Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

ProtoSPM: Protocol-Structured Pretraining for Encoder Reuse in Encrypted Traffic Classification

Wenlei Dai
Southwest Petroleum University
Qin Luo
dorothy_lq@163.com
Southwest Petroleum University
Peng Wu
Sichuan Tourism University

Research Article

Keywords:

Posted Date: November 9th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-11437739/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: No competing interests reported.

ProtoSPM: Protocol-Structured Pretraining for Encoder Reuse in Encrypted Traffic Classification

Wenlei Dai[1], Qin Luo[1*], Peng Wu[2]

[1]Southwest Petroleum University, Chengdu, Sichuan, China.
[2]Sichuan Tourism University, Chengdu, Sichuan, China.

*Corresponding author(s). E-mail(s): dorothy_lq@163.com;

Abstract

Reusable traffic representations can reduce repeated encoder training as application and website labels change. Existing methods commonly pretrain on unlabeled traffic and fine-tune the entire model using labeled data. However, full-model fine-tuning updates the encoder using task-specific labels, making it difficult to isolate the reusability of pretrained representations. We investigate how unlabeled traffic can yield a shared encoder that balances representation quality and model size while supporting different classification tasks without backbone updates. We propose ProtoSPM, a parameter-efficient pretraining model guided by protocol structure. It represents bidirectional flows as protocol-field matrices and learns traffic context through masked reconstruction. Type-aware embeddings, protocol-group temporal convolution, hierarchical field attention, and field-relation biases incorporate value semantics, local packet dynamics, and protocol dependencies into a compact encoder. After pretraining on unlabeled backbone-network traffic, the same encoder remains fixed while separate task heads are trained. On ISCX-VPN-app, TLS-120, and CipherSpectrum, the 12.03M-parameter model achieves macro-averaged F1 (Macro-F1) scores of $\mathbf{61.29 \pm 1.54\%}$, $\mathbf{67.98 \pm 1.23\%}$, and $\mathbf{71.28 \pm 1.20\%}$, respectively. These scores exceed a same-input, near-parameter-matched plain field Transformer by 2.13–26.65 percentage points. The results show that ProtoSPM learns useful shared representations from unlabeled traffic and supports adaptation to different classification label spaces without updating encoder weights.

Keywords: Encrypted traffic classification, Self-supervised pretraining, Encoder reuse, Protocol structural priors, Frozen transfer

1 Introduction

Encrypted traffic classification identifies applications and websites without directly reading communication content. Packet headers, lengths, arrival times, direction changes, and visible handshake metadata still provide useful classification signals (Velan et al. 2015; Rezaei and Liu 2019; Shen et al. 2023; Alshammari and Zincir-Heywood 2011; Anderson and McGrew 2017). As application sets and website labels change, optimizing an entire encoder for every task adds training and model-maintenance costs. Shared traffic representations offer an alternative: keep the backbone weights fixed and learn only the mapping to target labels.

Self-supervised pretraining is an important route to shared representations. ET-BERT, YaTC, NetMamba, and TrafficFormer construct masked-recovery or relation-prediction tasks from unlabeled traffic before fine-tuning with labeled data (Lin et al. 2022; Zhao et al. 2023; Wang et al. 2024; Zhou et al. 2025). An audit of encrypted-traffic representation learning found that random packet-level splits can distribute packets from the same flow across training and test sets (Zhao et al. 2025). Network identifiers can also connect collection environments to labels through shortcuts, producing optimistic classification results. Moreover, full-model fine-tuning directly updates the encoder using downstream labels, allowing it to learn additional task-specific discriminative features. Fine-tuned performance therefore reflects both

pretrained initialization and downstream supervised optimization, rather than the quality of fixed encoder representations alone.

Previous studies have explored different routes to shared traffic representations and task adaptation. Horowicz et al. learned flow embeddings through contrastive self-supervision on traffic images and applied them to few-shot classification (Horowicz et al. 2024). Luxemburk et al. used QUIC domain recognition for supervised pretraining and transferred flow embeddings to seven datasets (Luxemburk et al. 2026). They compared nearest-neighbor classification, linear probing on fixed embeddings, and full-model fine-tuning. TrafficLLM combines multitask prompts, low-rank adaptation, and experts to reduce adaptation costs for multitask traffic analysis (Liu et al. 2026). These studies demonstrate the potential of shared representations, although backbone fine-tuning, adapter updates, and fully frozen encoders require different downstream training procedures.

Real networks continuously generate unlabeled traffic, whereas application and website labels often require controlled collection, service information, or manual verification. Updating the entire encoder with labels for each task retains task-specific backbone training and maintenance costs. It also leaves open whether the same fixed representation can support different label spaces. We therefore learn a shared encoder through unlabeled pretraining and assess its utility by training only a target-task head (Zhao et al. 2025; Luxemburk et al. 2026; Kornblith et al. 2019). Because the head cannot relearn features by updating the backbone, the encoder must already provide sufficiently informative traffic representations.

Regarding shared-encoder design, Pcap-Encoder shows that observable protocol information can support representation learning through header pretraining (Zhao et al. 2025). However, freezing weights does not remove the encoder's parameter-storage and inference costs. Balancing adaptation across classification tasks with a smaller encoder, without downstream backbone updates, thus remains a design challenge.

Protocol structure provides a basis for this design. Field value domains suggest different input mappings, while packet dynamics within a protocol provide a basis for parameter sharing. Within-group and across-group dependencies also call for different levels of interaction. Encoding these properties as model constraints can organize traffic context within limited capacity and improve classification adaptation using fixed representations.

We propose ProtoSPM (Protocol-native Structured Prior Model), a protocol-structured pretraining model that represents bidirectional flows as protocol-field matrices. A structured encoder and masked reconstruction learn packet dynamics and field dependencies; after pretraining, the fixed encoder connects to separate task heads. Type-aware embeddings (TAE) handle heterogeneous field values, while protocol-group-specific temporal convolution (GC) extracts local packet changes through within-group parameter sharing. Hierarchical field attention (HA) organizes within-group and across-group interactions, and field-relation biases (FRB) adjust directed field-pair attention scores. Together, these modules translate protocol semantics into structural constraints for a compact encoder.

We pretrain through masked reconstruction on unlabeled MAWI traffic collected from a real backbone network. The same pretrained encoder then serves ISCX-VPN-app application identification and TLS-120 and CipherSpectrum website identification. These scenarios differ in label-set size and traffic source. A randomly initialized frozen control tests pretraining gains, and a same-input, near-parameter-matched Plain-Field control tests the combined structural design. Component removals further examine the roles of individual modules.

The contributions are as follows:

1. **A compact traffic-pretraining framework for encoder reuse.** ProtoSPM takes protocol-field matrices as input and encodes value semantics, parameter sharing, and hierarchical dependencies as architectural constraints. The resulting compact encoder learns traffic context that different classification tasks can use.
2. **Implementation and evaluation of encoder reuse after unlabeled pretraining.** The same pretrained encoder remains fixed across application and website classification, while independent heads adapt to target labels without reoptimizing the backbone. Pretrained-weight and structural controls further support the benefits of shared representations. The evaluation provides empirical evidence for reusable classification backbones learned from unlabeled traffic collected in real networks.

Section 2 reviews related work, and Section 3 describes the model. Section 4 presents the experimental protocol, results, and component evidence. Section 5 discusses implications and limitations, and Section 6 concludes.

2 Related work

2.1 Encrypted-traffic representations and self-supervised pretraining

Encrypted-traffic representation learning extracts classification information from packet bytes, packet sequences, and flow behavior. Deep Packet processes raw traffic with autoencoders and convolutional networks, while FS-Net learns flow features from packet sequences (Lotfollahi et al. 2020; Liu et al. 2019). These approaches helped move traffic classification from manually designed features toward data-driven representations. More recently, self-supervised pretraining has used large amounts of unlabeled traffic to learn context. ET-BERT encodes datagram bytes as sequences and learns through masked modeling and datagram-relation prediction (Lin et al. 2022). YaTC organizes packets into multilevel flow representations and combines packet- and flow-level attention for masked reconstruction (Zhao et al. 2023). NetMamba uses a state-space model to process traffic sequences efficiently (Wang et al. 2024), whereas TrafficFormer incorporates intra-packet and inter-packet relations into pretraining (Zhou et al. 2025). These approaches emphasize byte context, multilevel representations, efficient sequence processing, and packet relations, respectively.

Such methods typically adapt to target classes through downstream fine-tuning, so their classification performance combines pretrained initialization with supervised optimization. ProtoSPM likewise pretrains on unlabeled traffic, but uses protocol-field matrices and incorporates value types, protocol groups, and field relations into its encoder. The focus is on improving representations formed during pretraining so that fixed encoder weights can support different classification tasks.

