance of NetMamba using four typical metrics: Accuracy(AC), Precision(PR), Recall(RC), and weighted F1 Score(F1).

B. Overall Evaluation

We evaluated the performance of NetMamba in categorizing traffic using six publicly available datasets. As shown in Table IV and Table V, NetMamba demonstrates superior performance compared to all baseline methods across five datasets. However, it falls slightly short in comparison to TFE-GNN on the CICIoT2022 dataset, with a marginal 0.15 percentage point difference in accuracy and f1 score. On average, NetMamba achieves accuracy levels ranging from 0.9869 to 0.9993 and f1 scores between 0.9864 and 0.9993. Notably, NetMamba maintains the fewest parameters among all deep learning methods, underscoring its efficient yet effective capabilities in traffic representation learning.

TABLE III
HYPER-PARAMETER DETAILS OF NETMAMBA

Variable	Value	Variable	Value	Variable	Value
M	5	$\mathtt{D}_{\text{enc}}$	256	L_s	4
N_h	80	$\mathtt{D}_{\text{dec}}$	128	$\mathtt{N}$	16
N_p	240	$\mathtt{E}_{\text{enc}}$	512	$\mathtt{L}$	401
L_b	1600	$\mathtt{E}_{\text{dec}}$	256	$\mathtt{L}_{\text{vis}}$	41

1) CrossPlatform (Android): As indicated in Table IV, NetMamba demonstrates significant improvements over existing methods on the CrossPlatform(Android) dataset. Specifically, our model achieves a notable improvement of 4.83% in accuracy and 4.64% in f1 score compared to the state-of-the-art method (ET-BERT). ET-BERT focuses solely on learning traffic representations from packet payloads, overlooking the valuable content information carried by packet headers. In contrast, NetMamba effectively models both header and payload characteristics, leading to a more comprehensive analysis of traffic patterns.

2) CrossPlatform (iOS): On the CrossPlatform(iOS) dataset, NetMamba surpasses all baseline methods and outperforms the state-of-the-art technique (YaTC) by more than 5% across all evaluation metrics. Beyond the variances in base model architecture, NetMamba establishes a more resilient traffic representation scheme compared to YaTC. This enhancement is achieved through techniques such as IP masking and stride cutting, contributing to a stronger overall performance.

3) CICIoT2022: Concerning the CICIoT2022 dataset, NetMamba outperforms all baselines except TFE-GNN, which achieves a slight advantage of 0.15% across all evaluation metrics. TFE-GNN represents traffic flow through a byte-level correlation graph and utilize graph neural networks to capture traffic patterns. However, TFE-GNN lags significantly behind NetMamba on other datasets, particularly trailing by over 23% on the ISCXTor2016 dataset, indicating its unstable classification performance.

4) ISCXTor2016, ISCXVPN2016 & USTC-TFC2016: As depicted in Table VI, NetMamba surpasses all existing methods across the three encrypted datasets. Particularly noteworthy is the significantly unstable performance exhibited by methods other than YaTC and YaTC(OF) across different datasets. Given the similarities between NetMamba and YaTC, we contend that a robust traffic representation scheme, which incorporates both header and payload data, alongside a well-designed pre-training task, plays a crucial role in enhancing

TABLE IV
COMPARISON RESULTS ON CROSSPLATFORM(ANDROID), CROSSPLATFORM(IOS) AND CICIOT2022

Method	Params(M)		CrossPlatform(Android) [31]				CrossPlatform(iOS) [31]				CICIoT2022 [34]			
	PT	FT	AC	PR	RC	F1	AC	PR	RC	F1	AC	PR	RC	F1
AppScanner [3]	-	-	0.1626	0.1646	0.1456	0.1413	0.1718	0.1400	0.1440	0.1283	0.7556	0.8093	0.7244	0.6938
FlowPrint [2]	-	-	0.8739	0.8941	0.8739	0.8700	0.8712	0.8687	0.8712	0.8603	0.5820	0.4164	0.5820	0.4643
FS-Net [4]	-	5.3	0.0147	0.0023	0.0147	0.0034	0.0293	0.0014	0.0293	0.0025	0.5747	0.3800	0.5747	0.4216
TFE-GNN [6]	-	44.3	0.8141	0.8308	0.8141	0.8067	0.8241	0.8326	0.8241	0.8130	1.000	1.000	1.000	1.000
ET-BERT [9]	187.4	136.4	0.9386	0.9451	0.9386	0.9401	0.9105	0.8809	0.9105	0.8850	0.9937	0.9938	0.9937	0.9937
YaTC(OF) [10]	2.3	2.1	0.9177	0.9203	0.9177	0.9176	0.9277	0.9277	0.9277	0.9262	0.9949	0.9949	0.9949	0.9949
YaTC [10]	2.3	2.1	0.9042	0.9081	0.9042	0.9042	0.9310	0.9307	0.9310	0.9295	0.9959	0.9959	0.0059	0.9959
NetMamba	2.2	1.9	0.9869	0.9871	0.9869	0.9864	0.9881	0.9885	0.9881	0.9881	0.9985	0.9985	0.9985	0.9985

