NEURINFERNO: Field Inference for Unknown Binary Protocols Without Priors or Binaries

Sachith Abeywickrama*†, Min Wu†, Xiaoli Li‡, and Chau Yuen*
* *School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore*
† *Institute for Infocomm Research, Agency for Science, Technology and Research (A*STAR), Singapore*
‡ *Information Systems Technology and Design, Singapore University of Technology and Design, Singapore*

Abstract*—Reverse engineering unknown binary protocols from raw network traffic is a foundational task in offensive and defensive security. It supports fuzzing, intrusion detection, vulnerability discovery, and the analysis of malware command-and-control channels. Its core step is field inference, the recovery of byte-level field boundaries from raw byte streams. Existing methods resolve the inherent ambiguity of binary messages by importing external knowledge that is often unavailable in practice. Mainly, (1) rule-based analyzers require analyst-supplied endianness and capture-time priors, (2) execution-trace methods require a runnable implementation of the target protocol, and (3) recent learning-based methods perform well on protocols close to their training distribution but generalize poorly to formats they have not seen. We observe that the disambiguating structural signal lies in the co-occurrence of same-format messages rather than in any individual message alone. Hence, we introduce NEURINFERNO, a trace-only neural field inference system that operates without specifications, priors, or executables. NEURINFERNO couples a pre-trained Byte Generative Model (Byte-GM), which supplies per-position predictive entropy, with a Cross-Message Byte Encoder (Cross-MBE), a transformer model that aggregates mean, variance, and maximum-activation statistics across messages at every layer and injects them as explicit features. This design exposes the disambiguating structure directly to the model. We evaluate under a strict protocol-wise leave-one-out across twelve widely used binary protocols, with each test protocol withheld from every training. NEURINFERNO attains an average boundary F1 of 0.81, lifting recall from 0.54 to 0.81 over the strong trace-only baselines while remaining competitive in precision. Its performance is nearly unchanged when the training pool shrinks to eight, demonstrating that the learned signal is protocol-agnostic rather than a form of format memorization. All experimental datasets are derived from protocol dissectors and released with the benchmark, ensuring the evaluation is fully reproducible.

1. Introduction

The traffic of industrial control systems, routing infrastructure, embedded and IoT devices, and proprietary enterprise services is often encoded as binary messages. Protocols define how messages are structured and interpreted. However, their specifications are often unavailable, intentionally hidden, or no longer reflect deployed implementations. This information asymmetry is central to security analysis. An attacker may conceal command-and-control (C2) traffic in a *custom* binary protocol to evade detection, while an analyst must recover the protocol's structure from captured packets to detect the intrusion, build a parser, or fuzz the implementation for vulnerabilities. From the analyst's perspective, the protocol is undocumented, there is no implementation available, and only the raw bytes on the wire can be observed.

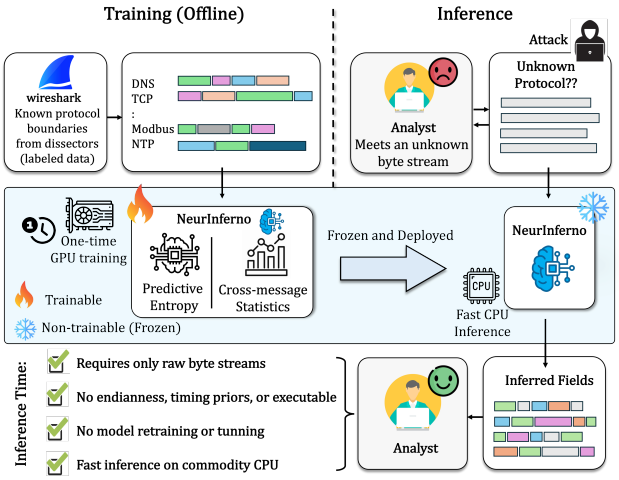

Figure 1. NEURINFERNO separates one-time training from frozen deployment on unknown protocols. During offline training, known protocols provide dissector-derived field labels, which are used to train the model once. At inference time, an **Analyst** receives only raw byte messages from an unknown protocol, with no specification, executable, endianness prior, timing prior, or retraining on the target protocol **(Potential Attack)**. The deployed model infers field boundaries in a single forward pass, enabling low-cost deployment on commodity CPUs.