2.2 Shared traffic representations and encoder reuse

Shared representations transfer existing model knowledge to new traffic-classification tasks, with different requirements for downstream model updates. Horowicz et al. learned flow embeddings through contrastive self-supervision on traffic images and evaluated few-shot classification (Horowicz et al. 2024). Luxemburk et al. used QUIC domain recognition for supervised pretraining, then compared nearest-neighbor classification, linear probing, and full fine-tuning across multiple datasets (Luxemburk et al. 2026). TrafficLLM combines low-rank adaptation and experts to support multitask traffic analysis while reducing the number of updated parameters (Liu et al. 2026). These approaches explore self-supervised embeddings, supervised transfer, and parameter-efficient adaptation as different ways to serve target tasks with shared encoders.

Frozen-encoder evaluation examines discriminative information already available in pretrained representations, whereas full fine-tuning allows further feature learning from target labels. Encrypted-traffic audits also identify packet-level random splitting and network identifiers as potential sources of leakage and collection-environment shortcuts (Zhao et al. 2025). Classification scores therefore depend partly on dataset construction. ProtoSPM fixes the same pretrained encoder for application and website classification and trains independent task heads. Randomly initialized frozen and structural controls examine the utility of the representations learned from unlabeled traffic. Unlike approaches requiring backbone or adapter updates, this setting preserves one set of encoder weights across label spaces.

2.3 Protocol semantics and structured encoding

Encryption hides communication content, but packet lengths, directions, timing, and visible protocol headers still reveal aspects of communication behavior. FlowPrint builds application fingerprints from temporal correlations among related flows (van Ede et al. 2020). GraphDApp uses graph representations and graph neural networks to identify decentralized applications, illustrating the value of explicitly organizing traffic relations (Shen et al. 2021). Pcap-Encoder restricts pretraining inputs to visible protocol headers and learns protocol semantics through autoencoding and question-answering tasks (Zhao et al. 2025). Behavioral correlation, graph structure, and header semantics thus provide alternatives to directly modeling encrypted bytes.

Protocol fields provide structural information beyond observed numerical values, including value domains, protocol membership, and field relationships. Categorical codes, binary states, and numerical quantities call for different mappings. Fields within the same protocol can share temporal modeling parameters, while within-group and across-group dependencies require different interaction levels. ProtoSPM implements these properties through type-aware mappings, shared protocol-group temporal filters, and hierarchical field interactions. Learnable relation biases further adjust field-pair scores. The aim is to organize limited model capacity using protocol structure and learn traffic representations suitable for frozen reuse.

2.4 Heterogeneous fields and hierarchical interactions

Heterogeneous-field modeling provides general tools for protocol-field representations. FT-Transformer maps numerical and categorical features to field tokens and learns their interactions through a Transformer (Gorishniy et al. 2021). VIME combines mask estimation and value reconstruction to learn representations from unlabeled tabular data (Yoon et al. 2020). The former unifies interactions among heterogeneous fields, while the latter supports reconstruction-based pretraining for structured data. Traffic-field matrices additionally contain packet order and protocol groups, requiring the encoder to organize both temporal changes and field relationships.

Hierarchical and relation-aware attention provide tools for these interactions. Hierarchical attention networks aggregate elements at different granularities (Yang et al. 2016). Shaw et al. incorporate relative relations into self-attention so that relationships between positions contribute to context computation (Shaw et al. 2018). ProtoSPM applies hierarchical interaction to fields within protocol groups and summaries across groups, with field-relation biases learning directed interaction preferences. Compared with generic tabular models, protocol grouping provides an additional basis for parameter sharing and interaction organization. Field types, packet dynamics, and protocol relations thereby jointly contribute to representation learning in a compact encoder.

3 Method

Different classification tasks use different traffic information, but a fixed encoder cannot learn missing features during downstream adaptation. ProtoSPM therefore organizes field values, local packet changes, and protocol dependencies during unlabeled pretraining. Parameter sharing and hierarchical interactions control model size while providing traffic context that task heads can recombine.

Figure 1 presents the architecture. TAE handles heterogeneous value domains, and GC extracts short-range changes through shared within-group filters. Temporal attention and HA extend temporal and field context, while FRB adjusts field-pair relation scores. Each encoder layer applies the same feed-forward network (FFN) twice; relation biases preserve interaction preferences with few additional parameters. Masked reconstruction trains these components jointly on unlabeled traffic. After pretraining, the reconstruction head is removed, the encoder is frozen, and separate classification heads are attached.

3.1 Flow-level protocol-field matrices and training roles

We organize visible protocol headers, parseable handshake information, and packet-level measurements into a fixed field configuration. The fields cover packet attributes, connection state, handshake negotiation, and temporal behavior. TCP fields and extensions follow RFC 9293 and RFC 7323, while TLS handshake fields follow RFC 8446 (Eddy 2022; Borman et al. 2014; Rescorla 2018). Protocol fields describe communication state and negotiation; lengths, directions, and time differences add flow-behavior context. The configuration contains 51 fields in five groups defined by protocol origin and function: Other (6), IP (13), TCP (22), UDP (4), and TLS handshake (6). These groups support subsequent parameter sharing and hierarchical interactions.

Packets are organized by a canonical bidirectional five-tuple $(\min(e_1, e_2), \max(e_1, e_2), \text{proto})$, where each endpoint e_i contains an IP address and port. Each example takes the first $T = 10$ packets in capture order to form $\mathbf{X} \in \mathbb{R}^{T \times N}$, with $N = 51$. Scapy reads standard header fields, and a lightweight parser extracts visible TLS handshake information from individual TCP packet payloads. Short flows and unparsed fields are filled with zero in the original value domain. Pretraining and downstream tasks use the same column order and input mappings; downstream standardization reuses the pretraining statistics. A lists all fields and details flow aggregation, direction assignment, deduplication, and retransmission handling.

Field origin determines interaction groups, whereas the field-generation mechanism determines its pretraining role. Lengths, delays, flags, and handshake attributes describe protocol and behavioral patterns and are designated generalizable fields. Connection-state or operational values, including sequence numbers, acknowledgments, and checksums, remain as context but are not reconstruction targets. Source and destination IP addresses directly identify endpoints and are zeroed before input processing. This assignment does not use downstream labels; Table 1 summarizes the treatment. The resulting columns still have different value domains, which TAE maps into vectors of a common dimension.

Table 1: Field training roles and treatment for the 51-field configuration.

Role	Count	Criterion and examples	Input and pretraining treatment
Generalizable (G)	38	Behavioral fields with stable protocol meanings across networks: lengths, delays, flags, ports, and TLS handshake attributes	Retained; eligible for packet/field masking and reconstruction loss
Random/ operational (R)	11	Connection-state, fragmentation, or integrity values without stable reconstruction targets: sequence numbers, acknowledgments, and checksums	Retained as context; neither masked nor reconstructed
Non-generalizable (N)	2	Endpoint identifiers `ip.src` and `ip.dst`	Zeroed in the original value domain before standardization; neither masked nor reconstructed

3.2 Type-aware embeddings (TAE)

Protocol fields may have similar numerical forms but different meanings. Low-cardinality codes, such as protocol numbers, have no continuous quantitative relationship; binary fields describe states, while time differences and payload lengths have nonnegative long-tailed distributions. TAE therefore chooses mappings by value domain instead of applying the same scalar projection to every field. Inspired by the distinction between categorical and numerical features in tabular Transformers (Gorishniy et al. 2021), it uses four mapping paths:

- *Binary fields* (12 fields, including direction, IP/TCP flags, and reserved bits) use Embedding$(2, d)$ for the two states.
- *Categorical fields* use Embedding(K_i, d) to assign separate embeddings to valid codes. This path covers three low-cardinality fields: `ip.proto`, `icmp.type`, and `icmp.code`.
- *Nonnegative long-tailed fields* (two fields: time difference and payload length) use $\log(1 + x)$ followed by Linear$(1, d)$ to compress the long tail.
- *General scalar fields* (all remaining fields, including numerical quantities and scalar-encoded high-cardinality codes) use standardized values followed by Linear$(1, d)$.

Categorical lookup assigns independent vectors to codes. For example, `ip.proto`=6 and 17 denote TCP and UDP, rather than quantities separated by a meaningful numerical distance. Logarithmic compression reduces the influence of extreme time differences and payload lengths. High-cardinality or sparse codes, including ports, TLS versions, and cipher suites, instead use scalar linear projections to avoid large vocabularies.

Because the input matrix is standardized, original values of binary, categorical, and nonnegative long-tailed fields are recovered using the saved means and standard deviations before transformation. General scalar fields use their Z-score values directly. Every task reuses these mappings and statistics so that each field enters a consistent vector space.

The matrix also requires identities for packet order and field columns. TAE outputs are added to two sets of learnable positional embeddings:

$$\mathbf{e}_{t,n} = \mathrm{TAE}(X_{t,n}) + \mathbf{p}_t^{\mathrm{time}} + \mathbf{p}_n^{\mathrm{field}}, \tag{1}$$

where d is the embedding dimension, $\mathbf{p}_t^{\mathrm{time}}$ denotes packet position, and $\mathbf{p}_n^{\mathrm{field}}$ identifies the fixed field column. During pretraining, masking follows positional encoding:

$$\mathbf{z}_{t,n} = (1 - M_{t,n})\mathbf{e}_{t,n} + M_{t,n}\mathbf{m}, \tag{2}$$

where $M_{t,n} \in \{0, 1\}$ is the mask indicator and $\mathbf{m}$ is a shared learnable mask vector. The entire selected embedding, including its explicit positional vectors, is replaced. Unmasked positions retain their field values and positional embeddings. Downstream adaptation applies no pretraining masks, so $\mathbf{z}_{t,n} = \mathbf{e}_{t,n}$. Section 3.6 defines mask sampling.