TABLE V
COMPARISON RESULTS ON ISCXTOR2016, ISCXVPN2016 AND USTC-TFC2016

Method	Params(M)		ISCXTor2016 [32]				ISCXVPN2016 [33]				USTC-TFC2016 [35]			
	PT	FT	AC	PR	RC	F1	AC	PR	RC	F1	AC	PR	RC	F1
AppScanner [3]	-	-	0.4034	0.2850	0.2149	0.2113	0.7643	0.8047	0.7045	0.7256	0.6998	0.8591	0.6062	0.6633
FlowPrint [2]	-	-	0.1316	0.0173	0.1316	0.0306	0.9666	0.9733	0.9666	0.9681	0.7992	0.7745	0.7992	0.7755
FS-Net [4]	-	5.3	0.7020	0.7010	0.7020	0.6999	0.7023	0.7487	0.7023	0.6660	0.4381	0.2011	0.4381	0.2672
TFE-GNN [6]	-	44.3	0.7692	0.8030	0.7692	0.7618	0.8428	0.8508	0.8428	0.8447	0.9747	0.9747	0.9747	0.9734
ET-BERT [9]	187.4	136.4	0.9980	0.9981	0.9980	0.9980	0.9566	0.9566	0.9566	0.9565	0.9910	0.9911	0.9910	0.9910
YaTC(OF) [10]	2.3	2.1	0.9986	0.9986	0.9986	0.9986	0.9848	0.9848	0.9848	0.9848	0.9951	0.9950	0.9951	0.9950
YaTC [10]	2.3	2.1	0.9959	0.9959	0.9959	0.9959	0.9819	0.9820	0.9819	0.9819	0.9947	0.9749	0.9747	0.9734
NetMamba	2.2	1.9	0.9993	0.9993	0.9993	0.9993	0.9899	0.9899	0.9899	0.9899	0.9990	0.9991	0.9990	0.9990

encrypted traffic analysis capabilities.

C. Inference Efficiency Evaluation

To evaluate the inference efficiency of NetMamba, we conducted experiments comparing its speed and GPU memory consumption with existing deep learning methods. Speed is measured as the number of traffic data samples processed by the model per second: packets for ET-BERT and flows for the others. As shown in Figure 3(a), NetMamba achieves the highest inference speed across various input batch sizes, with improvements ranging from 1.22 to 60.11 times. This advantage is particularly notable due to the substantial model parameters and inefficient model architecture design present in models such as ET-BERT, TFE-GNN, and FS-Net. Even when compared with models possessing similar parameter counts, NetMamba continues to outperform NT-Vanilla, YaTC and its faster variant. This superiority is primarily attributed to Mamba's lower computational complexity compared to Transformer models. Given a token sequence $\mathbf{X} \in \mathbb{R}^{1\times \mathtt{L}\times \mathtt{D}}$ and the default setting $\mathtt{E} = 2\mathtt{D}$, $\mathtt{N} = 16$, the computational complexities of vanilla or linear attention in Transformers and SSM in Mamba are as follows:

$$\Omega(\text{Vanilla-Attention}) = 4\mathtt{L}\mathtt{D}^2 + 2\mathtt{L}^2\mathtt{D} \tag{9}$$

$$\Omega(\text{Linear-Attention}) = 3\mathtt{L}\mathtt{D}^2 + 2\mathtt{L}\mathtt{D} \tag{10}$$

$$\Omega(\text{SSM}) = 3\mathtt{L}\mathtt{E}\mathtt{N} + \mathtt{L}\mathtt{E}\mathtt{N} = 96\mathtt{L}\mathtt{D} + 32\mathtt{L}\mathtt{D} \tag{11}$$

Self-attention exhibits quadratic complexity to the sequence length L, whereas SSM operates linearly. This computational efficiency makes NetMamba more scalable than Transformer-based models like YaTC and ET-BERT. Although both NT-Linear and NetMamba achieve linear complexity and have similar parameter counts, based on Equation (10) and (11), the SSM requires less computational cost when $\mathtt{D} > 42$. Given our use of $\mathtt{D} = 256$, NT-Linear's slower inference speed compared to NetMamba is reasonable.

