netFound: A Domain-specific Foundation Model for Network Security

Anonymous author(s)

ABSTRACT

Despite a rich history of applying ML to network security, most existing solutions lack generalizability. This lack of progress can be attributed to an overreliance on supervised learning techniques and the associated challenges of curating well-specified labeled training data. In this paper, we take inspiration from the recent success of foundation models in other application domains (e.g., GPT4, Vision Transformer) and the growing ease of collecting (unlabeled) telemetry data using programmable networks, to develop a novel transformer-based network foundation model, netFound. It leverages various attributes and constraints unique to network data (packet traces) by developing *multi-modal embeddings*, *protocol-aware tokenization*, *data-driven token composition*, and *hierarchical transformers*. This model is pre-trained using abundant and unlabeled telemetry data from production networks (self-supervised learning) and fine-tuned for disparate downstream learning tasks with sparse, skewed, and noisy labeled datasets (supervised learning). Our evaluation demonstrates the efficacy of the pre-trained model in capturing the complex hidden network context in production settings. We also show that the fine-tuned models not only outperform existing state-of-the-art ML solutions but are also robust to noisy labels and resilient to learning shortcuts—scenarios prevalent in most downstream learning tasks in practice.

1 INTRODUCTION

Machine learning for network security. The National Security Commission on AI report [47] indicates that AI advances have unfavorably empowered malicious actors, making digital ecosystems more vulnerable to a wide range of cyber threats, especially advanced persistent threats (APTs) [4, 23, 43]. It emphasizes that our current approach of countering AI-powered cyber threats, such as APTs, with human intelligence alone is impractical, highlighting a critical need to develop an AI-powered cybersecurity stack, accessible to all. This system, which has often been referred to as the self-driving network [19, 45, 62] should leverage tens or perhaps hundreds of machine learning (ML) models to extract subtle discriminatory signals in the network telemetry data (i.e., packet traces, IDS/firewall let's, etc.) and synthesize surgical corrective actions to neutralize disparate AI-powered cyber threats.

We have a rich history of developing ML-based solutions for disparate learning problems in network security [2, 6, 27, 44, 56, 58], which in theory can serve as the building blocks of the envisioned self-driving network. However, recent works have demonstrated the limitations of existing state-of-the-art ML artifacts for network security [7, 29], specifically highlighting their lack of generalizability. Consequently, we do not have as rich of a history of deploying ML-based solutions in production settings.

We can attribute the lack of progress in developing generalizable ML models for network security to several factors. First, most ML approaches predominantly employ supervised learning techniques that require well-specified training data. However, determining the "right" data for any given learning problem and target network environment, and more importantly, collecting it, is non-trivial. Given this inherent challenge, researchers often resort to using publicly available, yet underspecified training data. These datasets are typically skewed, limited in size, may include insufficient labels, and contain noisy or even mislabeled data points [7, 29, 40]. This reliance on underspecified datasets for training has led to the prevalence of underspecification issues in resulting ML artifacts, such as shortcut learning, out-of-distribution problems, and spurious correlations [29]. Moreover, this reliance on a limited set of public datasets has also narrowed the scope of learning problems that researchers can effectively pursue, resulting in ML solutions that are overengineered with underspecified data and lack the ability to generalize in production.

Key observations. These challenges are not unique and have also impacted other application domains, including vision, natural language processing (NLP), and more. To address these issues, researchers in these domains have explored the design of *foundation models* [76], pre-trained solely using unlabeled data. These pre-trained models learn the inherent relationships in the data through self-supervised learning methods, effectively capturing the hidden context. Applying these pre-trained models to various downstream tasks has yielded transformative results in different ML application domains, particularly in NLP (e.g., GPT4 [48], BERT [15]) and vision (e.g., Vision transformer [60]). Recent works have demonstrated the transformative potential of foundation models on disparate application domains beyond NLP, such as text, images, speech, and reinforcement learning [57]. Moreover, the growing prevalence of software-defined networks (SDN) and the commoditization of programmable data-plane targets have facilitated the collection of unsampled packet traces from production networks for extended durations [10, 32].

netFound— a network foundation model. Inspired by these observations, this paper aims to develop a network foundation model, netFound, which employs self-supervised learning techniques to utilize abundant and unlabeled telemetry data (e.g., packet traces). This approach is intended to create a performant and generalizable network foundation model capable of extracting the hidden networking context from packet-level network traffic data. This context includes the semantics of different applications, network protocols, and their interactions with dynamic network conditions. The model can then be fine-tuned with sparse, skewed, and noisy labeled data for various downstream learning tasks, ultimately producing ML artifacts ready for production use.

Despite all the promises, developing a performant and generalizable network foundation model is challenging as it requires taking several domain-specific attributes and constraints—unique

to network traffic data—into consideration. To this end, this paper makes the following four technical innovations to designing network-specific foundation models:

First, network data is inherently *multi-modal*, comprising diverse types of information such as packet fields (e.g., `TCP Flag`), temporal details (inter-arrival time), contextual information (e.g., direction), and statistical aggregates (e.g., total number of packets). To extract the hidden networking context, it is critical to understand relationships between different packet fields within a packet and across different packets. For instance, relationships within packet fields are dictated by data-plane protocols (e.g., TCP, IPv4, Ethernet), and those across a packet stream in a connection (e.g., TCP session) are influenced by the dynamics between the transport protocol and application logic under varying network conditions (e.g., packet loss, queuing delays). Understanding these relationships requires extracting the inherent multi-modal information. To capture the multi-modal information effectively, we introduce a novel *multi-modal embedding* method that integrates information from packet fields with relevant metadata (e.g., statistical features, direction, position, etc.), enhancing our model to accommodate the diverse data modalities present in network traffic.

Second, network packet headers feature a *pre-defined structure* with fields formatted according to networking protocols (e.g., TCP, UDP, IP). A careful extraction of packet header content is critical for preserving the semantic meaning of the fields. To accurately model the packet header structure, we propose a structure-aware tokenizer that maintains the integrity of the field structures within header tokens. Rather than treating each byte or byte pair as a single token, we develop a *protocol-aware tokenization* method that implements modifications or splitting of header fields for different protocols (TCP, UDP, IP, etc.) to preserve their semantic meanings.

Third, network data exhibits a *hierarchical* structure, with header fields grouped into packets, packets into bursts,[1] bursts into flows, and flows into services and hosts. Capturing interdependencies across these hierarchies and modalities is crucial for modeling the hidden networking context. For example, to understand how protocols like TCP interact with dynamic network conditions, it is crucial for the model to learn the relationships between different multi-modal information (e.g., `TCP Flag`, inter-arrival time, etc.) within a burst as well as across groups of (neighboring) bursts. To effectively capture the inherent hierarchy in network data, we have designed our model as a *hierarchical transformer*. This approach facilitates parameter sharing across different levels of granularity (e.g., packets, bursts, flows, etc.), enhancing the model's ability to learn the inherent hierarchies within the network data.

Finally, the network data exhibits a *heavy-tailed* sequence length distribution at all granularities (packets, bursts, flows, etc.), meaning most sequences are short, but some are extremely long. For instance, the number of fields in a packet, packets in a burst, and bursts in a flow vary significantly, each exhibiting a heavy-tailed distribution. This attribute necessitates careful consideration of the input sequence length for the model at each granularity. While longer sequences provide more information and can enhance model training, they also increase overheads; on the other hand, shorter sequences, though quicker to process, may lack critical insights. To effectively manage heavy-tailed sequences across different hierarchies, we develop a *data-driven token composition* approach to determine the composition of tokens for sequences at different hierarchies, i.e., packets, bursts, and flows—striking a balance between performance and scalability.

By incorporating these innovative features, our transformer-based network foundation model transforms a sequence of (encrypted) network packets as input to fixed-size network data representations that capture complex hidden networking context effectively at different granularities. In contrast, existing network foundation models [16, 24, 39, 52, 75] fail to leverage the critical attributes of network data, often simplifying the data into formats like natural language [24, 39] or images [66, 75] for use with transformer-based models—designed for these specific application domains (e.g., BERT for NLP).

Contributions. This paper makes the following contributions.

- **Novel domain-specific network foundation model.** We present the design (Section 3) and implementation (Section 4) of a novel network foundation model, netFound, that employs multi-modal embedding, protocol-aware tokenization, data-driven token composition, and hierarchical transformer to leverage unique network data attributes and constraints to capture complex hidden networking context at different spatial granularities.
- **An extensive evaluation of the pre-trained model (Section 5).** We pre-train our model, netFound, using packet traces from a production campus network, demonstrating its efficacy in capturing the hidden networking context and its robustness to concept drift. Through a case study, we illustrate how netFound effectively learns multi-modal relationships to decode hidden network contexts. An accompanying ablation study highlights the impact of our design choices.
- **An extensive evaluation of fine-tuned models (Section 6).** We consider five distinct downstream tasks and four state-of-the-art ML models as baseline, including two network foundation models (i.e., ET-BERT [39] and YaTC [75]). We demonstrate that netFound *outperforms all baselines*, achieving significantly better performance on more challenging learning problems. For instance, in traffic classification on a dataset from a production campus network, netFound shows a 9% higher F_1-score. Additionally, netFound exhibits *robustness* to noisy labels, with less than a 5% drop in accuracy even when 40% of the training data is mislabeled. Finally, we highlight netFound's generalizability by showing its resilience to known learning shortcuts that compromise the generalizability of other baselines.
- **Artifacts.** We plan to make the full source code of the system and the datasets used in the paper publicly available.

2 BACKGROUND AND PROBLEM SCOPE

2.1 Machine learning in Network Security

Network traffic classification/Application fingerprinting. This problem involves the categorization of network traffic based on its characteristics, patterns, or content [1, 64]. The most common traffic classification task is application identification, which recognizes applications such as web browsing, file sharing, video streaming,

[1] A burst is a group of unidirectional packets in a flow that are transmitted together.

and email based on collected network traffic and corresponding network configurations [49]. ML technique models application classification as a multi-class classification problem, where each class represents one application and trains supervised classifiers from a (well-)labeled training dataset. Existing research has demonstrated the effectiveness of ML techniques in application classification for network traffic under various network communication setups (VPN and Tor) [35, 67, 68], and when network traffic is encrypted [2, 39].

Network intrusion/anomaly detection. This application identifies network traffic that may indicate malicious activities or intrusion attempts. At a coarse-grained level, existing research often treats anomaly detection as a binary classification problem, where ML models are trained to determine whether a specific network packet or group of packets (i.e., a burst or a flow) is benign or malicious. Similar to traffic classification, existing techniques learn a supervised classifier using a training set with labeled benign and malicious traffic [27, 71]. Some researchers also explore unsupervised learning techniques, which train a model to capture the characteristics of normal traffic and detect deviations from it [44, 73]. At a more fine-grained level, researchers label traffic with the specific type of attack it belongs to and train supervised multi-class classifiers. These classifiers recognize specific intrusion attack types associated with the analyzed traffic.

Advanced persistent threats detection. Advanced Persistent Threats (APT) refer to the attacks that typically infiltrate a target system through malicious uploads, social engineering attacks, etc. After successful infiltration, they move laterally over relatively long intervals to expand their presence, with the ultimate goal of extracting, locking, or transferring critical data. APTs decompose their tasks into multiple stages so that discriminating each of these tasks from normal operations is non-trivial. Intrusion detection introduced above defends against APTs during the infiltration stage. Going beyond this protection, existing research also leverages machine learning at later stages for APT defense. For example, existing research uses machine learning to identify the specific attacks involved in an APT attack's playbook (e.g., Log4J CVE-2021-44228) or identify which hosts are compromised by attackers [4].

Other learning problems. ML has been successfully applied to various other network security challenges, including botnet detection [37], vulnerability assessment [36], etc. These applications are also modeled as either supervised classification problems, where the models learn from labeled data to classify instances, or unsupervised outlier detection problems, where the models identify anomalies without prior knowledge of specific classes.

2.2 Existing Techniques and Limitations

We can divide existing ML-based network security tools into two categories that are either task-specific or task-agnostic.