The resulting packet–field positions have a common embedding dimension, but isolated positions cannot yet describe changes between neighboring packets. We therefore introduce temporal convolution before attention.

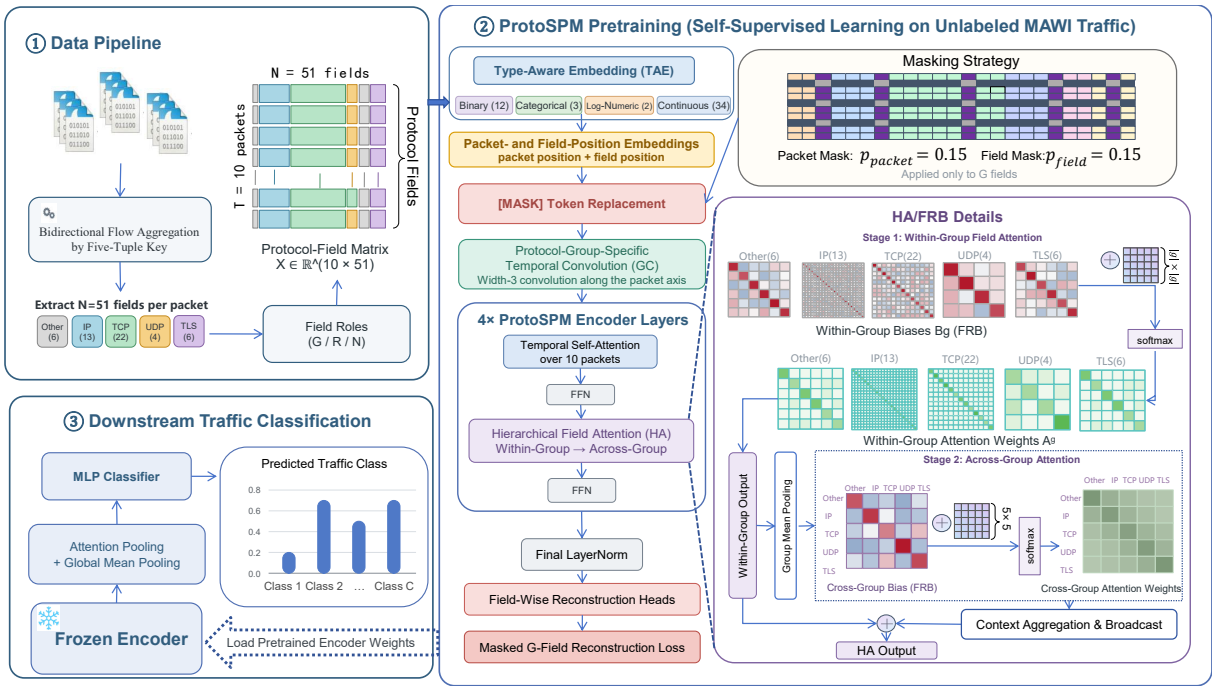


Fig. 1 Overview of ProtoSPM. Bidirectional flows become 10×51 field matrices, followed by TAE, two-dimensional positional embeddings, GC, and four encoder layers. Pretraining masks replace embeddings after positional encoding and before GC. Each layer applies temporal attention, an FFN, HA, and a second application of the same FFN. HA combines within-group interaction with pooled and broadcast across-group context; FRB adjusts relation scores. Pretraining uses field-wise reconstruction heads. Downstream classification removes these heads, freezes the encoder, and trains only attention pooling and an MLP

3.3 Protocol-group-specific temporal convolution (GC)

TCP handshake flags, windows, and lengths change with connection stages, whereas TLS handshake attributes concentrate in early packets. GC extracts these short-range changes before global attention by convolving each field along its packet sequence and sharing filters within protocol groups. Each group has an independent $\mathrm{Conv1d}(d, d, k = 3, \mathrm{pad} = 1)$. For field n in group $g(n)$, the computation is

$$\tilde{\mathbf{z}}_{t,n} = \mathbf{z}_{t,n} + \mathrm{Conv}_{g(n)}\big([\mathbf{z}_{t-1,n}, \mathbf{z}_{t,n}, \mathbf{z}_{t+1,n}]\big). \tag{3}$$

Zero padding preserves sequence length, and a residual connection retains the input state. Fields in the same group share filters, while different groups learn separate responses. Five filter sets thus cover all fields instead of assigning separate convolution parameters to each of the 51 columns. The output preserves each field's temporal trajectory. Temporal attention subsequently extends its context, and HA exchanges information between fields.

3.4 Two-stage hierarchical field attention (HA)

GC captures local temporal features, but classification also requires changes across the full input window and relationships between fields. Each encoder layer therefore applies temporal self-attention, a feed-forward network, hierarchical field attention, and a second feed-forward computation in sequence. The two feed-forward computations share the same FFN parameters.

Temporal self-attention. Standard multihead self-attention ($h = 8$) operates separately on each field across T time steps, capturing longer-range changes between packets. It does not mix different fields; the next stage handles field interaction.

Hierarchical field attention. Fields within a protocol jointly describe headers and connection state, while across-protocol interactions exchange information from different layers. HA first performs within-group interaction and then communicates through group summaries, replacing flat $N \times N$ field attention with two stages.

Within-group attention maintains independent projections $\{\mathbf{W}_Q^g, \mathbf{W}_K^g, \mathbf{W}_V^g, \mathbf{W}_O^g\}$ for each group g. It also uses a learnable additive bias $\mathbf{B}^g \in \mathbb{R}^{|g|\times|g|}$:

$$\mathrm{Attn}^g(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \mathrm{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_h}} + \mathbf{B}^g\right)\mathbf{V}, \tag{4}$$

where $d_h = d/h$ is the dimension of each attention head. Every encoder layer learns its own within-group and across-group biases; heads in the same layer share the corresponding bias matrices.

Across-group attention mean-pools the within-group outputs to obtain group summaries. The same summary set generates queries, keys, and values for self-attention with a learnable bias $\mathbf{B}^{\text{cross}} \in \mathbb{R}^{5\times5}$. The resulting across-group context is broadcast back to fields within each group. The implementation averages the fixed columns in each group without reweighting them according to field presence in the current packet. Local aggregation followed by global interaction aligns the attention structure with groups defined by protocol and function (Yang et al. 2016).

Feed-forward network. Both temporal self-attention and HA are followed by feed-forward computation; LayerNorm and residual connections wrap the operations. The FFN expands dimensions as $d \to 4d \to d$.

HA places within-group and across-group dependencies in different attention subspaces. For example, `tcp.hdr_len` and `tcp.options_len` interact directly within TCP, whereas IP, TCP, and TLS fields communicate through five summaries. Flat attention over 51 fields requires $51^2 = 2601$ field scores per time step. The within-group and across-group score count in HA is

$$\sum_{g\in\mathcal{G}} |g|^2 + |\mathcal{G}|^2 = 6^2 + 13^2 + 22^2 + 4^2 + 6^2 + 5^2 = 766, \tag{5}$$

approximately 70.5% fewer than flat interaction. This count measures attention-score matrix sizes and captures the structural and computational constraint introduced by protocol grouping. HA defines the interaction hierarchy, while FRB further models preferences between directed field pairs within that hierarchy.

3.5 Zero-initialized field-relation biases (FRB)

Content-dependent $QK^\top$ scores vary between packets, but some field relationships recur across inputs. FRB introduces shared parameters for directed field pairs to adjust content matching in HA. Relation-aware attention provides a basis for this design (Shaw et al. 2018). ProtoSPM uses within-group biases $\{\mathbf{B}^g\}$ and an across-group bias $\mathbf{B}^{\text{cross}}$, all initialized to zero and learned through pretraining gradients.

Biases enter attention scores additively. Given a content score s_{ij}, the resulting weight is proportional to $\exp(s_{ij})\exp(B_{ij})$. Thus, B_{ij} can strengthen or suppress matching in the corresponding direction. Only 766 bias parameters are required per layer, totaling 3,064 across four layers. Relation biases preserve shared interaction preferences, while content scores adjust responses to individual packets. The embeddings, convolution, and attention structures learn jointly through masked reconstruction; relation parameters remain fixed with the encoder during downstream adaptation.

3.6 Masked pretraining objective

We reconstruct masked field values to learn temporal and field context from unlabeled traffic. Masked modeling has been applied to language, vision, and tabular data (Devlin et al. 2019; He et al. 2022; Yoon et al. 2020). For protocol-field matrices, masks are sampled along both packet positions and field columns so that reconstruction uses the remaining context.