Recovering message structure under these conditions is protocol reverse engineering ***(PRE)***, and its first step is ***Field Inference***: determining where one protocol field ends and the next begins, while assigning a coarse structural role to each segment when possible. ***Field Inference*** is the load-bearing step of the workflow, as message typing, semantic analysis, parser construction, and state-machine recovery all build on it [1], [2], [3], [4], [5]. An analyst

who cannot recover the field boundaries cannot reliably recover the higher-level protocol structure, which limits the effectiveness of downstream security tasks.

Field inference is challenging because a single binary message hides its own structure. The same four bytes may represent a 32-bit integer, a timestamp, a checksum, a length, or several smaller fields. Unlike textual protocols, binary formats do not carry delimiters or human-readable tokens to distinguish them. The disambiguating evidence appears only across a collection of messages in the same format, where different field roles leave different statistical fingerprints. A fixed field stays near-constant across messages, a length field co-varies with message size, and a payload byte varies freely. These fingerprints are not tied to any one protocol. A fixed header byte and a length field maintain the same signature regardless of whether the protocol comes from a public standard, a proprietary system, or a custom malware channel. The central question is whether these regularities can be learned once, from documented protocols, and transferred to protocols that have never been seen before.

Previous work obtains this signal by making stronger assumptions and falls into three broad categories, each of which falls short in the setting where ***PRE*** is most needed. First, ***Rule-Based Heuristic Analyzers*** match byte regions with detectors for floats, timestamps, length encodings, and other known-idioms [2], [6], [7]. They stay precise by firing only when confident, but the same conservatism limits recall: boundaries outside the detector catalog are systematically missed; stable-byte alignment alone does not recover parser-relevant roles; and many approaches require thresholds or specialized choices that are hard to set for an unknown protocol. The strongest of them also benefit from analyst-supplied priors, such as endianness and approximate capture time, precisely the facts absent when a protocol is unknown [7]. Second, ***Execution-Trace Methods*** observe a running implementation and recover the structure from its memory accesses, taint propagation, or program behavior [1], [4], [8], [9], [10], [11]. This provides the richest available signal because the program itself reveals how bytes are parsed and used. However, it requires an accessible executable, live server, or an analyzable implementation. This assumption is often unmet for malware traffic, undocumented industrial devices, proprietary systems, and legacy deployments. Third, ***Learning-Based Methods*** relax the demands of manual rule design but leave a different gap. Supervised image-based systems show that cross-message structure can be learned by stacking same-format messages and applying neural image or sequence models [12], [13]. However, they rely on substantial training data, the construction of target-traffic images, and knowledge-driven traffic simulation based on protocol documents. Recent self-supervised systems remove the labeled-data requirement but learn separately from the target protocol's own traffic, requiring sufficient messages in that format to capture reliable byte-level relationships [14]. Thus, existing learning-based approaches either import protocol knowledge, require substantial target traffic, or fit the model to the target protocol itself.

None of these settings matches the moment an analyst first encounters a protocol, with only a set of captured binary messages, and they require a protocol specification, expert- or analyst-supplied priors, executable code, or target-protocol training data. Because one or more of these requirements are often unavailable during real-world inference time.

This paper introduces NEURINFERNO, a data-driven, trace-only field inference system that requires no specification, no analyst-supplied priors, no executable, and no traffic from the target protocol at inference time (Figure 1). The proposed system is trained once on documented protocols, in which field boundaries are automatically obtained from dissectors [15], [16]; the trained model artifact is then frozen and deployed. A new and *fully unknown protocol* arrives as nothing more than a set of byte messages, and the frozen model infers its boundaries in a single forward pass with no retraining, fast enough to run on a commodity CPU, while training is a one-time cost on a single GPU.

This design sidesteps the obstacle that supervised field inference seems impossible without labels for the unknown protocol [14]. NEURINFERNO never trains on labels from the target protocol. Instead, it learns only from documented protocols, where dissectors automatically supply boundaries, and transfers the learned disambiguating signal to protocols it has never seen. Two mechanisms expose the cross-message signal. First, a pretrained Byte Generative Model **(Byte-GM)** supplies per-position predictive entropy, a surprise signal that shifts at field boundaries. Second, a Cross-Message Byte Encoder **(Cross-MBE)** jointly processes a batch of same-format messages, aggregating the mean, variance, and maximum of each offset across messages at every layer and injecting them as explicit features. Fixed-versus-variable structure thus becomes a quantity the model reads off directly rather than a texture it must recover from one message at a time.