2.2.1 Task-specific techniques. The solutions in this category involve extracting different features from the network traffic data to curate a (labeled) training set and then using it to train a supervised model for each learning task. Most of the techniques in this category focus on decision-making at the flow-level granularity and vary in the modality of features considered (temporal information [1, 41, 54, 61], packet fields [13, 38, 42, 53, 73, 74], aggregate statistics [35], etc.), feature extraction methods, and model specifications.

Among these efforts, a noteworthy solution is "Look Behind the Curtain" [2] (further denoted as "Curtain" in this paper), which leverages multi-modal information in network data, outperforming all other solutions in this category. However, developing such complex models heavily relies on the availability of abundant high-quality labeled training data that accurately represents the target environment. Unfortunately, curating such datasets remains a daunting task, leading to the use of low-quality publicly available training data that is noisy with limited labeled data points. Recent works [7, 29] have demonstrated that training complex learning models with low-quality data results in underspecification issues (e.g., learn shortcuts, overfit to training data, learn spurious correlations, etc.), leading to poor generalization. Consequently, models trained using these techniques are either not performant or fail to generalize effectively.

2.2.2 Task-agnostic techniques. In this category, solutions develop foundation models using unlabeled network data to learn intermediate network data representations, often referred to as pre-trained or foundation models. These can later be fine-tuned for various downstream learning tasks using labeled data. These solutions vary in feature categories, feature extraction methods, and representation learning models [8, 12, 27, 44]. For instance, nPrintML [27] represents traffic data as fixed-size binary feature vectors (referred to as nPrint vectors), where each bit for different packet fields has a pre-defined position. Although there is no explicit learning component, it is argued that the nPrint representation can adapt to various model architectures for different learning tasks. However, as shown in Section 6, nPrint vectors can only represent superficial network features, potentially leading to learning shortcuts in downstream tasks. In contrast, Kitsune [44] extracts statistical features from the incoming packet stream across multiple temporal windows. These features are clustered and fed to an ensemble of AutoEncoders to learn compressed data representations. While it is theoretically possible to fine-tune these representations for various tasks, Kitsune has primarily been used for anomaly detection.

Transformer-based network foundation models. Inspired by the success of foundation models in other domains and the ease of collecting unlabeled telemetry data, we have witnessed the development of multiple network foundation models in recent years [22, 24, 39, 52, 66, 70, 75]. These solutions vary in embedding, tokenization, token composition methods, model specifications, and related pre-training tasks. Most of these solutions treat network data as natural language or images and employ transformer models developed for these domains for pre-training. For example, ET-BERT treats network data as natural language and uses a transformer architecture designed for NLP, specifically BERT, with pre-training tasks tailored to network data. In contrast, YaTC [75] and Flow-MAE [22] treat network data as images and use transformers developed for the computer vision domain, such as Vision Transformer [17] and Masked Autoencoders [25].

All the existing network foundation models fail to leverage unique network data attributes and constraints, missing the opportunity to fully utilize the network data to extract the underlying

Table 1: Comparison of different network foundation models in capturing network-specific attributes.

Models	Hierarchy	Protocol-aware tokenization	Long sequences	Multi-modal
PERT [24]				
ET-BERT [39]				
Lens [66]				
YaTC [75]	✓			
Flow-MAE [22]	✓		✓	
MTSecurity [70]		✓		✓
TrafficGPT [52]			✓	✓
netFound (ours)	✓	✓	✓	✓

complex hidden networking context (see Table 1). More concretely, ET-BERT considers only a single information modality (the packet's encrypted payload content), employs protocol-agnostic tokenization, and does not utilize temporal information to identify packet bursts. It also limits input to tokens from the first two bytes of up to five packets, ignoring the inherent hierarchy in network data. PERT and LENS are similar to ET-BERT but employ different pre-training tasks. TrafficGPT [52] attempts to handle long sequences with sparse attention and designs an objective function similar to GPT models rather than BERT but still fails to preserve the semantic meanings of different packet fields. Moreover, despite ingesting longer sequences, its approach does not capture the inherent hierarchy, and naively takes a longer sequence of the first few tokens as input, compared to previous work. As an improvement, both YaTC [75] and Flow-MAE [22] attempt to capture the internal hierarchy of network traffic, but they do not support parameter sharing to learn interdependencies across different granularities.

This inability to capture some or all these critical attributes jeopardizes the potential of these foundation models to extract the hidden networking context, which is reflected in their performance for various downstream learning tasks (see Section 6 for details).

3 OVERVIEW OF DESIGN CHOICES

We now describe the design choices that enabled us to develop a network foundation model, which incorporates various domain-specific attributes and constraints of network data. This approach addresses the fundamental limitations of existing solutions, offering better opportunities to learn the complex and dynamic hidden networking context. By leveraging abundant unlabeled network telemetry data, our model catalyzes the development of performant, generalizable, and robust fine-tuned learning models. These models are tailored for diverse learning problems in network security, even when faced with sparse, skewed, and noisy labeled data.

3.1 Preserving Packet Field Semantics

As a first step, we explore how to tokenize packet fields in network data. Recall that packet headers are structured according to various data-plane protocols such as TCP, UDP, and IP. To effectively capture the hidden networking context, it is crucial for the foundation model to consider individual packet fields and preserve their semantic integrity during tokenization. Traditional approaches like ET-BERT utilize protocol-agnostic fixed-sized chunks (e.g., 2 bytes) for tokenization, which can blend different packet fields, resulting in a loss of semantic meaning. Conversely, nprintML segments packet headers into one-bit chunks, maintaining their relative positions across protocols. While this method partially preserves semantic integrity, it complicates the learning process by requiring the model to discern relationships between tokens that represent segments of the same packet field, increasing the underspecification risk [7, 29]. To overcome these challenges, we employ a *protocol-aware tokenization* strategy, segmenting packet headers based on their protocol-specific fields. This approach allows for variable byte lengths among tokens without complicating embedding or model training. By preserving the semantic integrity of packet fields, the model learns the hidden network context more effectively.

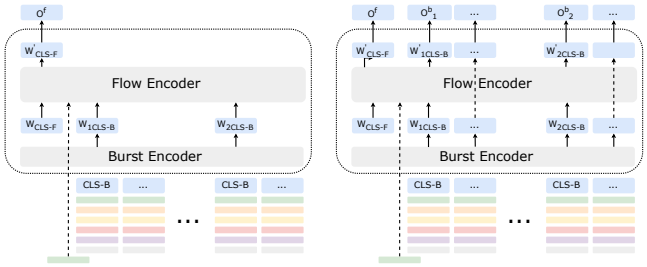
3.2 Capturing Multi-Modal Inputs

Next, we explore how to embed multi-modal information/features into tokens. As previously mentioned, network data contains various types of multi-modal information, which are essential for uncovering the complex hidden networking context. To capitalize on this, we embed multi-modal data—such as temporal details (timestamps), statistical aggregates (e.g., number of bytes or packets per sequence), and contextual information (e.g., direction, hostname, ASN)—as *metadata* within the tokens derived from packet fields. This metadata is often invariant across tokens within the same granularity level. For example, all tokens within a burst share the meta information, such as the same number of packets in a burst.

We have two options for embedding this metadata: directly concatenating it to each input token at the finest granularity or combining it with the learned data representations at coarser granularities. We opt for the former approach because it enables the transformer model to discern and utilize cross-modal dependencies between metadata features and packet content using self-attention mechanisms. This method not only preserves the integrity of the metadata but also ensures the model's adaptability for incorporating future, more effective features. Importantly, by embedding additional metadata, we enhance each token's content without increasing the sequence length, thus maintaining scalability. This strategy results in a thorough capture and utilization of multi-modal information, significantly enhancing the model's ability to interpret and leverage the intricate complexities of network data.

3.3 Handling Variable-Length Sequences

We now explore the selection of input tokens for model training, addressing two key challenges: (1) the scalability of training models on very long sequences limits us to a finite number of tokens; and (2) the heavy-tailed distribution of sequence lengths at various granularities necessitates maximizing the information extracted from tokens across different layers such as packets, bursts, and flows to enhance learning opportunities for the network foundation model. Existing solutions often overlook the inherent hierarchy in network data, typically serializing tokens within and across packets and selecting only the initial ones for input, thus neglecting significant information in later packet fields within a burst or flow and failing to capture dependencies across packets and bursts [22, 27, 39]. To overcome these limitations, we adopt a data-driven approach where we explore the distribution of sequence lengths in the training data and then select median bursts to represent each flow and median packets for each burst, using padding for shorter sequences. This



(a) Strawman 2 (b) Proposed architecture

Figure 1: Comparison between a naive hierarchical model (strawman 2) and the proposed hierarchical transformer.

median selection strikes a balance between sequence length and minimizing padded tokens, which is wasteful. Note that selecting higher percentile values (e.g., max or 100 percentile) instead of the median would result in longer sequences (difficult to scale) with a significant fraction of padding (wasteful), whereas smaller percentile values would involve selecting fewer tokens, compromising the learning ability of the pre-trained foundation model.

3.4 Leveraging Inherent Hierarchy

We now explore how to design the model architecture for the network foundation to ensure it leverages inherent hierarchy in network data. As a first step, we utilize the transformer as the core architecture as it offers several advantages over other architectures, such as variational auto-encoders (VAEs) [34], generative adversarial networks (GANs) [20]), etc. For instance, transformers excel at capturing long-term dependencies, handling varied input modalities, and supporting self-supervised learning—crucial for dealing with the abundant and multi-modal unlabeled data in networking. These capabilities, coupled with the self-attention mechanism, enhance the model's adaptability and generalizability across different datasets, making it ideal for learning the inherent network data hierarchies and applying them to various downstream tasks.

Strawman 1: Disaggregated models. One option is to disaggregate foundation models, i.e., train different foundation models to represent packets, bursts, flows, sessions, devices, etc. However, this approach is ineffective at finer granularities and hard to scale for coarser ones. Specifically, such an approach fails to capture the relationships across various packets, bursts, flows, and so on, resulting in ineffectiveness at finer granularities. Moreover, models working at coarser granularity levels must manage extremely long sequences to perform effectively. Balancing performance and scalability with this approach is challenging given the non-linear relationship between sequence length and training time for transformers [77]. These observations motivate a *hierarchical transformer architecture*, where we iteratively feed representations from finer granularities to learn representations for the coarser ones.

Strawman 2: Naive hierarchical architecture. Figure 1a illustrates the adaptation of hierarchical models from NLP and binary analysis tasks [21, 72] to network data. It shows the division of a network flow into multiple bursts, each containing a set number of packets, which in turn hold fixed token counts. The feature vectors from each burst, detailed later in Section 4, are fed into a burst encoder—a transformer that outputs burst representations. These are then processed by a flow encoder, another transformer, which constructs a comprehensive flow representation. However, this method faces significant challenges: it primarily supports end-to-end supervised learning at the flow level without enabling token-level self-supervised pre-training, and it fails to account for dependencies within the same granularity, such as the interplay between packet attributes across bursts.

Proposed hierarchical architecture. To tackle these limitations, we borrow the idea from the hierarchical transformer in NLP [46] and propose a customized hierarchical architecture for network data with a *skip connection*. Similar to the naive structure, we also use a transformer model as the first layer to process each burst. As demonstrated in Figure 1b, this model outputs a representation for each token in the current burst together with a holistic representation for the burst (CLS). Then, we feed the concatenated input into the second-layer transformer, where we integrate the skipping connection. More specifically, we only input the burst representations to the second layer (indicated by the solid lines) and *skip the token representations* (indicated by the dash lines). Similar to the aforementioned hierarchical structure, the first layer can still capture the token dependencies within bursts, and the second layer can still capture the cross-burst dependency.

In addition, our model could further capture the token dependencies across bursts, which cannot be modeled by naive structure in Figure 1a. More importantly, the outputs of the second layer contain the representation for each input token. This enables us to mask input tokens and predict the masked tokens with their hidden representations. As such, we can train the model with the standard transformer's masking and prediction objective function, which is much more effective and efficient than the auto-encoder objective function [15]. Our method integrates additional CLS tokens at each granularity level to obtain a holistic representation, which will then be used for downstream tasks at that level. As detailed in Section 4, skip connections facilitate parameter sharing across hierarchical layers, enabling our model to handle long flows without truncation.