For each example, time-step masks are sampled with $p_{\text{packet}} = 0.15$ and field-column masks with $p_{\text{field}} = 0.15$. Their union is intersected with the generalizable-field set. The expected masking rate for a generalizable position is therefore $1 - (1 - 0.15)^2 = 27.75\%$. Selected positions are replaced with the mask vector as described in Section 3.2. Field-wise reconstruction heads map encoder outputs to scalar predictions. Predictions and targets remain in standardized numerical space to balance loss scales across fields.

The implementation uses a field-balanced reconstruction loss. Let $\mathcal{U}$ be the generalizable-field set and $M_{b,t,n}$ indicate whether position (t, n) in example b is masked:

$$\mathcal{L}_{\text{rec}} = \frac{1}{|\mathcal{U}|} \sum_{n \in \mathcal{U}} \frac{\sum_{b,t} M_{b,t,n} \left(\hat{X}_{b,t,n} - X_{b,t,n} \right)^2}{\sum_{b,t} M_{b,t,n} + \epsilon}. \tag{6}$$

Here ϵ prevents division by zero when a field has no masked positions in a batch. Random/operational fields remain as context but are excluded from masking and reconstruction loss; non-generalizable fields are also excluded. After pretraining, reconstruction heads are removed and the encoder becomes a shared feature extractor for downstream tasks.

3.7 Downstream classification heads

The pretrained encoder maps field matrices to contextual representations using fixed parameters θ^*. Each task learns separate pooling and classification parameters ϕ_k:

$$\mathbf{H} = f_{\theta^*}(\mathbf{X}), \qquad \hat{\mathbf{y}}^{(k)} = h_{\phi_k}(\mathbf{H}). \tag{7}$$

Here $\mathbf{H} \in \mathbb{R}^{T \times N \times d}$, and k denotes the target classification task. All downstream tasks load the same pretrained encoder and optimize only ϕ_k for their respective label sets. The encoder supplies traffic context, while each head learns the mapping to target classes.

Because tasks may emphasize different packets and fields, the head uses content-aware attention pooling. A two-layer scoring network assigns a scalar weight to every packet–field position, followed by softmax normalization and a weighted sum. Adding the global mean-pooled representation yields a d-dimensional flow vector. A two-layer MLP ($d \to d/2 \to C_k$) then predicts among C_k classes. Target labels therefore train context aggregation and classification without updating encoder parameters.

4 Experiments

4.1 Experimental setup

Pretraining data

We use public MAWI backbone-link traffic to learn shared representations from unlabeled data naturally accumulated in real networks (Cho et al. 2000). Its packet-level records support consistent field extraction and bidirectional-flow construction without downstream task labels. The traffic was collected on January 1, 2025, yielding approximately 3.57 million TCP/UDP flows after parsing and aggregation. Pretraining uses a flow-level 70% MAWI training partition, with 10% of that partition reserved for validation and early stopping. Inputs comprise plaintext protocol headers, packet-level behavioral measurements, and parseable TLS handshake metadata. No decryption keys or decrypted application payloads are used. MAWI and the downstream datasets originate from different collection institutions and periods; pretraining does not use downstream labels.

Downstream datasets

We select three complementary scenarios for application and website identification. ISCX-VPN-app covers multiple application behaviors, while TLS-120 provides a larger label space for fine-grained website recognition. CipherSpectrum adds a different collection source, balanced classes, and multiple modern cipher-suite configurations. Differences in labels, class distributions, and traffic sources provide settings for evaluating fixed-encoder adaptation across classification label spaces.

- **ISCX-VPN-app** (16 classes) is derived from ISCX VPN-nonVPN and contains traffic from 16 applications in VPN and non-VPN scenarios (Draper-Gil et al. 2016; Zhao et al. 2025).
- **TLS-120** (120 classes) is derived from CSTNET-TLS1.3 traffic, with labels corresponding to 120 websites (Lin et al. 2022; Zhao et al. 2025). It tests fixed representations in fine-grained, multiclass website identification.
- **CipherSpectrum** (41 classes) is public TLS 1.3 website-identification traffic covering AES-128-GCM, AES-256-GCM, and ChaCha20-Poly1305 (Wickramasinghe et al. 2025). We use the three separate

cipher-suite directories in the released package. After label organization, they contain 123,000 flows, with 3,000 per class. A details release statistics and label organization.

ISCX-VPN-app and TLS-120 retain fixed, flow-stratified train/validation/test splits of 70/10/20. CipherSpectrum captures contain multiple flows, so capture groups are assigned together in approximately the same proportions. This prevents correlated flows from the same source capture from crossing partitions. Dataset splits remain fixed across controls. ProtoSPM reuses the pretrained encoder and normalization statistics, processes inputs according to Section 3.1, and trains independent task heads.

Table 2: Dataset statistics. ProtoSPM uses protocol-field matrices with $T = 10$ and $N = 51$. MAWI is used only for unlabeled pretraining.

Dataset	Purpose	Flows	Classes	Min./mean class size
ISCX-VPN-app	Downstream	34,516	16	73 / 2,157
TLS-120	Downstream	44,128	120	16 / 368
CipherSpectrum	Downstream	123,000	41	3,000 / 3,000
MAWI (2025-01-01)	Pretraining	∼3.57M	—	—

Architecture hyperparameters

Table 3 lists the encoder configuration. All structural controls retain the same input window and field order; the corresponding experiments describe individual variants.

Table 3: ProtoSPM architecture hyperparameters. The downstream encoder contains 10.70M parameters; the complete model, including reconstruction heads, contains 12.03M.

Hyperparameter	Value	Description
T (packets)	10	First ten packets of each bidirectional flow
N (fields)	51	Columns organized by protocol origin and function
d_{model}	256	Embedding and Transformer hidden dimension
nhead	8	Number of attention heads
L (layers)	4	Number of stacked encoder layers
d_{ff}	1,024	Intermediate FFN dimension

Baselines

We compare ET-BERT, YaTC, NetMamba, TrafficFormer, and netFound using their public implementations' native inputs and flow-level aggregation interfaces. Pretrained encoders remain fixed during downstream adaptation, and only classification heads are trained. Table 6 presents their results alongside ProtoSPM. To examine protocol structure, we also construct Plain-Field, which uses identical field inputs and a similar parameter count.

On CipherSpectrum, the five baselines use the same source flows and capture-group splits, processing the first five packets through their native interfaces. For byte and image inputs, source/destination IP addresses and parseable SNI hostname bytes are zeroed before encoding; lengths and ports are retained. The official netFound-v1 token fields exclude IP addresses, ports, and SNI hostnames; original addresses determine only direction and relative sequence numbers. ProtoSPM uses ten-packet field matrices and includes SNI length rather than hostname content. ET-BERT, YaTC, NetMamba, and netFound cache frozen features in evaluation mode. TrafficFormer and ProtoSPM retain encoder dropout during classification-head training.

Frozen-encoder evaluation

Downstream evaluation fixes each pretrained encoder and trains only pooling and classification heads to assess the utility of pretrained representations. ProtoSPM reuses the same encoder weights across all three datasets, with a separate head for each task. Unlike end-to-end fine-tuning, this procedure does not

update encoder parameters using downstream labels. The randomly initialized frozen control uses the same architecture and downstream settings without loading pretrained weights.

Training settings

Pretraining uses AdamW with an initial learning rate of 1×10^{-4}, weight decay of 0.05, and batch size of 256. Five linear-warmup epochs precede cosine learning-rate decay, with at most 100 training epochs. Packet- and field-mask probabilities are both 0.15; training stops after ten epochs without improvement in validation loss. ProtoSPM downstream heads follow Section 3.7 and use class-weighted cross-entropy with AdamW at a learning rate of 5×10^{-4}. They train for 50 epochs with batch size 16. Encoder dropout (0.1) remains active during head training and is disabled for validation and testing. Checkpoints are selected by validation accuracy. Downstream runs use seeds $\{42, 2024, 3407\}$; test results report means and sample standard deviations. Accuracy measures the fraction of correctly classified flows. Macro-F1 is the arithmetic mean of per-class F1 scores, weighting classes equally.

4.2 Frozen-encoder adaptation and the contribution of pretraining

Table 4 reports adaptation of the same pretrained encoder to application and website classification. Macro-F1 reaches 61.29% on ISCX-VPN-app application identification and 67.98% and 71.28% on TLS-120 and CipherSpectrum website identification. Without updating encoder weights, separate task heads can use the shared representations to classify their respective label sets.

Table 4: Main frozen-encoder results for ProtoSPM. Values are means ± sample standard deviations over three downstream runs.

Dataset	Accuracy	Macro-F1	Test flows
ISCX-VPN-app (16 classes)	$80.24 \pm 1.07\%$	$61.29 \pm 1.54\%$	6,904
TLS-120 (120 classes)	$71.26 \pm 1.05\%$	$67.98 \pm 1.23\%$	8,825
CipherSpectrum (41 classes)	$72.30 \pm 1.13\%$	$71.28 \pm 1.20\%$	24,860

Control for pretrained weights. We construct an identical, randomly initialized frozen encoder to distinguish unlabeled-pretraining gains from the benefits of field input and network architecture alone. Both groups use identical fields and downstream training settings, with encoder weights fixed.