In Figure 3(b), NetMamba demonstrates lower GPU memory consumption than most models, except FS-Net, YaTC and NT-Linear, when using large batch sizes. FS-Net's reliance on RNNs, which require linear memory relative to sequence length, reduces memory costs but results in slower inference and poorer classification performance. YaTC reduces memory usage by shortening input sequence length through a model forward trick. Without such an optimization, YaTC(OF) consumes up to four times more GPU memory than NetMamba As shown in the subsequent ablation study, NT-Linear exhibits unstable classification performance due to over-compression of standard attention mechanisms. Compared to other baselines, NetMamba achieves improved memory efficiency primarily by customizing GPU operators that minimize the storage of extensive intermediate states and conduct recomputation during the backward pass.

When the input batch size is set to 64 (the value used in fine-tuning), as depicted in Figure 4, NetMamba exhibits an improvement in speed, being 2.24 times faster than the best baseline, YaTC(OF). Apart from FS-Net, memory-optimized YaTC, and NT-Linear, NetMamba surpasses other methods in

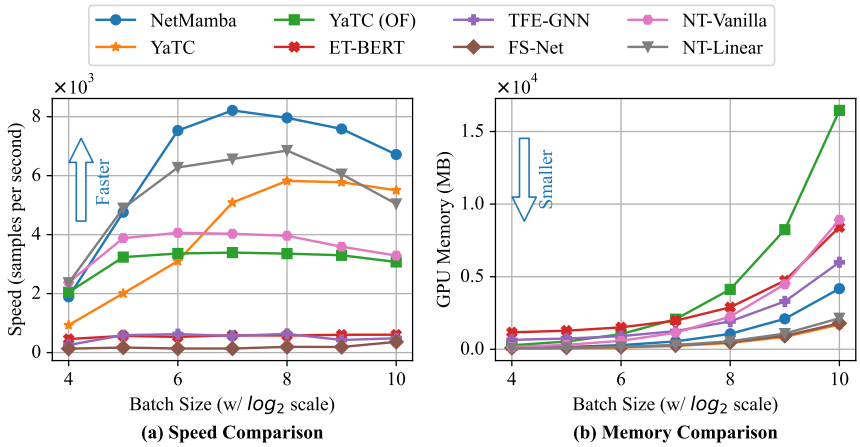

Fig. 3. The Inference Speed and GPU Memory Comparison

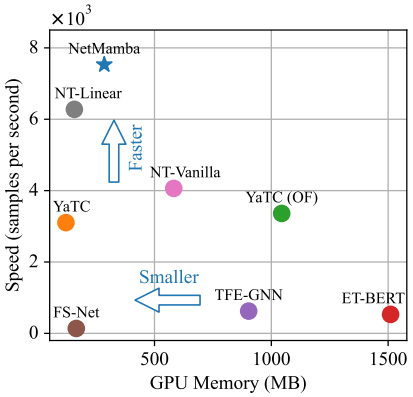

Fig. 4. The Inference Efficiency Comparison on Fine-tuning Batch Size

terms of GPU memory utilization. In summary, NetMamba achieves the highest inference speeds among all deep learning methods while maintaining comparably low memory usage.

D. Ablation Study

To further validate both the model design and the traffic representation scheme of NetMamba, we conducted ablation studies to assess the contribution of each component across six public datasets. The results are presented in Table VI.

1) Model-level Ablation: Initially, we replaced all unidirectional Mamba blocks in NetMamba with bidirectional ones used by [12]. The experimental results revealed a slight performance decline across all datasets except for ISCXVPN2016. This suggests that unidirectional Mamba is well-suited for processing network traffic data, given that packets are transmitted sequentially and earlier packets possess limited information about subsequent ones. Moreover, incorporating bidirectional or even omnidirectional Mamba blocks introduces additional computational and memory overheads due to extra scan passes, ultimately reducing efficiency. Thus, unidirectional Mamba stands out as the preferable choice.