4 NETFOUND'S WORKFLOW

4.1 Data Pre-processing

We describe how we transform raw packet traces into fixed-sized tokenized representations amenable to transformers.

Step 1: Data extraction. We leverage passively collected packet traces as input. First, we split the larger packet captures (stored as pcaps) into smaller files, one for each flow. Similar to previous works [8, 27, 30], we consider a flow as a group of packets with the same five tuples, i.e., `srcIP`, `dstIP`, `srcPort`, `dstPort`, and `proto` fields. We discard all flows with just 1-2 packets, as most of them are attributable to noisy scanning activities and don't contribute to learning meaningful networking context.

Next, we categorize packets within a flow into "bursts", defined as groups of packets sharing the same direction (inbound or outbound) and having an inter-packet gap of less than or equal to 10 ms. This categorization results in variable quantities of packets per burst and bursts per flow. We have chosen 10 ms as an inter-packet gap as a value that covers 90% percentile of last-mile RTT for our training

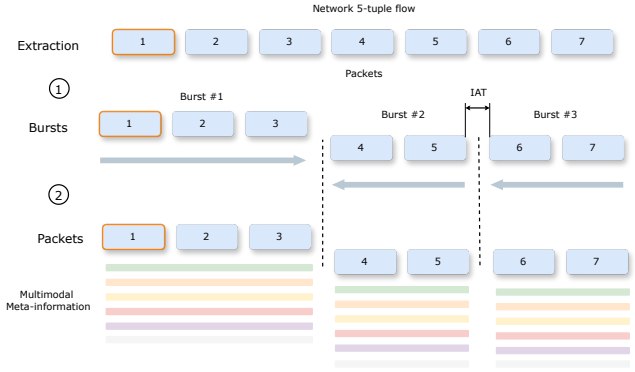


Figure 2: Data extraction & Featurization

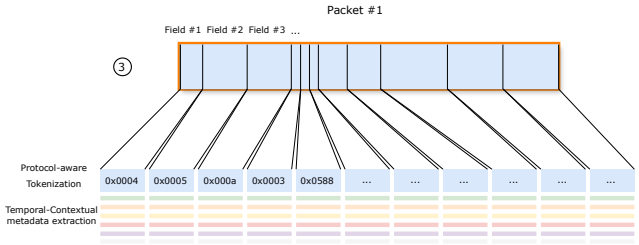


Figure 3: Protocol-aware Tokenization

data, allowing us to set the clear separation border between bursts. Please note, that this value can be changed as per the environment of packet capture.

Then, we standardize the number of packets per burst and bursts per flow. This enables applying the batch operation for transforming training that improves the scalability. We consider up to **12 bursts** per flow and **six packets** (including padding if necessary) per burst.[2] This strategy strikes a reasonable balance between the issues of excessive padding and sequence length while preserving critical inter-packet dynamics.

Step 2: Featurization. For each packet in a burst, we extract various network, transport, and application layer packet fields, transforming these raw fields into a fixed-size vector. Our model eschews flow identifiers like IP addresses, port numbers, SNIs, and domain names, which remain constant within a flow, and do not contribute to learning the flow's latent representation. This strategy ensures that our model focuses on understanding how the hidden networking context influences the spatial-temporal interrelations between packet fields within a flow. To this end, we limit our extraction to the first 12 bytes in the application layer. This specific choice enables the model to glean insights from unencrypted application-layer fields (e.g., DNS header) while circumventing potential learning shortcuts via service name identifiers (SNI) in TLS headers or domain names in DNS messages. Specifically, we consider up to 13 different packet fields and up to 279-bit long vectors to represent packets, depending on the transport layer protocol. Table 8 (in the Appendix B) lists the set of packet fields we consider for generating the feature vector.

Besides packet fields, we also extract various *metadata* fields from the network data. Specifically, we extract contextual and statistical information at packet- and burst-level, such as direction (outbound vs. inbound), number of bytes per burst, number of packets per burst, start time of a burst, inter-arrival time, etc. Note that for tasks necessitating decisions at coarser granularities, such as OS fingerprinting, we can leverage our modular design to integrate the hidden representations of a flow, learned by the model with the flow-level meta information, which includes identifiers (e.g., five-tuples) as well as various flow-level statistical features (e.g., average inter-arrival time, the total number of bytes, etc.) to decipher the latent attributes of pertinent target end hosts.

Step 3: Tokenization. To tokenize the extracted packet-level feature vector, we opt for 2-byte (16-bit) tokens. As mentioned in 2.2, we are using a protocol-aware tokenization strategy, splitting the tokens as per the packet fields to help model the dependencies between them.

- We parse the packet headers and extract them to separate 2-byte-wide tokens. We intentionally extract separate fields from the headers and expand them to 2 bytes if needed, presenting the model with structurally correct information. For example, the IP Header Length field takes 4 bits in the IP packet header and is expanded into a separate 2-byte token during tokenization.
- As TCP Sequence and Acknowledgement numbers are 32-bit values, we split them into 2 tokens each, representing higher bits and lower bits of these fields as separate tokens.
- We don't consider more than 12 bytes of payload per packet, this is to ensure shortcuts from TLS fields such as SNI values are not captured.
- The fields vary across protocols, and hence we generate different tokens as per protocol.

Consequently, for a given vector, we generate a maximum of 18 tokens per packet, resulting in a maximum of 108 tokens per burst (up to 6 packets in a burst) and 1296 tokens per flow (up to 12 bursts per flow). In order to discern the change in token lengths due to protocol, we also add the protocol number as a metadata feature for each flow.

We also incorporate several *special tokens* for learning. Firstly, the [PAD] token is used to equalize input lengths, facilitating batch operation. For instance, for an input set to 90 tokens, each burst comprising 108 tokens gets segmented into six inputs, with the final segment containing 18 [PAD] tokens to maintain uniform token counts across inputs. Secondly, we introduce two distinct tokens: a burst-level [CLS-B] token and a flow-level [CLS-F] token. As illustrated in Figure 1b, the [CLS-B] token precedes each burst, with its output serving as an aggregate representation of the bursts. Conversely, the [CLS-F] token is positioned at the forefront of all burst representations within a flow, providing a comprehensive representation of the entire flow. Lastly, we employ the [MASK] token to represent nullified entries that require restoration during the self-supervised pre-training phase.

[2]Note that these numbers are specific to the training data collected from a campus network, which we used for pre-training (see Section 5 for details).

4.2 Token Embedding

This step receives a set of tokens at different granularity levels. To ensure the model is differentiable, we need to convert the discrete one-hot representation of each token into a continuous representation that is differentiable. We consider three types of embedding for each token: packet field, positional, and metadata. More concretely, we denote each input sequence as $\mathbf{X}^p \in \mathbb{R}^{l \times p}$, where $p = 65,539$ equals the vocabulary size of all tokens, including the special tokens. The i-th row in $\mathbf{X}^p$ is the one-hot representation of the i-th token in the input, with one element as one and all the other elements as zero. We then denote the meta-information as $\mathbf{X}^m \in \mathbb{R}^k$, where each element represents k attributes. In our current implementation, please note that we considered five different meta attributes (i.e., $k = 5$) for each (packet-field) token: the direction of the burst it belongs to, the total number of packets and bytes in the burst, the difference in its burst's arrival time compared to the previous one, and protocol for the flow in order to ingest a different number of tokens due to different header fields across protocols.

Packet field token embedding. Following the classical WordEmbedding technique [3], we design a token embedding layer that takes as input $\mathbf{X}^p$ and outputs an embedding for $\mathbf{X}^p$, denoted as $\mathbf{E}^T \in \mathbb{R}^{l \times q}$. Here, q is a hyper-parameter, standing for the embedding dimension, and l denotes the input length. This layer conducts a linear operation with a learnable weight $\mathbf{W}_T \in \mathcal{R}^{p \times q}$ that transforms $\mathbf{X}^p$ into the token embedding $\mathbf{E}^T \in \mathbb{R}^{l \times q}$ ($\mathbf{E}^T = \mathbf{X}^p \mathbf{W}_T$).

Positional embeddings. We also designed two positional embeddings at the token and the burst level. As demonstrated in Figure 3, our token-level positional embedding is added to the tokens in each burst, ranging from 1 to 108. Each positional embedding is converted into a vector $\mathbf{E}^P \in \mathbb{R}^q$ using the positional embedding layer in the standard transformer model. The readers could refer to [15] for more details about computing this embedding. For the burst-level positional embedding, we add the embedding ranging from 1 to 13 to the [CLS-F] and the 12 burst representations obtained from the burst encoder. We use the same positional embedding layer to compute this positional embedding.

Metadata embedding. We design a meta-information embedding layer to transform the input meta-information $\mathbf{X}^m$ into an embedding with the same dimension as the token and positional embedding, denoted as $\mathbf{E}^m \in \mathbb{R}^q$. Like token embedding, we leverage a linear operation to compute $\mathbf{E}^m$. That is $\mathbf{E}^m = \mathbf{X}^m \mathbf{W}_M$, where $\mathbf{W}_M \in \mathbb{R}^{5 \times q}$ is a learnable weight.

After computing these five embeddings, we compute a final embedding for each token $\mathbf{E}_i$ as the summation of the corresponding three embeddings for each token ($\mathbf{E}_i = \mathbf{E}_i^T + \mathbf{E}_i^P + \mathbf{E}^M$). We treat this concatenation of this embedding as the input to our hierarchical transformer model, where each input is denoted as $\mathbf{E} \in \mathbb{R}^{l \times q}$. Different from the token and positional embedding, all the tokens in the same burst will share the same meta-information embedding.

4.3 Pre-training netFound

As mentioned above, we build the hierarchical model based on the transformer, the predominant architecture for foundation models in various domains [76]. In the following, we introduce the technical details of the transformer, followed by our design of stacking transformers with skip connections.

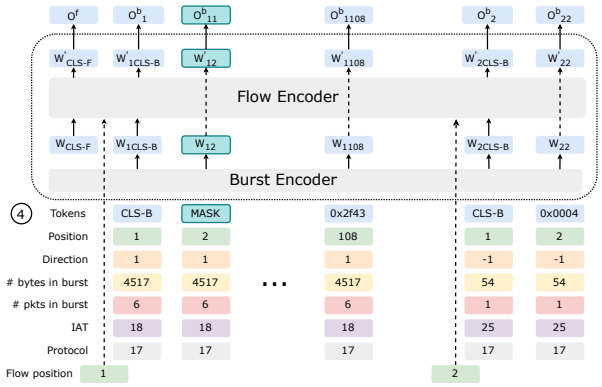


Figure 4: Pre-training—the hierarchical transformer uses a subset of tokens, selected using data-driven methods, for model training. These tokens are extracted from packet fields through protocol-aware tokenization and are augmented with multi-modal embeddings.

Transformer. The transformer model is composed of a series of attention layers, where each attention layer applies the self-attention mechanism [65] multiple times in parallel. More specifically, the first attention layer takes as input a sequence of the final embedding $\mathbf{E}$ introduced above. It consists of H self-attention mechanisms, denoted as attention head. Each attention head first maps this embedding into three distinct representations, denoted as $\mathbf{K}^h \in \mathbb{R}^{l \times q/H}$ (key), $\mathbf{V}^h \in \mathbb{R}^{l \times q/H}$ (value), and $\mathbf{Q}^h \in \mathbb{R}^{l \times q/H}$ (query). These mappings are computed through three affine transformations with learnable weights (fully connected layer). Then, the self-attention operation is conducted by computing the attention weights based on the key and query and then applying the weights to the value, $\mathbf{O}^h = \text{softmax}(\frac{\mathbf{Q}^h (\mathbf{K}^h)^T}{\sqrt{q/H}}) \mathbf{V}^h$, where $\sqrt{q/H}$ is a scaling factor, and softmax is for normalization. The attention weight for a square matrix $\mathbf{A}^h \in \mathbb{R}^{l \times l}$, where $\mathbf{A}_{ij}^h$ expresses the weight of $\mathbf{E}_j$ on $\mathbf{E}_i$ when updating $\mathbf{E}_i$. The attention output $\mathbf{O}^h \in \mathbb{R}^{l \times q/H}$ is a new embedding of the input $\mathbf{E}$ that captures the correlations within the input tokens. Finally, the output of each attention head will be concatenated and passed through a fully connected layer to obtain the final output $\mathbf{O}$, which has the same dimensionality as $\mathbf{E}$. This output fuses the different types of correlations captured by all attention heads in this layer. By stacking multiple attention layers, the transformer model can capture various global and local correlations and dependencies between the input tokens.