NEURINFERNO does not claim full semantic recovery. Protocol-specific names such as a transaction identifier or a window-size field require external specifications, so the system targets coarse structural roles, type-like and length-like fields, counts, constants, numeric regions, padding, checksums, payloads, and an explicit `UNKNOWN` category, avoiding unsupported semantics while still providing parser-relevant structure.

The evaluation targets the operational case in which the analyst receives an unknown-byte message from a protocol the model has never seen. We evaluate on real protocols using disjoint training and inference splits, ensuring that each inference protocol is withheld from training, validation, and hyperparameter tuning. For the training protocols, the ground-truth boundaries were derived from protocol dissectors. The held-out protocols reach the model only at inference time, exactly as a deployed protocol would, as raw bytes with no side information. This closes the unsupervised leakage path that cross-protocol evaluations usually overlook. Across twelve binary protocols, NEURINFERNO attains an average boundary precision of 0.81, recall of 0.82, false-positive rate of 0.079, and F1 of 0.81 under leave-one-

protocol-out evaluation. The low false-positive rate shows the model does not simply predict boundaries everywhere, and because the dissector labels for the held-out protocols are retained only for scoring, never for training, true precision, recall, and F1 are reported on every unseen protocol, rather than the coverage proxy that label-free methods fall back on. Under a harsher leave-four-protocols-out setting, performance remains close to leave-one-protocol-out, direct evidence that the model captures the protocol-agnostic regularities of field structure rather than memorizing the specific formats it was trained on.

This paper makes the following contributions.

- We formulate data-driven field inference for unknown binary protocols as gap-level boundary prediction with coarse byte-level role assignment, under a low-assumption setting with no specifications, no analyst-supplied priors, no executable binaries, and no target-protocol training data at inference time.
- We propose NEURINFERNO, a cross-message byte-level architecture that combines **Byte-GM** predictive entropy with **Cross-MBE** layer-wise cross-message statistics. This design makes the fixed-versus-variable structure an explicit model feature and enables frozen inference to run fast enough on a commodity CPU.
- We construct and release a reproducible benchmark of twelve binary protocols, with field boundaries and coarse roles derived programmatically from protocol dissectors. The benchmark supports strict unknown-protocol evaluation through leave-one-protocol-out and leave-four-protocols-out settings.
- We evaluate NEURINFERNO under these strict protocol-wise settings, showing strong boundary recovery on fully unseen protocols at a low false-positive rate. Performance remains stable as the training pool shrinks, isolating protocol-agnostic generalization from format memorization.

2. Related Work

Field inference is the central step of protocol reverse engineering and has been studied for two decades [17], [18], [19], [20]. Approaches are divided first by whether they require a running implementation. Execution-trace methods require an executable, while trace-only methods operate on captured bytes alone. Trace-only methods are further divided by how they obtain the missing structural signal: by hand-designed rules or by learning. We organize prior work along these lines, matching the three categories of Section 1, and then place NEURINFERNO within them.

Rule-Based Trace-Only Methods. The first and largest family works from traffic alone using hand-designed rules, by aligning messages, inspecting per-message structure, or matching bytes against semantic templates. **Netzob** [3], and **NetPlier** [21] align byte sequences across messages with techniques adapted from bioinformatics, treating consistently aligned positions as fields. Alignment depends on strong byte commonality and degrades with diverse message sets; NetPlier, in particular, divides messages into many one-byte fields, finding most boundaries at the cost of a high false-positive rate. **Discoverer** [22] clusters messages by token pattern but assumes the analyst supplies delimiters and yields no semantics for binary fields. Working inside a single message instead of across many, **Nemesys** [23] infers boundaries from bit-level change between adjacent bytes and therefore cannot use the cross-message regularities that separate a fixed byte from a variable one at the same offset, with later refinements sharpening these per-message boundaries through principal component analysis [24] and hybrid tree representations [25]. A second group matches bytes against known templates. **FieldHunter** [2] correlates byte regions with session metadata to identify type, length, and identifier fields, and **Awre** [6] searches for preambles, sync words, and length fields in wireless formats. **BinaryInferno** [7], the strongest recent member, combines detectors for floats, timestamps, and integer-length encodings with an entropy-based boundary detector and a serialization-pattern search, integrated by a maximum-coverage algorithm. It is built to avoid false positives and achieves high precision, but the same conservatism limits recall, since a detector fires only on the encodings it was built for, and its strongest configuration further requires analyst-supplied endianness and capture-date priors, exactly the knowledge absent when a protocol is unknown. Across this family, the recoverable structure is bounded by the rules that an expert anticipated.