Following [30], we substituted the original unidirectional Mamba block with a cascading structure where each inner block processes data of different granularity. We observed a notable drop in classification performance across all datasets, indicating that this complex structure is inferior to processing sequential traffic data.

To clarify the source of NetMamba's classification performance gains over existing Transformer-based models, we further evaluated two Transformer-based ablation variants, NT-Vanilla and NT-Linear. The results show that NT-Vanilla performs similarly to NetMamba across all datasets, suggesting that our proposed traffic representation scheme plays a key role in enhancing classification performance. With this representation scheme, a linear-time Mamba-based classifier achieves classification performance comparable to that of a quadratic-time Transformer-based model. However, NT-Linear, due to information loss from the overcompression of attention mechanisms, exhibits unstable performance and falls significantly behind NetMamba on three datasets.

While positional information is inherently preserved in sequence models such as Mamba, eliminating explicit positional embedding still results in a reduction in accuracy ranging from 0.03% to 1.35% across all datasets except for ISCXTor2016. This suggests that reinforced positional information aids the model in capturing correlations within sequential traffic data.

The pre-training process is designed to capture general traffic understanding from extensive unlabeled data. When compared to the non-pre-trained counterpart, pre-trained NetMamba demonstrates accuracy improvements ranging from 0.06% to 1.02%, affirming the effectiveness of our MAE-based pre-training task.

2) Data-level Ablation: When all header bytes in a packet are omitted, classification performance declines significantly, with accuracy dropping by 14.51% to 48.54%. This highlights the critical role of key fields within packet headers—such as port number, protocol, and packet length—which have been proven effective in traffic classification [4], [38], [39].

Regarding packet payloads, the ablation results show accuracy drops ranging from 0.26% to 8.29% across five datasets. This underscores the contribution of potential plaintext and specific encrypted payloads to improved traffic understanding.

Without IP masking, the model may learn biased shortcuts based on IP addresses present in the training set, resulting in a maximum decrease in accuracy of 6.27% in the test set.

Likewise, the vertical bias information introduced by the 2-dimensional patch splitting results in a maximum accuracy decline of 0.97%, highlighting the importance of 1-dimensional stride cutting.

E. Few-Shot Evaluation

To validate the robustness and generalization abilities of NetMamba, we conduct few-shot evaluations on four datasets, with labeled data size set to 10%, 40%, 70%, and 100% of the full training set (comprising 80% of the total data).

TABLE VI
ABLATION STUDY OF NETMAMBA ON ALL DATASETS

Method	CrossPlatform(Android)		CrossPlatform(iOS)		CICIoT2022		ISCXTor2016		ISCXVPN2016		USTC-TFC2016	
	AC	F1	AC	F1	AC	F1	AC	F1	AC	F1	AC	F1
NetMamba (default)	0.9869	0.9869	0.9881	0.9881	0.9985	0.9985	0.9993	0.9993	0.9899	0.9899	0.9990	0.9990
w/ Bidirectional Mamba[1]	0.9852	0.9851	0.9793	0.9793	0.9974	0.9974	0.9986	0.9986	0.9935	0.9935	0.9987	0.9987
w/ Cascading Mamba[1]	0.8939	0.8942	0.9470	0.9455	0.9826	0.9826	0.9952	0.9952	0.9646	0.9643	0.9938	0.9937
w/ Vanilla Transformer[2]	0.9815	0.9815	0.9755	0.9755	0.9979	0.9980	0.9986	0.9986	0.9928	0.9927	0.9997	0.9997
w/ Linear Transformer[2]	0.9806	0.9820	0.9871	0.9871	0.8954	0.8998	0.9986	0.9986	0.8295	0.8398	0.4014	0.4226
w/ Flash Transformer	0.9809	0.9811	0.9803	0.9803	0.9969	0.9969	0.9973	0.9973	0.9855	0.9855	0.9995	0.9995
w/o Position Embedding	0.9845	0.9841	0.9746	0.9726	0.9959	0.9959	1.0000	1.0000	0.9877	0.9877	0.9967	0.9966
w/o Pre-training	0.9807	0.9810	0.9779	0.9761	0.9949	0.9949	0.9959	0.9959	0.9798	0.9797	0.9984	0.9984
w/o Header	0.6933	0.7888	0.8430	0.8918	0.5525	0.5333	0.8464	0.8530	0.5441	0.4821	0.5136	0.5675
w/o Payload	0.9040	0.9040	0.9382	0.9383	0.9913	0.9913	1.0000	1.0000	0.9675	0.9676	0.9964	0.9964
w/o IP Masking	0.9242	0.9244	0.9403	0.9404	0.9959	0.9959	0.9979	0.9979	0.9812	0.9812	0.9964	0.9962
w/ Patch Splitting[3]	0.9826	0.9826	0.9784	0.9770	0.9979	0.9979	0.9993	0.9993	0.9892	0.9892	0.9974	0.9974