Hierarchical transformers with skip connection. To capture the unique dependency of packet traces at different granularity levels, we explicitly build netFound as a hierarchical structure. Specifically, each input flow consists of a sequence of 12 bursts. Each burst starts with a [CLS-B]. In between are the actual tokens (between 0 and 65,535), indicating the header and 12-byte payloads of the packets in that burst. We first design a burst encoder that takes as input all the tokens in each burst and outputs a hidden representation for each token. The output representation of each [CLS-B] token serves as a holistic representation for the corresponding burst that summarizes the key information in the burst.

Then, we append an additional [CLS-F] token at the beginning of the obtained hidden representations and input them into our second layer transformer – flow encoder. To capture inter-burst correlations, we innovatively design skip connections in our flow-level transformer. As shown in Figure 4, we only input the [CLS-F] token and the representations of the [CLS-B] tokens together with their positional embedding into the flow encoder and skip the intermediate tokens. With this design, the flow-level transformer could focus more on learning the dependencies across different bursts. In addition, this skip connection significantly reduces the input length for the flow encoder, avoiding handling the challenge of processing ultra-long sequences with the transformer. The output of the flow-level transformer would be a hidden representation for each input token that encodes the information of all 12 bursts. Similar to the standard transformer, our model's final input is still a hidden representation for each input token (denoted as $\mathbf{O} \in \mathbb{R}^{l \times q}$).

In addition to explicitly capturing two different levels of correlations within the packet traces, another benefit of our hierarchical structure is that it enhances the model's capability of handling long input sequences. As mentioned in ET-BERT [39], constrained by the power of the standard transformer, ET-BERT can only process the input with 512 tokens, corresponding to only the first burst in the flow. This significantly reduces the model's capability of capturing long-term and hidden dependency within a relatively longer flow. Here, our model allows parameter sharing in the first layer and skip connection in the second layer, which significantly reduces the complexity of the model and thus allows the model to handle much longer input sequences (input with 1296 tokens).

Self-supervised pre-training. We follow the standard approach to apply masking [15, 39]. We first randomly choose 30% of tokens from each input sequence. We mask these chosen tokens by replacing them with the special token [MASK] introduced above for 80% of them, 10% of them remain the same and the remaining 10% are randomly chosen tokens. We randomize the masked tokens in each input, even for the same input sequence at different training epochs. We then input the masked sequence into our foundation model and obtain the corresponding hidden representation for each input token, denoted as $[\mathbf{O}_1, ..., \mathbf{O}_l]$. To predict the masked tokens, we then stack a classification layer on top of the foundation model. It takes as input $[\mathbf{O}_1, ..., \mathbf{O}_l]$ and outputs the predicted token $[\hat{\mathbf{X}}_1^p, ..., \hat{\mathbf{X}}_l^p]$. The pre-training objective is to minimize the token prediction errors via the negative log-likelihood loss (NLL): $\min \frac{1}{l} \sum_i -\mathbf{X}_i^p \log(\hat{\mathbf{X}}_i^p)$. Solving this objective function with a first-order optimization method (e.g., ADAM [33]) enables us to efficiently learn the parameters for the foundation model in a self-supervised fashion.

4.4 Fine-tuning netFound

Given that the foundation model has already been pre-trained to provide high-quality representations that capture the hidden correlations within the input sequence, for each task, we create a shallow multi-layer perceptron model (MLP) with only two layers and stack it on top of the netFound model. This model takes the output of [CLS-B] or [CLS-F] (for burst-level and flow-level tasks respectively) token and produces the corresponding prediction.

For training task-specific models we use corresponding loss functions (such as NLL loss for classification tasks), and we also unfreeze and update the foundation model during the fine-tuning process, allowing it to customize the learned representation for the specific downstream task.

As we use only [CLS-B] (or [CLS-F]) output for downstream task learning, the fine-tuning process will only update a subset of parameters that are connected to this token in the foundation model, which ensures fine-tuning efficiency.

We also explored the possibility of training only the shallow model for a downstream task without unfreezing the foundation model. This improves the efficiency of the fine-tuning process and reduces computational complexity, but significantly downgrades the resulting performance on downstream tasks. We choose to update the pre-trained model, given that our selected tasks are more suitable for customizing the representations (See Section 6).

5 EVALUATION OF PRE-TRAINED MODEL

In this section, we aim to answer the following questions. ❶ How effectively can the pre-trained model predict missing tokens, and is it robust to concept drift? ❷ What is the contribution of different design choices on netFound's token prediction performance? ❸ How well can the pre-trained model understand the hidden networking context, i.e., network protocol and conditions?

5.1 Implementation and Experimental Setup

Implementation. We implement our transformer models using PyTorch 1.13.1 and Hugging Face Transformers 4.38.2. We select our model's hyper-parameters via grid search. Here, we specify our choice of hyperparameters (See Appendix D for more details). Specifically, we configure a burst input as a sequence of 108 tokens (6 packets with 18 tokens each), and a flow with 12 bursts, resulting in a sequence length of 1296 tokens. For our architecture, we utilize 12 burst encoders and 12 cross-burst encoders in an interleaved fashion. Each layer has a hidden dimension of 768 and 24 attention heads. This hidden size aligns with the standards set in both the BERT and ET-BERT papers, making it a common choice in most Transformer-related works. To convert metadata into embeddings of size 768, we employ a two-layer MLP with a hidden size of 1024. During pre-training, we randomly mask 30% of input tokens and calculate the loss based on the prediction of these masked tokens.

Datasets. We collect unsampled passive packet traces from our campus network's border router for data collection.[3] We collect data in 15-minute bursts from two different campuses in December 2022 (Campus 1) and September 2023 (Campus 2), utilizing four bursts from each period. To ensure data quality, we exclude flows with fewer than six packets and those where each burst contains two or fewer packets. The Campus 1 dataset includes approximately 2.6 million flows. To test our pre-trained model against temporal variations, we divide Campus 1 flows into a training set (70 %) and a testing set (30 %) based on timestamps. Additionally, we collected another testing set from Campus 2, consisting of 5.5 million flows captured 9 months after the first set.

[3] Our data collection setup, approved by the university's IRB and Committee on IT policies, ensures user privacy by processing only the first 96 bytes and randomizing downstream IP addresses (i.e., campus users' IP) in a prefix-preserving manner.

Table 2: The token prediction F_1 score of our pre-trained model on two testing sets with different token mask percentages.

Datasets	Total Tokens (M)	Masked Tokens (%)		
		10	30	50
Campus 1	22.34	85.89	85.26	83.29
Campus 2	11.32	84.67	84.51	83.01

5.2 Masked Token Prediction (❶)

To answer ❶, we conducted an experiment where we randomly masked a fraction of tokens in two test sets and tasked the model with predicting the masked values. The results, shown in Table 2 as F1-scores (F_1), demonstrate the model's proficiency in learning dependencies between different packet fields across different granularities under varying network conditions. Specifically, a marginal decrease in prediction accuracy with more masked tokens illustrates the model's capacity to capture hidden networking contexts effectively, even with fewer tokens. Notably, the model's performance remains robust over time, as evidenced by the minimal performance decline on the Campus 2 data, collected 9 months after the Campus 1 data, indicating its resilience to concept drift—-a highly-desired attribute for ML models in security applications. Section H (in Appendix) breaks down netFound's performance across various packet fields. Notably, netFound exhibits excellent performance for high entropy packet fields like `ACK2`[4]—demonstrating that overall high performance is not merely attributable to low-entropy fields. However, token prediction accuracy is lower for deeper payload fields and UDP fields, which is attributable to the prevalence of encryption and insufficient representation in pre-training datasets, respectively.

5.3 Ablation Study (❷)

To investigate the impact of design choices on masked token prediction performance (❷), we conducted experiments with various netFound variants, each incorporating additional design elements. The netFound-L model employs a flat transformer architecture and Byte-Pair Encoding (BPE) for long sequences. It uses the same data-driven token composition as netFound but does not include network flow metadata as input. The netFound-LT model enhances netFound-L by using a protocol-aware tokenizer to preserve the semantics of packet fields. Further, netFound-LTM improves token representation by embedding metadata. Lastly, netFound-LTMH adopts a hierarchical attention transformer architecture, incorporating all our design choices.

Figure 5 displays the performance of these variants on all packet fields and specifically on two high-entropy TCP fields: `Flags` and `ACK2`, highlighting the model's ability to grasp the TCP protocol dynamics in varied network conditions. We observe performance improvements with each added feature, confirming the incremental benefit of individual design choices. Notably, netFound-LTMH outperforms all other variants, showcasing the synergistic effect of our comprehensive design approach. This includes a notable enhancement in handling more tokens effectively, especially for `Flags` and leveraging the hierarchical attention mechanism to learn relationships across bursts effectively, significantly impacting the performance on the `ACK2` field.

[4]Lower 16 bits of the TCP Acknowledgement number

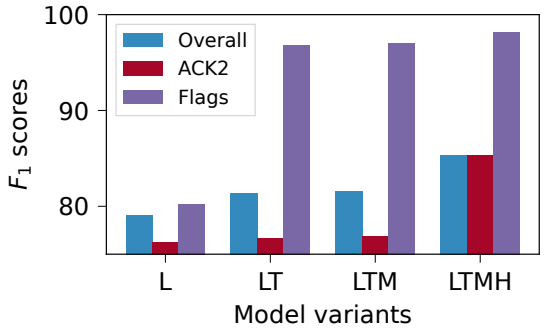


Figure 5: The token prediction performance between netFound and its different ablated variations using long sequences (L), protocol-aware tokenization (T), multi-modality (M), and hierarchy (H).

Case study (❸). We conduct an in-depth case study to answer ❸ and demonstrate how effectively the pre-trained model learns the semantics of the TCP protocol under various network conditions, capturing relationships between different tokens within a packet, across packets in a burst, and across bursts in a flow. See Appendix F for more details.

6 EVALUATION OF FINE-TUNED MODELS

We now aim to answer the following questions: ① Do the fine-tuned models trained with netFound's pre-trained model outperform existing state-of-the-art ML-based solutions for various downstream learning tasks? Are these fine-tuned models ② immune to learning shortcuts, and ③ robust to noisy labels? Finally, ④ what is the impact of different design choices on downstream tasks?

6.1 Experiment Setup

Datasets for downstream tasks. We consider five different downstream supervised learning tasks—traffic classification using the campus dataset, application fingerprinting using the Crossmarket and ISCX-VPN dataset, intrusion detection using the CIC-IDS dataset, and HTTP bruteforce attack detection. Among the five datasets, three of them are publicly available, and we curated the other two locally. We divided each dataset into training and testing sets using a 70:30 ratio. In the following, we introduce the selected dataset for each task.

Campus dataset. We collected packet traces from our campus network and employed the labeling method described in [2], resulting in 72,577 traffic flows grouped into eleven classes, each representing a different type of service. This approach primarily leverages the SNI values in TLS headers for labeling. Table 6 in Appendix A reports the sample distribution of this dataset. Notably, this dataset was collected from the same environment as the pretraining dataset, but we ensured it included traffic with a later timestamp to avoid overfitting.

Crossmarketss dataset [63]. This publicly available dataset contains packet traces for 46,179 flows, categorized into 210 distinct classes.