Table 5: Control for pretrained weights. Both groups share the architecture and downstream settings; the random encoder is reinitialized for each run. Values are means ± sample standard deviations over three runs (%). Gains are computed from unrounded means and expressed in percentage points.

Dataset	Random Acc.	Pretrained Acc.	Random M-F1	Pretrained M-F1	M-F1 gain
ISCX-VPN-app	71.83 ± 3.18	80.24 ± 1.07	47.00 ± 2.20	61.29 ± 1.54	+14.29
TLS-120	55.92 ± 6.10	71.26 ± 1.05	51.24 ± 6.67	67.98 ± 1.23	+16.75
CipherSpectrum	41.76 ± 1.37	72.30 ± 1.13	37.29 ± 1.50	71.28 ± 1.20	+33.99

The randomly initialized frozen encoder retains the field inputs and architecture but has not undergone unlabeled pretraining. Loading pretrained weights increases application-identification Macro-F1 by 14.29 points on ISCX-VPN-app. For website identification, the increases are 16.75 points on TLS-120 and 33.99 points on CipherSpectrum. With both encoders fixed and classification heads trained identically, these gains support additional discriminative information learned through unlabeled pretraining across label spaces.

4.3 Pretrained-model comparison through native interfaces

Table 6 summarizes frozen adaptation of five pretrained baselines across the three datasets. The 12.03M-parameter ProtoSPM achieves the highest Macro-F1 in the table for each label space. Scores are $61.29 \pm 1.54\%$ on ISCX-VPN-app, $67.98 \pm 1.23\%$ on TLS-120, and $71.28 \pm 1.20\%$ on CipherSpectrum. Figure 2(b) summarizes cross-task performance using the equally weighted arithmetic mean of the three Macro-F1 scores and shows model size. ProtoSPM averages 66.85%, compared with 33.51% for TrafficFormer, the

highest-scoring baseline on this aggregate. Together with the individual dataset results, this comparison shows effective frozen classification representations at a moderate model size.

Table 6: Frozen transfer through native model interfaces on three datasets. Accuracy and Macro-F1 are means ± sample standard deviations over three classification-head seeds (%). Parameter counts are approximate model sizes.

Method	Parameters	ISCX-VPN-app		TLS-120		CipherSpectrum	
		Accuracy	Macro-F1	Accuracy	Macro-F1	Accuracy	Macro-F1
ET-BERT	∼178.86M	47.38 ± 1.46	33.81 ± 2.49	20.74 ± 0.89	13.74 ± 0.92	13.22 ± 0.35	9.51 ± 0.99
YaTC	∼2.33M	52.35 ± 0.70	43.19 ± 0.98	38.01 ± 0.19	31.92 ± 0.27	9.88 ± 0.38	5.48 ± 0.58
NetMamba	∼2.17M	44.54 ± 0.75	33.33 ± 1.47	23.89 ± 0.57	17.70 ± 0.39	9.76 ± 0.21	5.70 ± 0.48
TrafficFormer	∼178.87M	63.98 ± 1.46	54.33 ± 2.08	46.25 ± 0.20	41.59 ± 0.21	9.12 ± 0.19	4.60 ± 0.41
netFound	∼640M	34.18 ± 0.91	29.19 ± 1.43	37.15 ± 0.52	30.44 ± 0.33	38.79 ± 0.58	35.68 ± 1.18
ProtoSPM	**12.03M**	$\mathbf{80.24 \pm 1.07}$	$\mathbf{61.29 \pm 1.54}$	$\mathbf{71.26 \pm 1.05}$	$\mathbf{67.98 \pm 1.23}$	$\mathbf{72.30 \pm 1.13}$	$\mathbf{71.28 \pm 1.20}$

On CipherSpectrum, ProtoSPM reaches 71.28% frozen Macro-F1, exceeding the strongest baseline, netFound, by 35.60 points. Together with the other datasets, this result shows that one ProtoSPM encoder provides useful representations for application and website identification without backbone updates. Different heads can exploit pretrained features without fine-tuning the entire encoder for each task. The following same-input, near-parameter-matched Plain-Field comparison examines the contribution of protocol-structured encoding.

4.4 Same-input, near-parameter-matched structural control

Cross-model comparisons show frozen adaptation by different pretrained systems. To examine protocol-structured encoding, Plain-Field uses the same field matrices, MAWI pretraining data, and masking objective as ProtoSPM. It disables the four protocol-structure modules, uses plain dual-axis attention, and expands the FFN dimension to 4096. Plain-Field contains 11.89M parameters, compared with 12.03M in the full ProtoSPM model. The two models are pretrained separately and evaluated using identical task splits and classification-head settings.

Table 7: Same-input, near-parameter-matched structural control. Values are means ± sample standard deviations over three downstream seeds (%). Gains are computed from unrounded means in Macro-F1 percentage points.

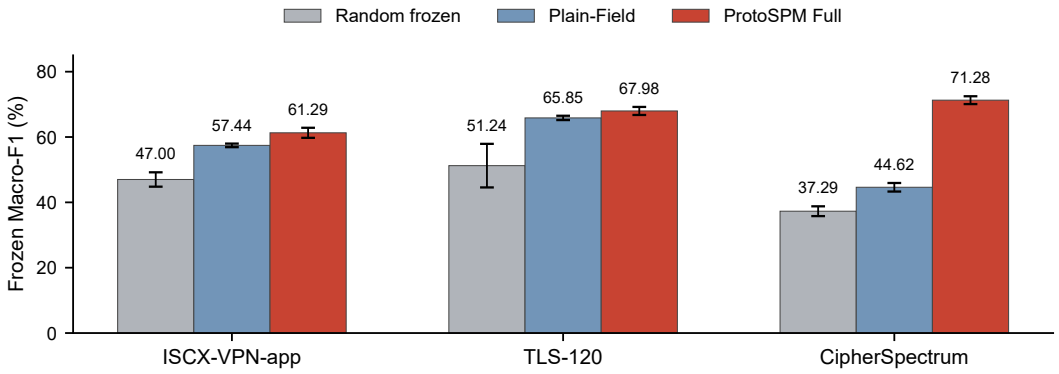
Dataset	Plain Acc.	Full Acc.	Plain M-F1	Full M-F1	Gain
ISCX-VPN-app	79.95 ± 0.83	80.24 ± 1.07	57.44 ± 0.55	61.29 ± 1.54	+3.85
TLS-120	69.55 ± 0.49	71.26 ± 1.05	65.85 ± 0.66	67.98 ± 1.23	+2.13
CipherSpectrum	47.49 ± 1.05	72.30 ± 1.13	44.62 ± 1.31	71.28 ± 1.20	+26.65

For application identification, ProtoSPM exceeds Plain-Field by 3.85 Macro-F1 points. The website-identification gains are 2.13 points on TLS-120 and 26.65 points on CipherSpectrum. With identical field inputs, pretraining data, and similar parameter counts, these results support the combined protocol-structured design for frozen representations. Component removals next examine the roles of its modules.

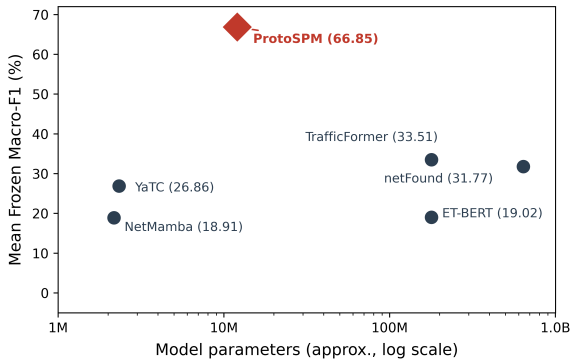
4.5 Component-removal experiments

Table 8 examines the four component-removal variants. The four component removals are evaluated on all three datasets. Every variant is pretrained independently rather than disabling a trained module only during downstream evaluation.

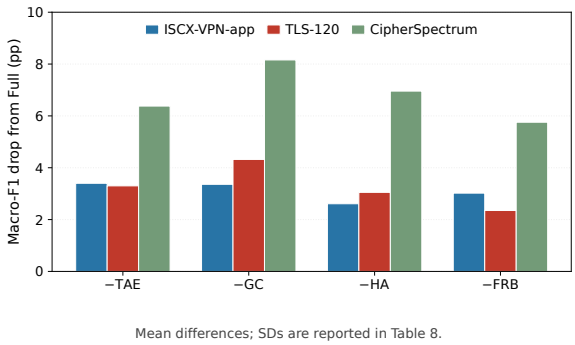
Removing any component reduces mean Macro-F1 on all three datasets. On CipherSpectrum, removing TAE, GC, HA, and FRB decreases Macro-F1 by 6.38, 8.16, 6.96, and 5.76 points, respectively; GC has the largest effect. The four designs improve results across application and website label spaces. This supports heterogeneous-field mapping, local packet dynamics, and relation-based interactions for frozen representations. Table 10 reports variant sizes, including the substantial parameter reduction in −HA. Plain-Field tests the combined structure at a comparable parameter count.