Execution-Trace Methods. The second family observes a protocol implementation as it processes a message, using taint analysis to determine which bytes are accessed together by which instructions. **Polyglot** [1], **AutoFormat** [8], and **Tupni** [9] established this approach, and **Dispatcher** [26] and **Prospex** [27] extended it toward command-and-control analysis and specification recovery. Recent advances in systems have substantially improved it. **BinPRE** [4] uses instruction-level semantic similarity with a cluster-and-refine stage, reporting format F1 of 0.86 and uncovering a zero-day during fuzzing. **DynPRE** [10] augments passive analysis by probing a live server and updating its field model from the responses, and **ICEPRE** [11] applies data-driven concolic execution to industrial control protocols, reaching an F1 of 0.76 without a fully executable environment. These methods obtain a far richer signal than any trace-only approach, because the implementation itself reveals boundaries through its memory accesses, but at the cost of a hard requirement: access to a runnable implementation, which is unavailable for malware traffic, undocumented devices, and legacy systems.

Learning-Based Trace-Only Methods. The third family, closest to NEURINFERNO, learns field structure from traffic rather than encoding it by hand. Early convolutional and recurrent models classify or embed packets and optimize for a sub-task, such as protocol identification, rather than producing boundaries [28], [29]. **PREUNN** [30] reverses text-based protocols with a suite of neural networks, and **PREIUD** [31]

pairs a voting-expert segmenter with a BiLSTM-CRF for industrial protocols. **CNNPRE** [32], **ProsegDL** [12], and **DL-ProS^2** [13] render stacked same-format messages as a grayscale image and apply image-segmentation networks, with DL-ProS^2 adding a U-Net, Siamese network, and BiLSTM-CRF and synthesizing training traffic from protocol documents through a large language model. A separate direction applies large language models directly to PRE for segmentation and semantic labeling [33] and for malware C2 analysis through agentic workflows [34]. Most relevant is the recent self-supervised line. **SLMSP** [14] learns byte-relation embeddings without labels, fusing a next-byte order model with a co-occurrence model and a positional-entropy correction, then segments each message from the resulting relation curve, demonstrating that label-free segmentation is feasible.

NEURINFERNO occupies a position none of these methods does, distinguished on three axes. First, its inductive bias is explicit rather than emergent. The image-based line recovers cross-message structure indirectly as texture learned by convolutional filters, whereas NEURINFERNO aggregates mean, variance, and maximum across messages at every encoder layer and injects them as features, making fixed-versus-variable structure an explicit quantity. Second, SLMSP and the unsupervised methods train and evaluate on the same protocol, since their signal is computed from the target protocol's own traffic, so they require traffic from the format under analysis and offer no evidence on a format seen for the first time, whereas NEURINFERNO trains once on documented protocols and transfers to a protocol withheld from every stage. Third, and most consequential for a security tool, prior learning evaluations operate largely in the partially unknown regime, where the test protocol or a close relative appears in training. DL-ProS^2 explicitly separates partially-unknown from completely-unknown protocols and reports much weaker results on the latter. NEURINFERNO is evaluated only under strict protocol-wise leave-one-out, with each test protocol withheld from supervised training and from **Byte-GM** pretraining, and its performance is nearly unchanged as the training pool shrinks, direct evidence that it learns a protocol-agnostic notion of field structure rather than memorizing trained formats.