Table 3: F_1 scores of netFound and baseline models on the selected downstream tasks. The p-value in each cell is computed by comparing the corresponding result with the result of netFound. We highlight the statistically significant best performance in bold. *Acc*@10 represents the accuracy when the correct label is among the top predicted 10 classes.

Task	Type	Dataset	Curtains (%)	nPrintML (%)	ET-BERT (%)	YaTC (%)	netFound (our) (%)
1	Traffic Classification	Campus dataset	54.53 ± 0.97 $p < 0.001$	87.22 ± 0.12 $p < 0.001$	72.26 ± 0.38 $p < 0.001$	76.54 ± 0.23 $p < 0.001$	**96.08 ± 0.04** –
2	Application Fingerprinting	Crossmarkets [63] (*Acc*@10)	20.64 ± 0.13 $p < 0.001$	64.83 ± 0.28 $p = 0.098$	35.62 ± 0.39 $p < 0.001$	58.13 ± 0.89 $p = 0.010$	**66.35 ± 0.99** –
3		ISCXVPN-2016 [18]	66.85 ± 2.21 $p = 0.003$	84.10 ± 0.41 $p < 0.001$	77.57 ± 1.20 $p < 0.001$	83.84 ± 0.24 $p < 0.001$	**91.02 ± 0.10** –
4	Intrusion Detection	CICIDS2017 [55]	99.75 ± 0.16 $p = 0.082$	99.93 ± 0.01 $p = 0.012$	99.94 ± 0.01 $p = 0.018$	99.92 ± 0.01 $p = 0.005$	99.99 ± 0.01 –
5	HTTP Bruteforce Detection	netUnicorn [11]	96.82 ± 0.22 $p = 0.006$	98.51 ± 0.02 $p < 0.001$	98.63 ± 0.02 $p < 0.001$	98.73 ± 0.10 $p = 0.030$	**99.01 ± 0.01** –

The distribution of the number of flows across these classes is heavy-tailed, meaning most classes have very few samples while a few have a large number of flows, averaging 220 examples per class. Previous studies have highlighted the vulnerability of complex models trained on such underspecified and sparse datasets to spurious correlations [7, 29]. Additionally, many flows across different classes correspond to REST calls for retrieving images in related applications such as Amazon, Audible, and Kindle. These flows do not provide sufficient discriminatory information for differentiating between classes at the flow level. Eliminating flows common to multiple application classes would lead to even sparser datasets, increasing susceptibility to spurious correlations. As a compromise, we revised the original application fingerprinting problem's scope and now report top-10 accuracy score,[5] checking if the correct label is among the top 10 predictions out of the 210 classes to accommodate all related applications.

We also identified a learning shortcut in this dataset, where the timestamps in the TCP Options field were the same for all traffic within a class. The presence of this shortcut, ignored by previous works, explains why most previous efforts reported over 90% accuracy scores for this problem. We removed this shortcut and only reported the performance for all models using the dataset without the learning shortcut. Section 6.3 further evaluates the resilience of all considered models to such shortcuts.

ISCXVPN2016 dataset [18]. This publicly available dataset contains 9,536 flows categorized into 17 classes and is widely used in network security for classifying applications running behind VPNs. The inclusion of a VPN presents new challenges due to the increased entropy in the data and variations in packet distribution.

CIC-IDS-2017 dataset [35]. This publicly available dataset contains a mix of "benign" traffic and seven different malicious traffic activities, such as DDoS, SSH-Patator, or other network intrusion attacks. Table 7 in Appendix A describes the dataset in more details.

HTTP bruteforce attack detection. We followed the methodology described in previous work [11] to curate a labeled dataset for HTTP brute-force attack detection. Specifically, we utilized netUnicorn [11] to generate traffic for both benign (i.e., login attempts with valid credentials) and malicious (brute-force attack using the Patator tool [50]) activities within a multi-cloud environment (see Appendix C for details). The dataset comprises 251,047 benign and 142,377 malicious flows.

Baselines. We compare netFound with four leading ML-based approaches: Curtains [2], nPrintML [27], YaTC [75], and ET-BERT [39]. This comparison assesses if netFound surpasses state-of-the-art task-specific methods like Curtains, evaluates the advantages of self-supervised learning over rule-based approaches like nPrintML, and benchmarks our domain-specific strategy against existing network foundation models like ET-BERT and YaTC. We exclude other models from Table 1 due to the lack of open-source implementations (e.g., TrafficGPT [52], MT-Security [70] and LENS [66]) or because our selected baselines already enhance these models (e.g., PERT [24] and Flow-MAE [22]). We employ three popular classifiers—shallow MLP, random forest [26], and SVM [14] for the task-agnostic solutions—-choosing the best performer for results reporting. We have meticulously tuned hyperparameters for each method (see Appendix D for details).

6.2 Effectiveness on Downstream Tasks (①)

Design. To answer ①, we compare netFound with selected baseline methods across five downstream tasks. Unless specified otherwise, we report the mean and standard deviation of the F_1 scores to quantify performance, along with the p-value from the paired t-tests [59] to establish the statistical significance of our comparisons with netFound. Note that we report the top-10 accuracy (*Acc*@10) for the Crossmarkets dataset. This dataset has a large number of classes and each method has a low F1 score.

Results. Table 3 shows that netFound outperforms all four baselines across five downstream tasks, with the most notable advantage observed in the most challenging task, namely, traffic classification over production traffic from our campus network. We note statistically significant performance differences in the application fingerprinting tasks using Crossmarket and ISCXVPN datasets and in the HTTP bruteforce detection task using the netUnicorn dataset. The disparity in performance between Curtain and other network foundation models, particularly netFound, underscores the value of task-agnostic pre-training.[6] The differences between netFound and YaTC/ET-BERT highlight the benefits of our domain-specific approach. Although netFound surpasses nPrintML in all

[5] On the Crossmarket dataset, throughout the paper we intentionally use accuracy instead of F_1 for top-10 predictions as a widely adopted metric.

[6] We identified and eliminated a shortcut learning instance in Curtain, where it used the TLS SNI field both as a feature and for data labeling.

Table 4: F_1 score of netFound and baselines on original versions of datasets (with shortcuts) and fixed (without shortcuts). Performance drop signalizes that the model is vulnerable to shortcut learning.

	CIC-IDS (Heartbleed)		Crossmarket (*Acc*@10)	
	Original	Fixed	Original	Fixed
Curtains	99.43 ± 0.02	86.73 ± 0.04	20.64 ± 0.13	20.64 ± 0.13
nPrintML	99.99 ± 0.01	0.0 ± 0.0	98.35 ± 0.05	64.83 ± 0.28
ET-BERT	99.99 ± 0.01	0.0 ± 0.0	99.82 ± 0.03	35.62 ± 0.39
YaTC	99.99 ± 0.01	0.01 ± 0.01	99.69 ± 0.03	58.13 ± 0.89
netFound	99.99 ± 0.01	99.99 ± 0.01	66.35 ± 0.99	66.35 ± 0.99

tasks, nPrintML somewhat unexpectedly, outperforms YaTC and ET-BERT. We suspect this anomaly might be due to spurious correlations, as it considers a massive input vector size for relatively smaller datasets. Its vulnerability to spurious correlations has been illustrated by previous work [29]. These results conclusively show that the domain-specific design choices embraced by netFound enable the development of performant fine-tuned models for various downstream tasks in different settings.

6.3 Resilience to Learning Shortcuts (②)

Design. To address ②, we compare the resilience of different models to learning shortcuts. Specifically, we examine two learning tasks: intrusion detection (Section 6.3.1) and traffic classification (Section 6.3.2). For each task, we use two versions of datasets: the original version with known learning shortcuts ("Original") and an updated version with the shortcut removed ("Fixed"). We train models on the "Original" dataset with a 70:30 split and evaluate their performance on both the "Original" and "Fixed" datasets. This experiment helps to identify a model's vulnerability to learning shortcuts, thereby testing its generalizability.

6.3.1 CIC-IDS-2017 (Heartbleed). The "Original" dataset is a subset of the CIC-IDS-2017 dataset, which includes only the Heartbleed attack traffic as malicious. Previous work identified a learning shortcut in this dataset because the TCP connections for Heartbleed attacks were not closed between the heartbeat messages [29], enabling most models to learn to distinguish samples using the backward inter-arrival time feature. We use the publicly available "Fixed" version of this dataset (see [29] for details), which generated new traffic while addressing this issue, thereby removing this specific shortcut.

Results. Table 4 displays high performance on the "Original" dataset for nPrintML, ET-BERT, and YaTC, and extremely poor performance on the "Fixed" version, highlighting their vulnerability to this learning shortcut. Although the performance degradation for Curtain is not as significant, its F_1 score drops by more than 10 %. In contrast, netFound's performance remains consistent in both settings, demonstrating its resilience to learning shortcuts. We attribute this to netFound's effective learning from unlabeled data during pre-training, making it immune to such shortcuts.

6.3.2 TCP Options Shortcut in the Crossmarkets Dataset. As discussed before, we identified a new learning shortcut in the widely-used Crossmarkets dataset, where all TCP packets included an additional Timestamp TCP Option field indicating the session time. Analysis revealed that all flows from a single application were collected nearly simultaneously, thus their Timestamp TCP Option values were similar. This bias in the training data could potentially allow models to classify traffic based on timing information, which we consider a learning shortcut. The "Original" version contains this bias, while in the "Fixed" version, this feature is randomized.

Results. Table 4 shows that all models, except Curtains and netFound, experience a significant performance drop after removing the shortcut, indicating overoverreliance on this field for classification. Curtains and netFound, which do not use TCP Options for flow analysis, demonstrate stable performance on both the "Original" and "Fixed" datasets, underscoring the importance of domain knowledge in the tokenization process. Note that although we do not observe a drop in performance for Curtains, its performance is significantly poorer compared to netFound in both settings.

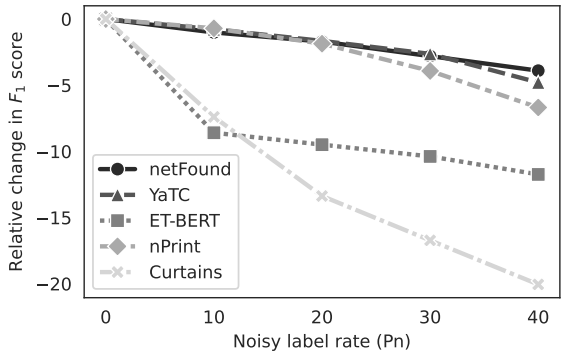


Figure 6: The testing performance of netFound and baselines trained on training sets with different noisy label rates (P_n).

6.4 Robustness against Label Noises (③)

Design. To evaluate the robustness of netFound and baseline models to noise in the training data, we generate several noisy training datasets from the campus dataset for the traffic classification problem. Specifically, we randomly select P_n percent of samples from each class and assign them labels other than their true labels. We consider the uniform noisy distribution, setting the probability of assigning a selected sample to each incorrect class as $\frac{1}{N-1}$, where N is the number of classes in the original dataset, excluding the correct class. Through this process, we construct noisy training datasets with a noise label rate of P_n. We vary P_n = 10%/20%/30%/40% to construct four different training datasets.

Results. Figure 6 shows the relative drop in performance, i.e., the decrease in F_1 score compared to models trained using the original dataset without noise, as the noisy label rate (P_n) increases. We observe only marginal performance degradation for netFound, YaTC, and nPrintML, demonstrating their robustness to noisy labels. In contrast, we observe significant performance degradation for ET-BERT and Curtains.