(a) Frozen Macro-F1 of pretrained Full, same-input Plain-Field, and randomly initialized frozen encoders on three datasets. Error bars show sample standard deviations over three downstream seeds



(b) Model parameters versus mean frozen Macro-F1 across three datasets



(c) Macro-F1 decreases after four component removals on three datasets

Fig. 2 Frozen representations and structural controls for compact encoders. (a) The same MAWI-pretrained Full encoder serves application and website label spaces, including CipherSpectrum, alongside random and near-parameter-matched Plain-Field controls. Values match Tables 4, 5, and 7. (b) Approximate model parameters and equally weighted mean frozen Macro-F1 across the three datasets. The diamond denotes ProtoSPM and circles denote baselines; Table 6 reports per-task means and sample standard deviations. (c) Mean Macro-F1 decreases relative to Full after each of the four component removals on all three datasets; Table 8 reports three-seed means and sample standard deviations

Table 8: Independently pretrained component removals. Every variant is pretrained from scratch and evaluated with the same downstream protocol. Values are means ± sample standard deviations over three downstream seeds (%).

Variant	ISCX-VPN-app		TLS-120		CipherSpectrum	
	Accuracy	Macro-F1	Accuracy	Macro-F1	Accuracy	Macro-F1
ProtoSPM Full	80.24 ± 1.07	61.29 ± 1.54	71.26 ± 1.05	67.98 ± 1.23	72.30 ± 1.13	71.28 ± 1.20
−TAE	78.70 ± 1.04	57.89 ± 1.94	68.14 ± 0.65	64.68 ± 0.53	66.26 ± 0.70	64.90 ± 0.67
−GC	79.24 ± 0.67	57.93 ± 2.00	67.43 ± 0.67	63.66 ± 0.80	64.87 ± 1.80	63.12 ± 1.93
−HA	79.15 ± 1.04	58.68 ± 1.89	68.78 ± 0.77	64.93 ± 0.67	67.82 ± 0.51	64.32 ± 0.84
−FRB	76.37 ± 1.05	58.27 ± 0.40	69.20 ± 0.71	65.63 ± 0.75	66.82 ± 0.30	65.52 ± 0.34

Field-relation bias analysis

Figure 3 visualizes TCP within-group field-relation biases learned during MAWI pretraining. Each cell shows the mean absolute bias across four encoder layers for the corresponding direction; diagonal cells are omitted. Direction follows the query–key roles in attention. The strongest relationship occurs between `tcp.options_len` and `tcp.hdr_len`. The mean absolute bias is 0.905 from the former to the latter and 0.557 in the reverse direction. The implementation represents `tcp.options_len` as the number of TCP option items. Data Offset specifies header length including the options area, linking these fields structurally in packets containing options (Eddy 2022). The bias from `tcp.options.wscale` to `tcp.window_size` is 0.325, consistent with window-scaling semantics (Borman et al. 2014). These patterns indicate selective modulation with protocol-related interpretations; the component-removal experiment in Table 8 provides performance evidence.

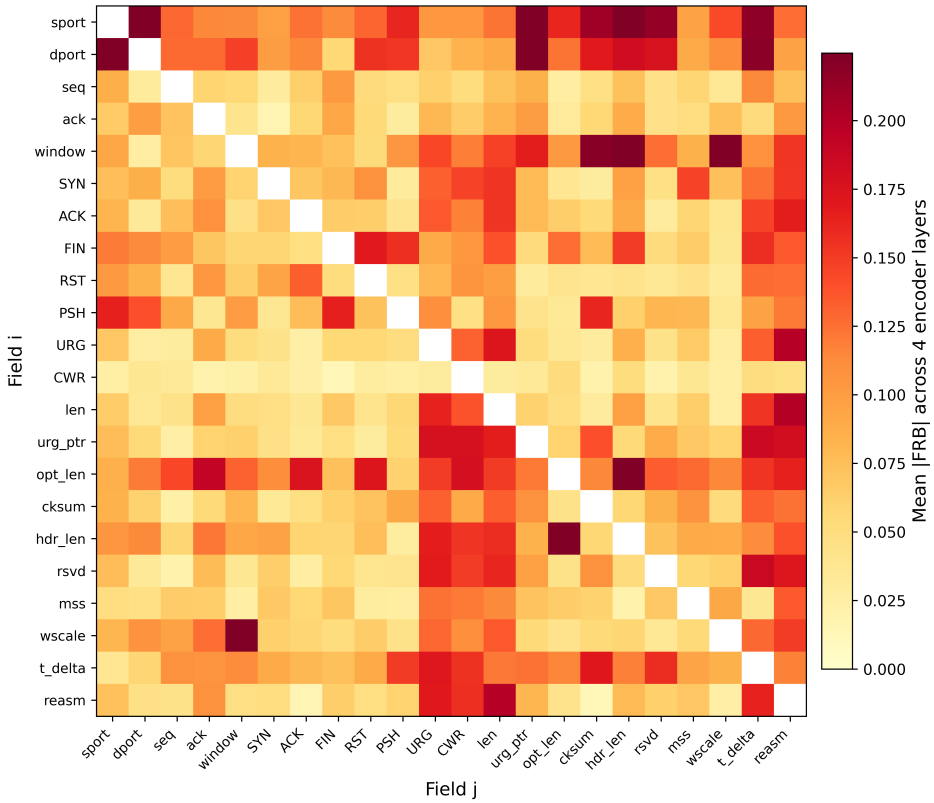


Fig. 3 TCP within-group field-relation biases in ProtoSPM. Each off-diagonal cell shows the mean absolute directed FRB value across four encoder layers. Darker colors indicate greater modulation magnitude, not its sign. The color scale is capped at the 98th percentile of off-diagonal values, with larger values shown at saturation. FRB is learned from zero initialization

4.6 Field-sensitivity analysis

We test whether frozen classification mainly depends on potentially shortcut-related fields by zeroing SNI length, TLS handshake fields, or random/operational fields. Classification heads are retrained while the pretrained encoder and dataset splits remain unchanged. Table 9 reports the resulting performance changes.

After zeroing SNI length or all TLS handshake fields, Macro-F1 changes by at most 0.28 points on either dataset. After head retraining, classification therefore does not depend mainly on these fields; other encoded traffic information still supports task adaptation. Random/operational fields have dataset-dependent effects. Zeroing them decreases ISCX-VPN-app Macro-F1 by 2.08 points but increases TLS-120 Macro-F1 by 0.98 points.

As a further port-shortcut control, we zero TCP/UDP source and destination ports while retaining the remaining settings. For the representative seed-42 run, accuracy/Macro-F1 is 79.81%/59.53% on ISCX-VPN-app and 70.58%/66.81% on TLS-120. Relative to Full with the same seed, changes are -0.56/ -0.51

Table 9: Field-sensitivity analysis. Values are means ± sample standard deviations over three downstream seeds. All settings match Full except that the designated fields are zeroed.

Input treatment	ISCX Acc.	ISCX Macro-F1	TLS Acc.	TLS Macro-F1
Full	$80.24 \pm 1.07\%$	$61.29 \pm 1.54\%$	$71.26 \pm 1.05\%$	$67.98 \pm 1.23\%$
Zero SNI length	$80.10 \pm 0.99\%$	$61.49 \pm 1.52\%$	$71.23 \pm 1.03\%$	$67.94 \pm 1.18\%$
Zero TLS fields	$80.42 \pm 1.34\%$	$61.09 \pm 1.74\%$	$71.15 \pm 0.79\%$	$67.70 \pm 1.02\%$
Zero random/operational fields	$80.15 \pm 0.27\%$	$59.21 \pm 0.60\%$	$72.53 \pm 0.48\%$	$68.96 \pm 0.63\%$

and $+0.53/+0.24$ percentage points, respectively. These results do not identify port fields as the main source of current performance.

4.7 Model size and encoder inference efficiency

Encoder reuse avoids repeated backbone optimization, while model size and inference overhead determine shared-encoder operating costs. The complete ProtoSPM pretraining model contains 12.03M parameters; removing reconstruction heads leaves a 10.70M-parameter encoder. Table 10 reports parameter counts for Full and component-removal variants. HA reduces field-attention scores per time step from 2601 to 766, although its separate within-group and across-group projections add parameters. Plain-Field compares the combined design at a similar model size. Table 11 further reports the shared encoder's runtime costs.

Table 10: Parameter counts from the current implementation. Complete models include pretraining reconstruction heads; encoders are the frozen downstream components. −HA uses one flat field-attention module. −FRB disables bias addition but still instantiates bias matrices, retaining Full's parameter count.

Configuration	Complete model	Encoder
Full	12.025M	10.704M
−TAE	11.830M	10.509M
−GC	11.041M	9.720M
−HA	6.769M	5.448M
−FRB	12.025M	10.704M

Runtime efficiency

We measure frozen-encoder FP32 inference on an NVIDIA GeForce RTX 4070 Laptop GPU (8 GB), using PyTorch 2.6.0 and CUDA 12.4. Inputs are 10×51 field matrices. For each batch size, we perform 20 warm-up passes and report the median latency over 100 encoder forward passes. Counting linear layers, temporal convolution, and attention matrix multiplications gives approximately 3.39 G MACs per flow. Counting each multiply–accumulate as two floating-point operations yields approximately 6.78 GFLOPs. Table 11 reports runtime measurements. Increasing batch size from 1 to 64 raises throughput from 45.22 to 588.18 flows/s. The downstream batch size of 16 corresponds to 450.87 flows/s and 176.38 MiB peak allocated GPU memory.