[1] Substituted unidirectional Mamba blocks with either bidirectional blocks [12] or cascading ones [30].
[2] Replaced Mamba blocks with either vanilla [36] or linear [37] Transformer blocks, termed NT-Vanilla and NT-Linear, respectively.
[3] Changed the 1-dimensional stride cutting to 2-dimensional patch splitting.

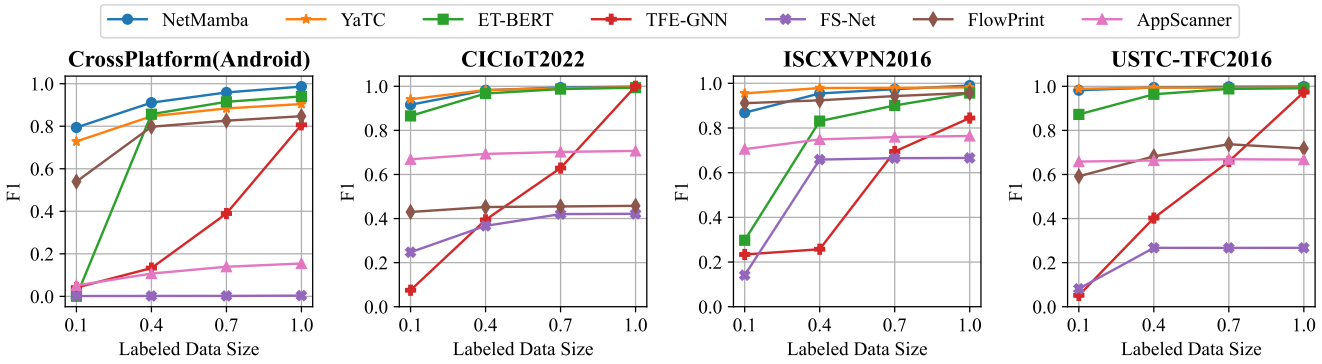

Fig. 5. The Performance Comparison on Few-Shot Settings

Specifically, we adopt a leave-one-out approach on the pre-training datasets to assess the transfer learning capability of the pre-trained models. In detail, the dataset used for fine-tuning is excluded from the pre-training datasets. As shown in Figure 5, the three pre-trained models, NetMamba, YaTC, and ET-BERT generally outperform other supervised methods under few-shot and leave-one-out settings. While conventional machine learning methods like FlowPrint and AppScanner show some robustness to limited labeled data, their classification performance varies significantly across different datasets. Although the supervised TFE-GNN model performs comparably to the pre-trained models with the full training dataset, its performance drops considerably with smaller training data sizes. Thus, pre-trained models demonstrate superior robustness and generalization capabilities due to their ability to extract high-quality traffic representations from large amounts of unlabeled data, thereby reducing the dependence on labeled data.

Among the pre-trained methods, ET-BERT shows less reliability across the four datasets. YaTC performs similarly to NetMamba on three datasets for most labeled data sizes but falls significantly behind NetMamba on the CrossPlatform (Android) dataset. Therefore, our model exhibits excellent robustness and is highly effective at solving classification problems with limited encrypted traffic data.

VIII. CONCLUSION AND FUTURE WORK

In this paper, we introduce NetMamba, a novel pre-trained state space model designed for efficient network traffic classification. To enhance model efficiency while maintaining performance, we utilize the unidirectional Mamba architecture for traffic sequence modeling and develop a comprehensive representation scheme for traffic data. Evaluation experiments on six public datasets demonstrate the superior effectiveness, efficiency, and robustness of NetMamba. Beyond classical traffic classification tasks, the comprehensive representation scheme and refined model design enable NetMamba to address broader tasks within the network domain, such as quality of service prediction and network performance prediction. However, the current implementation of NetMamba depends on specialized GPU hardware, which limits its deployment on real-world network devices. In the future, we plan to explore solutions to implement NetMamba on resource-constrained devices.