6.5 Ablation Study (④)

Design. We report the fine-tuning performance of different ablated variants of netFound. Each variant differs based on its support for

Table 5: Fine-tuning performance of different variants of netFound for two downstream tasks. We highlight variants with significant performance improvements (more than 1 %). *Acc*@10 **is top-10 accuracy.**

Variants					Traffic Classification (Campus)	App. Fingerprinting (Crossmarkets)	# Epochs
L	T	M	H	PT	F_1 (%)	*Acc*@10 (%)	
				✓	77.62 ± 0.13	40.95 ± 0.89	5
✓				✓	**94.51 (+16.89) ± 0.03**	**57.58 (+16.63) ± 0.95**	8
✓	✓			✓	94.69 (+0.18) ± 0.39	58.17 (+0.59) ± 0.91	5
✓	✓	✓		✓	94.98 (+0.29) ± 0.14	**64.63 (+6.46) ± 0.91**	4
✓	✓	✓	✓	✓	**96.08 (+1.10)± 0.04**	**66.35 (+1.72) ± 0.99**	4
✓	✓	✓	✓		87.44 (-8.64) ± 0.31	50.58 (-15.77) ± 0.24	7

long sequences **(L)**, use of protocol-aware tokenization **(T)**, the embedding of multi-modal data as metadata **(M)**, employment of a hierarchical attention-based transformer **(H)**, and whether the model was pretrained on unlabeled traffic rather than being fine-tuned from scratch **(PT)**. We fine-tune each of these models using the campus 1 dataset (traffic classification) and the Crossmarkets dataset (application fingerprinting) and report the F_1 scores and *Acc*@10, respectively.

Results. Table 5 shows a monotonic increase in performance as we cumulatively add domain-specific design elements to each preceding variant. The final version, which incorporates all design elements, outperforms all other variants. This result aligns with the observations from the ablation study of the pre-trained model. However, in contrast to the pre-trained model's ablation study, we observe that different problems benefit differently from our design choices. Specifically, the traffic classification problem benefits significantly from considering longer sequences and a hierarchical transformer. In contrast, the application fingerprinting problem shows significant improvements for all features except protocol-aware tokenization. These results highlight the varied and synergistic impact of different design choices on the performance of fine-tuned models for disparate downstream tasks.

Furthermore, the pretrained netFound outperforms netFound trained directly on the classification task from scratch, achieving faster convergence (4 vs. 7 iterations). This underscores the value of pre-training, as it learns general correlations within the input that benefit various downstream tasks, akin to NLP tasks.

7 DISCUSSION

Enhancing netFound with more networking-specific meta attributes. We acknowledge that going beyond the attributes discussed in this paper, there are more networking-specific attributes worth modeling. First, beyond traffic data, we recognize the existence of other data sources in providing valuable information for network security problems. For instance, to defend against APT attacks, we can collect data from deployed intrusion detection systems and system logs [5]. These data sources present heterogeneity and diverse modalities. As part of our future endeavors, we plan to explore the extension of our input embedding layers to accommodate more data modalities, thus empowering our model to effectively handle heterogeneous data. Second, although our model currently comprises two layers, it can be readily extended to capture more fine- or coarse-grained hierarchies of network traffic by simply adding additional transformer layers. Our future work will assess incorporating more layers to netFound to learn packet-level and host-level representations for network traffic. Third, in addition to our current objective function, we will investigate other possible objective functions to train netFound in our future research. For example, we plan to explore leveraging metric learning [31] to explicitly guide the model to identify different network environment setups. We will also explore adding the extra attention layer during the pre-training phase to yield lower-dimensional representations for the foundation model. Finally, we mainly evaluate netFound in an offline setup. Our future work will extend netFound to online setups, where the model needs to be continuously updated and generate representations for dynamic packet traces. We will explore efficient model updating with fast fine-tuning techniques (e.g., LoRA [28]) and integrate our model with methods that support dynamic representation generation (e.g., NTT [16]).

Adversarial robustness. Similar to other transformer-based foundation models, netFound can be vulnerable to adversarial attacks, including poisoning attacks and adversarial evasion attacks. Recent research has explored adversarial attacks against transformer-based models in the field of CV [69] and NLP [51]. However, generalizing these attacks to our model presents challenges, as it necessitates adversarial samples to be actual network traffic that preserves the original semantics. Given that network traffic is often encrypted, manipulating raw bytes while preserving the underlying semantic meaning becomes exceedingly difficult. To the best of our knowledge, there are no existing methods for generating adversarial attacks against transformer models that take as input raw bytes of network traffic. Thus, we defer the assessment of our model's adversarial robustness to future research.

Other future works. First, we will investigate collecting more data from diverse network environments (e.g., data centers or satellite networks) for pre-training and exploring whether and how having a larger number and more diverse pre-training data will affect our foundation model's performance and generalizability. Second, we acknowledge that there are a large number of existing methods for building ML models for network security problems. We focus on comparing netFound with the most representative ones and plan to conduct more extensive comparisons with other relevant methods in future research. Finally, we will explore the possibility of further increasing the sequence length using various methods (e.g., Longformer [9]) to improve the model's capabilities.

8 CONCLUSION

This paper presents the design and implementation of netFound—a domain-specific network foundation model. Through comprehensive evaluations using unlabeled production traffic from our campus network and five labeled datasets across four learning tasks, we demonstrate how our design choices enhance the model's ability to utilize unlabeled telemetry data effectively. This approach not only improves the performance, robustness, and generalizability of fine-tuned models for various learning tasks using sparse, skewed, and noisy data but also advances the capabilities of network foundation models beyond existing methods. netFound's modular design facilitates further exploration and development of advanced ML solutions for complex network security problems, paving the way for future self-driving networks.

REFERENCES

[1] Giuseppe Aceto, Domenico Ciuonzo, Antonio Montieri, and Antonio Pescapé. 2018. Multi-classification approaches for classifying mobile app traffic. *J. Netw. Comput. Appl.* 103 (2018), 131–145.

[2] Iman Akbari, Mohammad A. Salahuddin, Leni Ven, Noura Limam, Raouf Boutaba, Bertrand Mathieu, Stephanie Moteau, and Stephane Tuffin. 2021. A Look Behind the Curtain: Traffic Classification in an Increasingly Encrypted Web. *Proc. ACM Meas. Anal. Comput. Syst.* 5, 1, Article 04 (feb 2021), 26 pages. https://doi.org/10.1145/3447382

[3] Felipe Almeida and Geraldo Xexéo. 2023. Word Embeddings: A Survey. arXiv:1901.09069 [cs.CL]

[4] Meaad Alrehaili, Adel Alshamrani, and Ala Eshmawi. 2022. A Hybrid Deep Learning Approach for Advanced Persistent Threat Attack Detection. In *The 5th International Conference on Future Networks & Distributed Systems* (Dubai, United Arab Emirates) *(ICFNDS 2021)*. Association for Computing Machinery, New York, NY, USA, 78–86. https://doi.org/10.1145/3508072.3508085

[5] Adel Alshamrani, Sowmya Myneni, Ankur Chowdhary, and Dijiang Huang. 2019. A survey on advanced persistent threats: Techniques, solutions, challenges, and research opportunities. *IEEE Communications Surveys & Tutorials* 21, 2 (2019), 1851–1877.

[6] Giovanni Apruzzese, Michele Colajanni, Luca Ferretti, Alessandro Guido, and Mirco Marchetti. 2018. On the effectiveness of machine and deep learning for cyber security. *2018 10th International Conference on Cyber Conflict (CyCon)* (2018), 371–390. https://api.semanticscholar.org/CorpusID:49656174

[7] Daniel Arp, Erwin Quiring, Feargus Pendlebury, Alexander Warnecke, Fabio Pierazzi, Christian Wressnegger, Lorenzo Cavallaro, and Konrad Rieck. 2022. Dos and Don'ts of Machine Learning in Computer Security. In *31st USENIX Security Symposium (USENIX Security 22)*. USENIX Association, Boston, MA, 3971–3988. https://www.usenix.org/conference/usenixsecurity22/presentation/arp

[8] Behnaz Arzani, Selim Ciraci, Stefan Saroiu, Alec Wolman, Jack W. Stokes, Geoff Outhred, and Lechao Diwu. 2020. PrivateEye: Scalable and Privacy-Preserving Compromise Detection in the Cloud. In *Proceedings of the 17th Usenix Conference on Networked Systems Design and Implementation* (Santa Clara, CA, USA) *(NSDI'20)*. USENIX Association, USA, 797–816.

[9] Iz Beltagy, Matthew E. Peters, and Arman Cohan. 2020. Longformer: The Long-Document Transformer. arXiv:2004.05150 [cs.CL]

[10] Roman Beltiukov, Sanjay Chandrasekaran, Arpit Gupta, and Walter Willinger. 2023. PINOT: Programmable Infrastructure for Networking. In *Proceedings of the Applied Networking Research Workshop* (San Francisco, CA, USA) *(ANRW '23)*. Association for Computing Machinery, New York, NY, USA, 51–53. https://doi.org/10.1145/3606464.3606485

[11] Roman Beltiukov, Wenbo Guo, Arpit Gupta, and Walter Willinger. 2023. In Search of netUnicorn: A Data-Collection Platform to Develop Generalizable ML Models for Network Security Problems. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security (CCS)* (Copenhagen, DK).

[12] Francesco Bronzino, Paul Schmitt, Sara Ayoubi, Guilherme Martins, Renata Teixeira, and Nick Feamster. 2019. Inferring Streaming Video Quality from Encrypted Traffic: Practical Models and Deployment Experience. *Proc. ACM Meas. Anal. Comput. Syst.* 3, 3, Article 56 (dec 2019), 25 pages. https://doi.org/10.1145/3366704

[13] Zhiyong Bu, Bin Zhou, Pengyu Cheng, Kecheng Zhang, and Zhen-Hua Ling. 2020. Encrypted network traffic classification using deep and parallel network-in-network models. *Ieee Access* 8 (2020), 132950–132959.

[14] Corinna Cortes and Vladimir Vapnik. 1995. Support-vector networks. *Machine learning* 20, 3 (1995), 273–297.

[15] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *ArXiv* abs/1810.04805 (2019).

[16] Alexander Dietmüller, Siddhant Ray, Romain Jacob, and Laurent Vanbever. 2022. A New Hope for Network Model Generalization. In *Proceedings of the 21st ACM Workshop on Hot Topics in Networks* (Austin, Texas) *(HotNets '22)*. Association for Computing Machinery, New York, NY, USA, 152–159. https://doi.org/10.1145/3563766.3564104

[17] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. 2021. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. arXiv:2010.11929 [cs.CV]

[18] Gerard Draper-Gil, Arash Habibi Lashkari, Mohammad Saiful Islam Mamun, and Ali A. Ghorbani. 2016. Characterization of Encrypted and VPN Traffic using Time-related Features. In *International Conference on Information Systems Security and Privacy*.

[19] Nick Feamster and Jennifer Rexford. 2017. Why (and How) Networks Should Run Themselves. *CoRR* abs/1710.11583 (2017). arXiv:1710.11583 http://arxiv.org/abs/1710.11583

[20] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative Adversarial Networks. arXiv:1406.2661 [stat.ML]

[21] Wenbo Guo, Dongliang Mu, Xinyu Xing, Min Du, and Dawn Song. 2019. DEEPVSA: Facilitating Value-set Analysis with Deep Learning for Postmortem Program Analysis.. In *USENIX Security Symposium*.

[22] Zijun Hang, Yuliang Lu, Yongjie Wang, and Yi Xie. 2023. 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* (<conf-loc>, <city>Hong Kong</city>, <country>China</country>, </conf-loc>) *(RAID '23)*. Association for Computing Machinery, New York, NY, USA, 297–314. https://doi.org/10.1145/3607199.3607206

[23] Wajih Ul Hassan, Adam Bates, and Daniel Marino. 2020. Tactical provenance analysis for endpoint detection and response systems. In *Proceedings - 2020 IEEE Symposium on Security and Privacy, SP 2020 (Proceedings - IEEE Symposium on Security and Privacy)*. Institute of Electrical and Electronics Engineers Inc., United States, 1172–1189. https://doi.org/10.1109/SP40000.2020.00096 Publisher Copyright: © 2020 IEEE.; 41st IEEE Symposium on Security and Privacy, SP 2020 ; Conference date: 18-05-2020 Through 21-05-2020.