Table 11: Runtime efficiency of the frozen ProtoSPM encoder. Latency is the median of 100 FP32 forward passes; throughput is computed from unrounded latency. Memory denotes peak allocated GPU memory recorded by PyTorch.

Batch size	Latency (ms/batch)	Throughput (flows/s)	Peak memory (MiB)
1	22.12	45.22	61.14
16	35.49	450.87	176.38
64	108.81	588.18	543.80

5 Discussion

5.1 Unlabeled pretraining and reuse of fixed representations

A pretrained traffic encoder's value extends beyond fine-tuned classification to whether different tasks can continue using its fixed representations. ProtoSPM learns from unlabeled backbone traffic and adapts to application and website labels through independent heads, without task-specific backbone updates. Compared with full fine-tuning, this approach concentrates target-label supervision on feature aggregation and classification. The encoder retains unified weights, directly testing the utility of its existing traffic representations (Zhao et al. 2025; Luxemburk et al. 2026).

The randomly initialized frozen and Plain-Field controls together show that field inputs alone do not guarantee effective reusable representations. The former retains the inputs and architecture but lacks unlabeled pretraining. The latter retains the fields, pretraining corpus, and similar model size but uses plain field encoding. Both score below Full, supporting a joint role for pretraining and protocol-structured encoding. Masked reconstruction learns context from unlabeled traffic, while field types, temporal sharing, and hierarchical interaction organize how that context is encoded.

Structural gains vary across classification scenarios. Removing TAE, GC, HA, or FRB reduces Macro-F1 on all three datasets, but the magnitude varies; the Plain-Field gap is also larger on CipherSpectrum. This suggests that shared representations should retain multiple traffic cues for task heads to select and combine, rather than predetermine importance for one label set. Protocol grouping and relation biases provide an organizational basis for this encoding. More targeted controls could further relate particular dependencies to tasks and collection conditions.

5.2 Adaptation and deployment of a shared encoder

Real networks continuously accumulate unlabeled traffic, whereas application and website labels generally require dedicated collection or verification. Learning traffic context during unlabeled pretraining confines subsequent supervised optimization to task heads, offering an adaptation route as classification labels expand. Multiple tasks can share one set of encoder weights while maintaining smaller, separate heads. This reduces the need to optimize and store task-specific backbones repeatedly.

Model size and inference costs determine the shared backbone's operating requirements. Section 4.7 reports parameter counts, latency, throughput, and memory measurements to inform batch-size selection and resource allocation. Protocol-structured encoding produces effective frozen representations at a relatively small model size. Encoder reuse therefore has both a demonstrated classification basis and measured model-resource requirements.

5.3 Limitations

The fixed-width input matrix uses the same original-domain zero value for short-flow padding, inapplicable fields, and parsing failures; valid fields can also equal zero. Distinguishing these sources could represent protocol context more accurately. Random/operational fields also have different effects across datasets. Their treatment could be studied further using field roles and connection state.

The evaluated reuse concerns closed-set application and website identification, and each task head still requires target-label training. Evaluation under temporal changes, collection-environment shifts, and unknown classes would further characterize the scope of fixed representations. Multiple pretraining initializations could examine shared-encoder stability. These directions address distribution changes during continued deployment and the repeatability of pretraining outcomes, respectively.

6 Conclusion

To address repeated encoder optimization across traffic classification tasks, we proposed ProtoSPM, a protocol-structured pretraining model that incorporates field-value semantics, packet dynamics, and protocol dependencies into a shared encoder and learns reusable representations through masked reconstruction of unlabeled traffic. Experiments show that the same MAWI-pretrained encoder supports application and website identification with fixed weights and independently trained task heads, achieving Macro-F1 scores of 61.29%, 67.98%, and 71.28% on ISCX-VPN-app, TLS-120, and CipherSpectrum, respectively. The randomly initialized frozen control shows that unlabeled pretraining provides useful representations for downstream classification; the same-input, near-parameter-matched Plain-Field control and component ablations further support the benefits of protocol-structured encoding. These results

demonstrate that the 12.03M-parameter ProtoSPM can convert unlabeled real-network traffic into shared features useful for different classification tasks, providing a practical approach to encrypted traffic classification without task-specific backbone updates.

Statements and Declarations

Funding

The authors received no specific funding for this work.

Competing interests

The authors declare that they have no relevant financial or non-financial interests to disclose.

Author contributions

Wenlei Dai contributed to methodology, software development, investigation, formal analysis, and drafting the manuscript. Qin Luo and Peng Wu contributed to supervision and manuscript review and editing.

Data availability

The data supporting the findings of this study are available from the corresponding author on reasonable request, subject to the access conditions and licenses of the original datasets.

Code availability

The code and pretrained model checkpoints are not publicly archived at present and can be made available by the corresponding author on reasonable request.

Appendix A The 51-field configuration and data preprocessing

Flow aggregation and preprocessing

Scapy reads IPv4 packets in capture order. Dataset construction retains TCP/UDP packets but excludes packets with broadcast, multicast, or unspecified destination addresses, as well as DHCP packets. Pure ACKs and TCP retransmissions remain in capture order. ISCX-VPN-app and TLS-120 are aggregated within each pcap by canonical bidirectional five-tuple; MAWI uses online ten-packet windows. Deduplication and windowing operate within file boundaries for each dataset. The input unit is a fixed-length five-tuple prefix, not a complete connection-lifecycle session. Direction is relative to the source of the first TCP SYN in the window, or otherwise the first IPv4 packet's source. Section 4.1 describes CipherSpectrum capture-group splitting and release organization. Its field matrices retain the common column order, training roles, and pretraining normalization statistics.

CipherSpectrum release and partitioning

The official CipherSpectrum webpage describes 40 domain classes and 120,000 sessions (https://cgi.cse.unsw.edu.au/~cspectrum/). The three separate cipher-suite directories used here contain 41 domain classes, with 1,000 flows per class per suite, totaling 123,000 flows. The mixed-configuration directory `mix` is excluded from the main experiment. Based on filenames and handshake verification, 1,000 samples in the release's `chacha20` directory are assigned to `getpocket.com`. This corrects a directory label rather than adding a class. Reported class and sample counts follow the organized flow manifests.

Each single-flow PCAP filename identifies its original capture through the prefix preceding `.pcap.`; this prefix defines the capture group. All flows in a group are assigned together. We concatenate fixed seed 42 with the group identifier, compute SHA-256, and map the result to $[0, 1)$. Thresholds of 0.7 and 0.8 assign training, validation, and test groups. The partitions contain 85,902, 12,238, and 24,860 flows, corresponding to 18,647, 2,667, and 5,337 capture groups, respectively. Group identifiers have no pairwise overlap, and every partition covers all 41 classes.

Field configuration

For binary inputs, TAE lookup can be written as $\mathbf{v}_0 + x(\mathbf{v}_1 - \mathbf{v}_0)$, equivalent to an affine mapping at the two possible values. The design combines mappings for binary states, low-cardinality codes, long-tailed numerical values, and general scalars.

Table A1 lists all 51 fields in input-column order. TAE types B, C, L, and X denote binary lookup, categorical lookup, linear projection after $\log(1 + x)$, and general scalar projection, respectively. Training roles G, R, and N denote generalizable, random/operational, and non-generalizable fields; Table 1 specifies their treatment.

Table A1: The 51 ProtoSPM fields, protocol groups, TAE types, and training roles.