REFERENCES

[1] J. Hayes and G. Danezis, "k-fingerprinting: A robust scalable website fingerprinting technique," in *25th USENIX Security Symposium (USENIX Security 16)*, 2016, pp. 1187–1203.

[2] T. Van Ede, R. Bortolameotti, A. Continella, J. Ren, D. J. Dubois, M. Lindorfer, D. Choffnes, M. Van Steen, and A. Peter, "Flowprint: Semi-supervised mobile-app fingerprinting on encrypted network traffic," in *Network and distributed system security symposium (NDSS)*, vol. 27, 2020.

[3] V. F. Taylor, R. Spolaor, M. Conti, and I. Martinovic, "Robust smartphone app identification via encrypted network traffic analysis," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 1, pp. 63–78, 2017.

[4] C. Liu, L. He, G. Xiong, Z. Cao, and Z. Li, "Fs-net: A flow sequence network for encrypted traffic classification," in *IEEE INFOCOM 2019-IEEE Conference On Computer Communications*. IEEE, 2019, pp. 1171–1179.

[5] M. Lotfollahi, M. Jafari Siavoshani, R. Shirali Hossein Zade, and M. Saberian, "Deep packet: A novel approach for encrypted traffic classification using deep learning," *Soft Computing*, vol. 24, no. 3, pp. 1999–2012, 2020.

[6] H. Zhang, L. Yu, X. Xiao, Q. Li, F. Mercaldo, X. Luo, and Q. Liu, "Tfe-gnn: A temporal fusion encoder using graph neural networks for fine-grained encrypted traffic classification," in *Proceedings of the ACM Web Conference 2023*, 2023, pp. 2066–2075.

[7] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[8] K. He, X. Chen, S. Xie, Y. Li, P. Dollár, and R. Girshick, "Masked autoencoders are scalable vision learners," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2022, pp. 16 000–16 009.

[9] X. Lin, G. Xiong, G. Gou, Z. Li, J. Shi, and J. Yu, "Et-bert: A contextualized datagram representation with pre-training transformers for encrypted traffic classification," in *Proceedings of the ACM Web Conference 2022*, 2022, pp. 633–642.

[10] R. Zhao, M. Zhan, X. Deng, Y. Wang, Y. Wang, G. Gui, and Z. Xue, "Yet another traffic classifier: A masked autoencoder based traffic transformer with multi-level flow representation," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 37, no. 4, 2023, pp. 5420–5427.

[11] Q. Wang, C. Qian, X. Li, Z. Yao, and H. Shao, "Lens: A foundation model for network traffic in cybersecurity," *arXiv e-prints*, pp. arXiv–2402, 2024.

[12] L. Zhu, B. Liao, Q. Zhang, X. Wang, W. Liu, and X. Wang, "Vision mamba: Efficient visual representation learning with bidirectional state space model," *arXiv preprint arXiv:2401.09417*, 2024.

[13] J. Qu, X. Ma, and J. Li, "Trafficgpt: Breaking the token barrier for efficient long traffic analysis and generation," *arXiv preprint arXiv:2403.05822*, 2024.

[14] A. Gu and T. Dao, "Mamba: Linear-time sequence modeling with selective state spaces," *arXiv preprint arXiv:2312.00752*, 2023.

[15] W. He, K. Han, Y. Tang, C. Wang, Y. Yang, T. Guo, and Y. Wang, "Densemamba: State space models with dense hidden connection for efficient large language models," *arXiv preprint arXiv:2403.00818*, 2024.

[16] C. Wang, O. Tsepa, J. Ma, and B. Wang, "Graph-mamba: Towards long-range graph sequence modeling with selective state spaces," *arXiv preprint arXiv:2402.00789*, 2024.

[17] W. Zheng, J. Zhong, Q. Zhang, and G. Zhao, "Mtt: an efficient model for encrypted network traffic classification using multi-task transformer," *Applied Intelligence*, vol. 52, no. 9, pp. 10 741–10 756, 2022.

[18] R. Zhao, X. Deng, Z. Yan, J. Ma, Z. Xue, and Y. Wang, "Mt-flowformer: A semi-supervised flow transformer for encrypted traffic classification," in *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2022, pp. 2576–2584.

[19] H. Y. He, Z. G. Yang, and X. N. Chen, "Pert: Payload encoding representation from transformer for encrypted traffic classification," in *2020 ITU Kaleidoscope: Industry-Driven Digital Transformation (ITU K)*. IEEE, 2020, pp. 1–8.