[24] Hong Ye He, Zhi Guo Yang, and Xiang Ning Chen. 2020. PERT: Payload Encoding Representation from Transformer for Encrypted Traffic Classification. In *2020 ITU Kaleidoscope: Industry-Driven Digital Transformation (ITU K)*. 1–8. https://doi.org/10.23919/ITUK50268.2020.9303204

[25] Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. 2021. Masked Autoencoders Are Scalable Vision Learners. arXiv:2111.06377 [cs.CV]

[26] Tin Kam Ho. 1995. Random decision forests. In *Proceedings of 3rd international conference on document analysis and recognition*, Vol. 1. IEEE, 278–282.

[27] Jordan Holland, Paul Schmitt, Nick Feamster, and Prateek Mittal. 2020. New Directions in Automated Traffic Analysis. *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security (CCS)* (2020).

[28] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685* (2021).

[29] Arthur S. Jacobs, Roman Beltiukov, Walter Willinger, Ronaldo A. Ferreira, Arpit Gupta, and Lisandro Z. Granville. 2022. AI/ML for Network Security: The Emperor Has No Clothes. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security (CCS)*.

[30] Xi Jiang, Shinan Liu, Saloua Naama, Francesco Bronzino, Paul Schmitt, and Nick Feamster. 2023. AC-DC: Adaptive Ensemble Classification for Network Traffic Identification. arXiv:2302.11718 [cs.NI]

[31] Mahmut Kaya and Hasan Şakir Bilge. 2019. Deep metric learning: A survey. *Symmetry* 11, 9 (2019), 1066.

[32] Hyojoon Kim, Xiaoqi Chen, Jack Brassil, and Jennifer Rexford. 2021. Experience-Driven Research on Programmable Networks. *SIGCOMM Comput. Commun. Rev.* 51, 1 (mar 2021), 10–17. https://doi.org/10.1145/3457175.3457178

[33] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[34] Diederik P Kingma and Max Welling. 2022. Auto-Encoding Variational Bayes. arXiv:1312.6114 [stat.ML]

[35] Arash Habibi Lashkari, Gerard Draper-Gil, Mohammad Saiful Islam Mamun, and Ali A. Ghorbani. 2017. Characterization of Tor Traffic using Time based Features. In *International Conference on Information Systems Security and Privacy*.

[36] Triet H. M. Le, Huaming Chen, and M. Ali Babar. 2022. A Survey on Data-Driven Software Vulnerability Assessment and Prioritization. *ACM Comput. Surv.* 55, 5, Article 100 (dec 2022), 39 pages. https://doi.org/10.1145/3529757

[37] Moemedi Lefoane, Ibrahim Ghafir, Sohag Kabir, and Irfan-Ullah Awan. 2022. Machine Learning for Botnet Detection: An Optimized Feature Selection Approach. In *The 5th International Conference on Future Networks & Distributed Systems* (Dubai, United Arab Emirates) *(ICFNDS 2021)*. Association for Computing Machinery, New York, NY, USA, 195–200. https://doi.org/10.1145/3508072.3508102

[38] Rui Li, Xi Xiao, Shiguang Ni, Haitao Zheng, and Shutao Xia. 2018. Byte segment neural network for network traffic classification. In *2018 IEEE/ACM 26th International Symposium on Quality of Service (IWQoS)*. IEEE, 1–10.

[39] Xinjie Lin, Gang Xiong, Gaopeng Gou, Zhen Li, Junzheng Shi, and Jing Yu. 2022. ET-BERT: A Contextualized Datagram Representation with Pre-Training Transformers for Encrypted Traffic Classification. In *Proceedings of the ACM Web Conference 2022* (Virtual Event, Lyon, France) *(WWW '22)*. Association for Computing Machinery, New York, NY, USA, 633–642. https://doi.org/10.1145/3485447.3512217

[40] Lisa Liu, Gints Engelen, Timothy Lynar, Daryl Essam, and Wouter Joosen. 2022. Error Prevalence in NIDS datasets: A Case Study on CIC-IDS-2017 and CSE-CIC-IDS-2018. In *2022 IEEE Conference on Communications and Network Security (CNS)*. 254–262. https://doi.org/10.1109/CNS56114.2022.9947235

[41] Manuel Lopez-Martin, Belen Carro, Antonio Sanchez-Esguevillas, and Jaime Lloret. 2017. Network traffic classifier with convolutional and recurrent neural networks for Internet of Things. *IEEE access* 5 (2017), 18042–18050.

[42] Mohammad Lotfollahi, Mahdi Jafari Siavoshani, Ramin Shirali Hossein Zade, and Mohammdsadegh Saberian. 2020. Deep packet: A novel approach for encrypted traffic classification using deep learning. *Soft Computing* 24, 3 (2020), 1999–2012.

[43] Sadegh M. Milajerdi, Rigel Gjomemo, Birhanu Eshete, R. Sekar, and V.N. Venkatakrishnan. 2019. HOLMES: Real-Time APT Detection through Correlation of Suspicious Information Flows. In *2019 IEEE Symposium on Security and Privacy (SP)*. 1137–1152. https://doi.org/10.1109/SP.2019.00026
[44] Yisroel Mirsky, Tomer Doitshman, Yuval Elovici, and Asaf Shabtai. 2018. Kitsune: An Ensemble of Autoencoders for Online Network Intrusion Detection. In *25th Annual Network and Distributed System Security Symposium, NDSS*. The Internet Society.
[45] National Science Foundation: Workshop on Self-Driving Networks. [n. d.]. https://www.nsf.gov/awardsearch/showAward?AWD_ID=1748793
[46] Piotr Nawrot, Szymon Tworkowski, Michał Tyrolski, Lukasz Kaiser, Yuhuai Wu, Christian Szegedy, and Henryk Michalewski. 2022. Hierarchical Transformers Are More Efficient Language Models. In *Findings of the Association for Computational Linguistics: NAACL 2022*, Marine Carpuat, Marie-Catherine de Marneffe, and Ivan Vladimir Meza Ruiz (Eds.). Association for Computational Linguistics, Seattle, United States, 1559–1571. https://doi.org/10.18653/v1/2022.findings-naacl.117
[47] nscai [n. d.]. National Security Commission on Artificial Intelligence. https://www.nscai.gov/.
[48] OpenAI. 2023. GPT-4 Technical Report. arXiv:2303.08774 [cs.CL]
[49] Eva Papadogiannaki and Sotiris Ioannidis. 2021. A Survey on Encrypted Network Traffic Analysis Applications, Techniques, and Countermeasures. *ACM Comput. Surv.* 54, 6, Article 123 (jul 2021), 35 pages. https://doi.org/10.1145/3457904
[50] patator [n. d.]. Patator. https://github.com/lanjelot/patator.
[51] Shilin Qiu, Qihe Liu, Shijie Zhou, and Wen Huang. 2022. Adversarial attack and defense technologies in natural language processing: A survey. *Neurocomputing* 492 (2022), 278–307. https://doi.org/10.1016/j.neucom.2022.04.020
[52] Jian Qu, Xiaobo Ma, and Jianfeng Li. 2024. TrafficGPT: Breaking the Token Barrier for Efficient Long Traffic Analysis and Generation. *ArXiv* abs/2403.05822 (2024). https://api.semanticscholar.org/CorpusID:268351552
[53] Shahbaz Rezaei, Bryce Kroencke, and Xin Liu. 2019. Large-scale mobile app identification using deep learning. *IEEE Access* 8 (2019), 348–362.
[54] Tal Shapira and Yuval Shavitt. 2019. FlowPic: Encrypted Internet Traffic Classification is as Easy as Image Recognition. *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)* (2019), 680–687.
[55] Iman Sharafaldin, Arash Habibi Lashkari, and Ali A. Ghorbani. 2018. Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization. In *Proceedings of the 4th International Conference on Information Systems Security and Privacy - Volume 1: ICISSP,*. INSTICC, SciTePress, 108–116. https://doi.org/10.5220/0006639801080116
[56] Kamran Shaukat, Suhuai Luo, Vijay Varadharajan, Ibrahim A. Hameed, and Min Xu. 2020. A Survey on Machine Learning Techniques for Cyber Security in the Last Decade. *IEEE Access* 8 (2020), 222310–222354. https://doi.org/10.1109/ACCESS.2020.3041951
[57] Kevin Shen. 2024. Multi-world Model in Continual Reinforcement Learning. *Proceedings of the AAAI Conference on Artificial Intelligence* 38, 21 (Mar. 2024), 23757–23759. https://doi.org/10.1609/aaai.v38i21.30555
[58] Robin Sommer and Vern Paxson. 2010. Outside the Closed World: On Using Machine Learning for Network Intrusion Detection. In *2010 IEEE Symposium on Security and Privacy*. 305–316. https://doi.org/10.1109/SP.2010.25
[59] Student. 1908. The probable error of a mean. *Biometrika* (1908), 1–25.
[60] Xiu Su, Shan You, Jiyang Xie, Mingkai Zheng, Fei Wang, Chen Qian, Changshui Zhang, Xiaogang Wang, and Chang Xu. 2021. Vision Transformer Architecture Search. *ArXiv* abs/2106.13700 (2021).
[61] Mengxuan Tan, Alfonso Iacovazzi, Ngai-Man Man Cheung, and Yuval Elovici. 2019. A neural attention model for real-time network intrusion detection. In *2019 IEEE 44th conference on local computer networks (LCN)*. IEEE, 291–299.
[62] The Self-Driving Network: Restoring Economic Sustainability to Your Infrastructure. [n. d.]. https://www.juniper.net/us/en/dm/the-self-driving-network/.
[63] Thijs van Ede, Riccardo Bortolameotti, Andrea Continella, Jingjing Ren, Daniel J. Dubois, Martina Lindorfer, David R. Choffnes, Maarten van Steen, and Andreas Peter. 2020. FlowPrint: Semi-Supervised Mobile-App Fingerprinting on Encrypted Network Traffic. *Proceedings 2020 Network and Distributed System Security Symposium* (2020). https://api.semanticscholar.org/CorpusID:211265114
[64] Thijs van Ede, Riccardo Bortolameotti, Andrea Continella, Jingjing Ren, Daniel J. Dubois, Martina Lindorfer, David Choffness, Maarten van Steen, and Andreas Peter. 2020. FlowPrint: Semi-Supervised Mobile-App Fingerprinting on Encrypted Network Traffic. In *NDSS*. The Internet Society.
[65] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).
[66] Qineng Wang, Chen Qian, Xiaochang Li, Ziyu Yao, and Huajie Shao. 2024. Lens: A Foundation Model for Network Traffic in Cybersecurity. *ArXiv* abs/2402.03646 (2024). https://api.semanticscholar.org/CorpusID:267628222
[67] Wei Wang, Ming Zhu, Jinlin Wang, Xuewen Zeng, and Zhongzhen Yang. 2017. End-to-end encrypted traffic classification with one-dimensional convolution neural networks. *2017 IEEE International Conference on Intelligence and Security Informatics (ISI)* (2017), 43–48.
[68] Wenting Wei, Tianjie Ju, Han Liao, Weike Zhao, and Huaxi Gu. 2022. FLAG: Flow Representation Generator Based on Self-Supervised Learning for Encrypted Traffic Classification. In *5th Asia-Pacific Workshop on Networking (APNet 2021)* (Shenzhen, China, China) *(APNet 2021)*. Association for Computing Machinery, New York, NY, USA, 14–20. https://doi.org/10.1145/3469393.3469394
[69] Zhipeng Wei, Jingjing Chen, Micah Goldblum, Zuxuan Wu, Tom Goldstein, and Yu-Gang Jiang. 2022. Towards transferable adversarial attacks on vision transformers. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 36. 2668–2676.
[70] Jin Yang, Xinyun Jiang, Yulin Lei, Weiheng Liang, Zicheng Ma, and Siyu Li. 2024. MTSecurity: Privacy-Preserving Malicious Traffic Classification using Graph Neural Network and Transformer. *IEEE Transactions on Network and Service Management* (2024), 1–1. https://doi.org/10.1109/TNSM.2024.3383851
[71] Kun Yang, Samory Kpotufe, and Nick Feamster. 2020. A Comparative Study of Network Traffic Representations for Novelty Detection. *CoRR* abs/2006.16993 (2020). arXiv:2006.16993 https://arxiv.org/abs/2006.16993
[72] Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. 2016. Hierarchical attention networks for document classification. In *NAACL*.
[73] Yi Zeng, Huaxi Gu, Wei Wenting, and Yantao Guo. 2019. Deep-Full-Range: A Deep Learning Based Network Encrypted Traffic Classification and Intrusion Detection Framework. *IEEE Access* PP (01 2019), 1–1. https://doi.org/10.1109/ACCESS.2019.2908225
[74] Ruijie Zhao, Xianwen Deng, Zhicong Yan, Jun Ma, Zhi Xue, and Yijun Wang. 2022. 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* (Washington DC, USA) *(KDD '22)*. Association for Computing Machinery, New York, NY, USA, 2576–2584. https://doi.org/10.1145/3534678.3539314
[75] Ruijie Zhao, Mingwei Zhan, Xianwen Deng, Yanhao Wang, Yijun Wang, Guan Gui, and Zhi Xue. 2023. Yet Another Traffic Classifier: A Masked Autoencoder Based Traffic Transformer with Multi-Level Flow Representation. *Proceedings of the AAAI Conference on Artificial Intelligence* 37, 4 (Jun. 2023), 5420–5427. https://doi.org/10.1609/aaai.v37i4.25674
[76] Ce Zhou, Qian Li, Chen Li, Jun Yu, Yixin Liu, Guangjing Wang, Kai Zhang, Cheng Ji, Qiben Yan, Lifang He, Hao Peng, Jianxin Li, Jia Wu, Ziwei Liu, Pengtao Xie, Caiming Xiong, Jian Pei, Philip S. Yu, and Lichao Sun. 2023. A Comprehensive Survey on Pretrained Foundation Models: A History from BERT to ChatGPT. arXiv:2302.09419 [cs.AI]
[77] Chen Zhu, Wei Ping, Chaowei Xiao, Mohammad Shoeybi, Tom Goldstein, Anima Anandkumar, and Bryan Catanzaro. 2021. Long-Short Transformer: Efficient Transformers for Language and Vision. In *Advances in Neural Information Processing Systems*, M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan (Eds.), Vol. 34. Curran Associates, Inc., 17723–17736. https://proceedings.neurips.cc/paper_files/paper/2021/file/9425be43ba92c2b4454ca7bf602efad8-Paper.pdf