No.	Field	Group	TAE type	Role
1	frame.time_delta	Other	L	G
2	direction	Other	B	G
3	ip.src	IP	X	N
4	ip.dst	IP	X	N
5	ip.ttl	IP	X	G
6	ip.len	IP	X	G
7	ip.id	IP	X	R
8	ip.flags.df	IP	B	G
9	ip.flags.mf	IP	B	G
10	ip.frag_offset	IP	X	R
11	ip.proto	IP	C	G
12	ip.checksum	IP	X	R
13	ip.hdr_len	IP	X	G
14	ip.tos	IP	X	R
15	ip.flags.rb	IP	B	R
16	tcp.sport	TCP	X	G
17	tcp.dport	TCP	X	G
18	tcp.seq	TCP	X	R
19	tcp.ack	TCP	X	R
20	tcp.window_size	TCP	X	G
21	tcp.flags.syn	TCP	B	G
22	tcp.flags.ack	TCP	B	G
23	tcp.flags.fin	TCP	B	G
24	tcp.flags.rst	TCP	B	G
25	tcp.flags.psh	TCP	B	G
26	tcp.flags.urg	TCP	B	G
27	tcp.flags.cwr	TCP	B	G
28	tcp.len	TCP	X	G
29	tcp.urgent_pointer	TCP	X	R
30	tcp.options_len	TCP	X	G
31	tcp.checksum	TCP	X	R
32	tcp.hdr_len	TCP	X	G
33	tcp.reserved	TCP	B	R
34	tcp.options.mss	TCP	X	G
35	tcp.options.wscale	TCP	X	G
36	tcp.time_delta	TCP	X	G
37	tcp.reassembled.length	TCP	X	G
38	udp.sport	UDP	X	G
39	udp.dport	UDP	X	G
40	udp.length	UDP	X	G
41	udp.checksum	UDP	X	R
42	payload_len	Other	L	G
43	icmp.type	Other	C	G
44	icmp.code	Other	C	G
45	frame.cap_len	Other	X	G
46	tls.handshake.version	TLS	X	G
47	tls.handshake.ciphersuite	TLS	X	G
48	tls.handshake.ciphersuite_count	TLS	X	G

Continued on next page

Table A1 (continued)

No.	Field	Group	TAE type	Role
49	tls.handshake.extension_count	TLS	X	G
50	tls.handshake.sni_len	TLS	X	G
51	tls.handshake.supported_groups_count	TLS	X	G

Some field names retain historical labels and have implementation-specific meanings. frame.time_delta is the capture-time difference from the previous retained packet in the flow. tcp.options_len counts TCP option items parsed by Scapy. tcp.time_delta is the TSval difference between adjacent parsed TCP Timestamp options, not a capture-time difference. tcp.reassembled.length is cumulative TCP payload length in the current direction up to that packet, not a cross-segment reassembled message length. The six TLS fields are assigned only when an individual packet payload permits ClientHello/ServerHello parsing. Handshake fields requiring cross-segment recovery remain zero.

References

Alshammari R, Zincir-Heywood AN (2011) Can encrypted traffic be identified without port numbers, IP addresses and payload inspection? Computer Networks 55(6):1326–1350. https://doi.org/10.1016/j.comnet.2010.12.002

Anderson B, McGrew D (2017) Machine learning for encrypted malware traffic classification: Accounting for noisy labels and non-stationarity. In: Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp 1723–1732, https://doi.org/10.1145/3097983.3098163

Borman D, Braden B, Jacobson V, et al (2014) TCP extensions for high performance. Tech. Rep. RFC 7323, RFC Editor, https://doi.org/10.17487/RFC7323, URL https://www.rfc-editor.org/info/rfc7323

Cho K, Mitsuya K, Kato A (2000) Traffic data repository at the WIDE project. In: Proceedings of the 2000 USENIX Annual Technical Conference, FREENIX Track. USENIX Association, pp 263–270, URL https://www.usenix.org/publications/library/proceedings/usenix2000/freenix/cho.html

Devlin J, Chang MW, Lee K, et al (2019) BERT: Pre-training of deep bidirectional transformers for language understanding. In: Proceedings of NAACL-HLT 2019, pp 4171–4186, https://doi.org/10.18653/v1/N19-1423

Draper-Gil G, Lashkari AH, Mamun MSI, et al (2016) Characterization of encrypted and VPN traffic using time-related features. In: Proceedings of the 2nd International Conference on Information Systems Security and Privacy, pp 407–414, https://doi.org/10.5220/0005740704070414

Eddy WM (2022) Transmission control protocol (TCP). Tech. Rep. RFC 9293, RFC Editor, https://doi.org/10.17487/RFC9293, URL https://www.rfc-editor.org/info/rfc9293

van Ede T, Bortolameotti R, Continella A, et al (2020) FlowPrint: Semi-supervised mobile-app fingerprinting on encrypted network traffic. In: Proceedings of the Network and Distributed System Security Symposium, https://doi.org/10.14722/ndss.2020.24412

Gorishniy Y, Rubachev I, Khrulkov V, et al (2021) Revisiting deep learning models for tabular data. In: Advances in Neural Information Processing Systems, pp 18932–18943, URL https://proceedings.neurips.cc/paper/2021/hash/9d86d83f925f2149e9edb0ac3b49229c-Abstract.html

He K, Chen X, Xie S, et al (2022) Masked autoencoders are scalable vision learners. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp 16000–16009, https://doi.org/10.1109/CVPR52688.2022.01553

Horowicz E, Shapira T, Shavitt Y (2024) Self-supervised traffic classification: Flow embedding and few-shot solutions. IEEE Transactions on Network and Service Management 21(3):3054–3067. https://doi.org/10.1109/TNSM.2024.3366848

Kornblith S, Shlens J, Le QV (2019) Do better ImageNet models transfer better? In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp 2661–2671, https://doi.org/10.1109/CVPR.2019.00277

Lin X, Xiong G, Gou G, et al (2022) ET-BERT: A contextualized datagram representation with pre-training transformers for encrypted traffic classification. In: Proceedings of the ACM Web Conference 2022, pp 633–642, https://doi.org/10.1145/3485447.3512217

Liu C, He L, Xiong G, et al (2019) FS-Net: A flow sequence network for encrypted traffic classification. In: Proceedings of IEEE INFOCOM 2019, pp 1171–1179, https://doi.org/10.1109/INFOCOM.2019.8737507

Liu Y, Zheng X, Cai C, et al (2026) Adapting large language models for encrypted traffic analysis services: An efficient realization with mixture of LoRA experts. IEEE Transactions on Services Computing 19(2):906–919. https://doi.org/10.1109/TSC.2026.3671484

Lotfollahi M, Jafari Siavoshani M, Shirali Hossein Zade R, et al (2020) Deep packet: A novel approach for encrypted traffic classification using deep learning. Soft Computing 24(3):1999–2012. https://doi.org/10.1007/s00500-019-04030-2

Luxemburk J, Hynek K, Plný R, et al (2026) Universal embedding function for traffic classification via QUIC domain recognition pretraining: A transfer learning success. IEEE Transactions on Network and Service Management 23(1):1647–1663. https://doi.org/10.1109/TNSM.2025.3642984

Rescorla E (2018) The transport layer security (TLS) protocol version 1.3. Tech. Rep. RFC 8446, RFC Editor, https://doi.org/10.17487/RFC8446, URL https://www.rfc-editor.org/info/rfc8446

Rezaei S, Liu X (2019) Deep learning for encrypted traffic classification: An overview. IEEE Communications Magazine 57(5):76–81. https://doi.org/10.1109/MCOM.2019.1800819

Shaw P, Uszkoreit J, Vaswani A (2018) Self-attention with relative position representations. In: Proceedings of NAACL-HLT 2018, Volume 2 (Short Papers), pp 464–468, https://doi.org/10.18653/v1/N18-2074

Shen M, Zhang J, Zhu L, et al (2021) Accurate decentralized application identification via encrypted traffic analysis using graph neural networks. IEEE Transactions on Information Forensics and Security 16:2367–2380. https://doi.org/10.1109/TIFS.2021.3050608

Shen M, Ye K, Liu X, et al (2023) Machine learning-powered encrypted network traffic analysis: A comprehensive survey. IEEE Communications Surveys & Tutorials 25(1):791–824. https://doi.org/10.1109/COMST.2022.3208196

Velan P, Čermák M, Čeleda P, et al (2015) A survey of methods for encrypted traffic classification and analysis. International Journal of Network Management 25(5):355–374. https://doi.org/10.1002/nem.1901

Wang T, Xie X, Wang W, et al (2024) NetMamba: Efficient network traffic classification via pre-training unidirectional mamba. In: Proceedings of the 32nd IEEE International Conference on Network Protocols, pp 1–11, https://doi.org/10.1109/ICNP61940.2024.10858569

Wickramasinghe N, Shaghaghi A, Tsudik G, et al (2025) SoK: Decoding the enigma of encrypted network traffic classifiers. In: 2025 IEEE Symposium on Security and Privacy (SP), pp 1825–1843, https://doi.org/10.1109/SP61157.2025.00165, URL https://doi.ieeecomputersociety.org/10.1109/SP61157.2025.00165

Yang Z, Yang D, Dyer C, et al (2016) Hierarchical attention networks for document classification. In: Proceedings of NAACL-HLT 2016, pp 1480–1489, https://doi.org/10.18653/v1/N16-1174

Yoon J, Zhang Y, Jordon J, et al (2020) VIME: Extending the success of self- and semi-supervised learning to tabular domain. In: Advances in Neural Information Processing Systems, pp 11033–11043, URL https://proceedings.neurips.cc/paper/2020/hash/7d97667a3e056acab9aaf653807b4a03-Abstract.html

Zhao R, Zhan M, Deng X, et al (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):5420–5427. https://doi.org/10.1609/aaai.v37i4.25674

Zhao Y, Dettori G, Boffa M, et al (2025) The sweet danger of sugar: Debunking representation learning for encrypted traffic classification. In: Proceedings of the ACM SIGCOMM 2025 Conference, pp 296–310, https://doi.org/10.1145/3718958.3750498

Zhou G, Guo X, Liu Z, et al (2025) TrafficFormer: An efficient pre-trained model for traffic data. In: Proceedings of the 2025 IEEE Symposium on Security and Privacy, pp 1844–1860, https://doi.org/10.1109/SP61157.2025.00102