[20] Z. Hang, Y. Lu, Y. Wang, and Y. Xie, "Flow-mae: Leveraging masked autoencoder for accurate, efficient and robust malicious traffic classification," in *Proceedings of the 26th International Symposium on Research in Attacks, Intrusions and Defenses*, 2023, pp. 297–314.

[21] X. Meng, C. Lin, Y. Wang, and Y. Zhang, "Netgpt: Generative pretrained transformer for network traffic," *arXiv preprint arXiv:2304.09513*, 2023.

[22] Y. Liu, Y. Tian, Y. Zhao, H. Yu, L. Xie, Y. Wang, Q. Ye, and Y. Liu, "Vmamba: Visual state space model," *arXiv preprint arXiv:2401.10166*, 2024.

[23] L. Li, H. Wang, W. Zhang, and A. Coster, "Stg-mamba: Spatial-temporal graph learning via selective state space model," *arXiv preprint arXiv:2403.12418*, 2024.

[24] K. Li and G. Chen, "Spmamba: State-space model is all you need in speech separation," *arXiv preprint arXiv:2404.02063*, 2024.

[25] D. Liang, X. Zhou, X. Wang, X. Zhu, W. Xu, Z. Zou, X. Ye, and X. Bai, "Pointmamba: A simple state space model for point cloud analysis," *arXiv preprint arXiv:2402.10739*, 2024.

[26] Y. Qiao, Z. Yu, L. Guo, S. Chen, Z. Zhao, M. Sun, Q. Wu, and J. Liu, "Vl-mamba: Exploring state space models for multimodal learning," *arXiv preprint arXiv:2403.13600*, 2024.

[27] D. Barradas, N. Santos, L. Rodrigues, S. Signorello, F. M. Ramos, and A. Madeira, "Flowlens: Enabling efficient flow classification for ml-based network security applications." in *NDSS*, 2021.

[28] G. Zhou, Z. Liu, C. Fu, Q. Li, and K. Xu, "An efficient design of intelligent network data plane," in *32nd USENIX Security Symposium (USENIX Security 23)*, 2023, pp. 6203–6220.

[29] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020.

[30] T. Chen, Z. Tan, T. Gong, Q. Chu, Y. Wu, B. Liu, J. Ye, and N. Yu, "Mim-istd: Mamba-in-mamba for efficient infrared small target detection," *arXiv preprint arXiv:2403.02148*, 2024.

[31] J. Ren, D. Dubois, and D. Choffnes, "An international view of privacy risks for mobile apps," 2019.

[32] A. H. Lashkari, G. D. Gil, M. S. I. Mamun, and A. A. Ghorbani, "Characterization of tor traffic using time based features," in *International Conference on Information Systems Security and Privacy*, vol. 2. SciTePress, 2017, pp. 253–262.

[33] G. D. Gil, A. H. Lashkari, M. Mamun, and A. A. Ghorbani, "Characterization of encrypted and vpn traffic using time-related features," in *Proceedings of the 2nd international conference on information systems security and privacy (ICISSP 2016)*. SciTePress, 2016, pp. 407–414.

[34] S. Dadkhah, H. Mahdikhani, P. K. Danso, A. Zohourian, K. A. Truong, and A. A. Ghorbani, "Towards the development of a realistic multidimensional iot profiling dataset," in *2022 19th Annual International Conference on Privacy, Security & Trust (PST)*. IEEE, 2022, pp. 1–11.

[35] W. Wang, M. Zhu, X. Zeng, X. Ye, and Y. Sheng, "Malware traffic classification using convolutional neural network for representation learning," in *2017 International conference on information networking (ICOIN)*. IEEE, 2017, pp. 712–717.

[36] A. Vaswani, "Attention is all you need," *arXiv preprint arXiv:1706.03762*, 2017.

[37] A. Katharopoulos, A. Vyas, N. Pappas, and F. Fleuret, "Transformers are rnns: Fast autoregressive transformers with linear attention," in *International conference on machine learning*. PMLR, 2020, pp. 5156–5165.

[38] A. Madhukar and C. Williamson, "A longitudinal study of p2p traffic classification," in *14th IEEE international symposium on modeling, analysis, and simulation*. IEEE, 2006, pp. 179–188.

[39] C. Fu, Q. Li, M. Shen, and K. Xu, "Realtime robust malicious traffic detection via frequency domain analysis," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, 2021, pp. 3431–3446.