A DATASETS

Table 6: Table of traffic distribution across all the classes for the Traffic Classification dataset

Service Type	Number of flows
Streaming	3,868
Chat	1,316
Download	4,018
Social	10,000
Games	1,047
Search	6,637
Mail	3,933
Web	10,000
Cloud	10,000
WebApp	10,000
Advertisements	10,000

In this appendix, we provide brief information about traffic distribution between different classes for the campus fine-tuning dataset (see Table 6) and for CIC-IDS-2017 (see Table 7).

Table 7: Table of traffic distribution across all the classes for CIC-IDS-2017

Attack type	Number of flows
SSH Patator	3,958
FTP Patator	2,464
DDOS	45,168
DOS	29,754
Web	2,019
Infiltration	3,757
Port Scan	159,554
Benign	249,044

B PACKET FIELDS USED FOR TOKENIZATION

Table 8 provides information about what packet fields were used for different protocols in the tokenization process. In this work, we used some of the network and transport layer fields for tokenization, as well as the first 12 bytes of the payload. This selection of fields allowed us to include the most relevant information about the packet in the tokenization process but avoid possible shortcuts.

Table 8: Packet fields used in Tokenization

Protocol	Fields				
IPv4	HeaderLen	ToS	TotalLen	Flags	TTL
TCP	Flags	WinSize	SeqNum	AckNum	UrgentPtr
UDP	Length				
ICMP	Type	Code			
Payload	12 bytes				

C PATATOR MULTI-CLOUD DATASET

The Patator Multi-Cloud dataset is generated using virtual machines (VMs) in different cloud and physical infrastructures. We deployed two clusters of machines (five nodes each) on Amazon AWS and our campus infrastructure to use as attacker machines hosting *Patator* software [50], that were targeting the single victim VM, deployed in Microsoft Azure. We also deployed two clusters of machines (five nodes each) on Amazon AWS and our campus infrastructure to generate benign profile traffic with a pattern similar to the attacker's traffic targeting the same victim VM as attacker machines. We captured the full network traffic on the victim VM using *tcpdump*.

D HYPERPARAMETERS

For the netFound model, we used the hidden representation of 768 and 12 layers of BERT encoders for Burst and Flow transformations, and 24 attention heads for the model in total. For the pretraining of netFound model, we chose the learning rate of $2e^{-5}$ and the StepLR parameter as 0.995 every 10,000 steps. During the fine-tuning process, we chose the smaller learning rate of e^{-5} to prevent significant deviations of the already pre-trained weights.

For the ET-BERT implementation, we followed the original paper and used the original hyperparameters, i.e., the learning rate of $2e^{-5}$, hidden size representation of 768, 12 encoders, and 12 attention heads.

Table 9: The testing performance of netFound trained on training sets with different noisy label rates (P_n) using the Traffic Classification dataset.

Noise rate	Curtains(%)	NprintML (%)	ET-BERT (%)	YaTC (%)	netFound (%)
0	54.53 ± 0.97	87.22 ± 0.12	72.26 ±0.38	76.54 ± 0.23	96.08 ± 0.04
10%	47.14 ± 0.26	86.52 ± 0.17	63.67 ± 0.32	75.77 ± 0.09	95.07 ± 0.14
20%	41.16 ± 0.35	85.36 ± 0.19	62.77 ± 0.52	74.89 ± 0.04	94.38 ± 0.2
30%	37.81 ± 0.69	83.30 ± 0.16	61.88 ± 0.44	73.91 ± 0.08	93.28 ± 0.14
40%	34.48 ± 0.95	80.54 ± 0.22	60.52 ± 0.19	71.75 ± 0.37	92.18 ± 0.11

Table 10: Attention weights for flags in packets of the 3rd burst. The green color indicates the labels that were correctly predicted. The color of the cells represents the attention weights. The darker the cell, the more weight it has in the attention vector of the corresponding packet's flag.

Pkt #	TCP Flag (Masked)	Burst-3	Length	Window	Seq #	Ack #
N	ACK	CLS	52	271	517	1922
$N+1$	ACK	CLS	1426	271	517	1922
$N+2$	ACK	CLS	308	271	1891	1922
$N+3$	ACK+PSH	CLS	459	271	2147	1922

For the Curtains implementation, we also followed the approach of the original paper and used the original hyperparameters, i.e. the learning rate of e^{-3} and the StepLR parameter of 0.995.

E NOISY LABEL PERFORMANCE

In this section, we provide the performance of netFound and baselines using the Traffic Classification dataset. We introduce noisy labels in the training set by randomly flipping the labels of a certain percentage of the training samples to any label except the original. We evaluate the performance of the models on the correct test set (without noise). The results are shown in Table 9. We observe that netFound demonstrates the best performance compared to the baselines across all noise rates, and demonstrate robustness to noisy labels comparable to the baselines.

F CASE STUDY

To answer ❸, i.e., demonstrating the pre-trained model's understanding of the hidden networking context, we conduct a case study. In this study, we demonstrate how the pre-trained model learns hidden relationships between various packet fields, revealing the behavior of a network protocol (TCP) and its interaction with the underlying network conditions.

Specifically, we sample 100 K flows from the test data, ensuring they have at least three bursts, and then mask the TCP flag field for all packets in the third burst. By focusing on the third burst, we aim to illustrate the model's ability to comprehend hidden relationships in flows with longer sequence lengths. This sets our approach apart from existing foundation models like ET-BERT, which only focus on the first burst in a flow. Our findings reveal a prediction accuracy of 92.2%, indicating that the model successfully learned the concealed relationships among different packet fields, which can be attributed to the TCP protocol's logic and specification, as well as its interaction with the underlying network conditions.

To gain a deeper understanding of the relationships the model is learning, we further analyze the decision-making for one of the

Table 12: F_1 prediction score of masked headers fields and their corresponding entropy in the pretraining dataset.

Feature Names	TCP		UDP	
	Entropy	F_1	Entropy	F_1
IP header length	0	99.99	0	99.99
IP Type of Service	0.42	99.99	0.67	98.69
IP total length	3.36	89.52	4.75	89.31
IP flags	0.50	99.30	0.53	99.43
TTL	2.84	95.10	2.08	97.76
TCP flags	1.34	98.15	-	-
TCP wsize	5.25	88.10	-	-
TCP seq 1	0.22	99.79	-	-
TCP seq 2	6.14	84.10	-	-
TCP ack 1	0.74	98.67	-	-
TCP ack 2	6.77	85.33	-	-
TCP urp	0.001	99.99	-	-
UDP length	-	-	4.75	88.72
Payload 1	2.56	89.87	8.67	24.19
Payload 2	2.77	88.57	8.99	62.12
Payload 3	4.55	73.33	9.11	63.27
Payload 4	4.17	74.75	10.29	55.10
Payload 5	4.42	74.43	10.62	33.60
Payload 6	4.73	67.44	10.64	32.41

Table 11: The performance between netFound and its different ablated variations using long sequences (L), protocol-aware tokenization (T), multi-modality (M), and hierarchy (H), on the traffic classification task. "# Epochs" means the number of epochs needed for the model to converge during the fine-tuning.

Variations	Features				Token Prediction	Traffic Classification	# Epochs
	L	T	M	H	F_1 (%)	F_1 (%)	
DNN					-	49.12 ± 1.93	15
netFound-Zero					88.75	77.62 ± 0.13	5
netFound-L	✓				79.07 ± 0.04	94.51 ± 0.03	8
netFound-LT	✓	✓			81.38 ± 0.03	94.69 ± 0.39	5
netFound-LTM	✓	✓	✓		81.55 ± 0.02	94.98 ± 0.14	4
netFound-LTH	✓	✓		✓	85.04 ± 0.04	95.92 ± 0.10	4
netFound-LHM	✓		✓	✓	80.46 ± 0.41	94.52 ± 0.08	9
netFound-NoPT	✓	✓	✓	✓	-	87.44 ± 0.31	7
netFound-LTMH	✓	✓	✓	✓	85.26 ± 0.03	96.08 ± 0.04	4

flows in the test data. Randomly selecting a flow with at least three bursts, we mask the TCP flag fields for all packets in the third burst and use the pre-trained model to predict these fields. Table 10 shows that the model accurately predicted TCP flags for all six packets. We report the attention weights for a subset of relevant input fields, highlighting their contribution to predicting the TCP flag field.

We observe that the model gives weightage to the burst representations (CLS). Note that a burst representation captures interdependencies between different tokens within a burst as well as between other bursts in the flow. This result shows the value of employing a hierarchical transformer to capture sequence-wide relationships explicitly. The next highest weight is assigned to tokens pertaining to the packet length field. This finding indicates that the model successfully inferred that a higher value for packet length is a strong indicator for the ACK+PUSH flag. This aligns with the ACK+PUSH flag's purpose of piggybacking data along with the acknowledgment. Thus, the model autonomously learned this protocol-specific behavior. The other fields have little to no weightage, reaffirming that the model correctly learned that these other packet fields are unrelated to TCP flag fields in a burst.

G ABLATION PERFORMANCE

In Table 11 we show the performance of netFound with different combinations of features enabled for both token prediction and traffic classification. Note that netFound-NoPT represents the full model without pretraining (so, trained only on the traffic classification dataset), and DNN represents the simple (non-transformer) linear model fully trained only on the traffic classification dataset.

H HEADERS ENTROPY AND PREDICTION ACCURACY

The Table 12 shows the entropy of the packet headers in our pretraining dataset and the corresponding F_1 score of predicting these fields if they are masked. The results demonstrate that the model successfully predicts fields even with high entropy, which demonstrates its ability to grasp complex patterns of traffic.