3. Preliminaries

Problem Formulation

Let $\mathcal{X} = \{x^{(1)}, \ldots, x^{(N)}\}$ denote a batch of N binary messages drawn from the same unknown protocol format. Each message is a byte sequence

$$x^{(n)} = (b_1^{(n)}, b_2^{(n)}, \ldots, b_{L_n}^{(n)}),$$

with $b_i^{(n)} \in \{0, \ldots, 255\}$. Messages are padded to a common length L and a binary validity mask $m_i^{(n)}$ excludes padding positions from all computations.

Primary task: gap-level boundary prediction. For each valid inter-byte gap $(i, i{+}1)$ in message n, the model predicts

$$\hat{y}_i^{(n)} \in \{0, 1\}, \quad i = 1, \ldots, L_n{-}1,$$

where $\hat{y}_i^{(n)} = 1$ indicates a field boundary after byte i.

Auxiliary task: coarse field-role prediction. When byte-level annotations are available, the model also predicts a coarse structural role

$$\hat{r}_i^{(n)} \in \mathcal{C},$$

where $\mathcal{C}$ contains eight protocol-agnostic predicted roles: `LENGTH`, `TYPE_TAG`, `ADDRESS`, `PORT`, `FLAGS`, `CHECKSUM`, `NUMERIC`, and `OPAQUE`. These roles are obtained by merging fine-grained dissector labels with similar cross-protocol byte-level signatures. For example, `QUANTITY`, `TIMESTAMP`, `COUNTER`, `FLOAT`, and `INTEGER` are mapped to `NUMERIC`, while `ASCII`, `PADDING`, and uninterpreted payload-like regions are mapped to `OPAQUE`.

These roles are coarse by design. They do not attempt to recover protocol-specific field names, such as transaction identifier, advertised window, or operation code, which require external protocol context. Instead, the auxiliary task encourages the model to learn parser-relevant structural categories that transfer across protocols.

Neural Building Blocks

We briefly define the standard neural components used throughout the paper. Let D denote the hidden dimension. A byte value $b_i^{(n)}$ is mapped to a vector representation using a learned byte embedding E_b and a learned positional embedding E_p:

$$h_i^{(n,0)} = E_b(b_i^{(n)}) + E_p(i), \qquad E_b, E_p \in \mathbb{R}^D.$$

Here, E_b captures byte identity and E_p encodes the offset of the byte within the message.

A Transformer layer denotes a standard pre-norm self-attention block with a position-wise feed-forward network:

$$u = h + \mathrm{MHA}(\mathrm{LN}(h)),$$

$$\mathrm{TransformerLayer}(h) = u + \mathrm{FFN}(\mathrm{LN}(u)).$$

The multi-head attention module follows the standard scaled dot-product attention:

$$\mathrm{Attn}(Q, K, V) = \mathrm{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right) V,$$

where Q, K, and V are learned projections of the input sequence.

We use MLPs for the prediction heads. A two-layer MLP with activation ϕ is written as

$$\mathrm{MLP}(x) = W_2\phi(W_1 x + a_1) + a_2,$$

where ϕ is GELU in our implementation. For multi-class prediction, the logits are normalized with softmax:

$$\mathrm{softmax}(z)_c = \frac{\exp(z_c)}{\sum_{c'} \exp(z_{c'})}.$$

For binary boundary prediction, a scalar logit s is converted to a probability using the logistic sigmoid:

$$\sigma(s) = \frac{1}{1 + \exp(-s)}.$$

We use $[\,;\,]$ to denote vector concatenation and $\odot$ to denote element-wise multiplication.

4. Methodology

NEURINFERNO consists of four components: (1) a Byte Generative Model **(Byte-GM)** that provides predictive entropy; (2) a Cross-Message Byte Encoder **(Cross-MBE)** that injects offset-wise statistics; (3) a boundary prediction head; and (4) an auxiliary field-role prediction head. Messages are always processed in format-level groups: all messages in a batch share the same unknown format [14], [32], [35]. This batching strategy is essential because the cross-message statistics that distinguish fixed from variable structure are properties of a message collection, not of any individual message. Figure 2 shows the full architecture.

4.1. Byte Generative Model (Byte-GM)

The first source of evidence is predictive uncertainty. Our use of predictive entropy as a boundary cue follows two lines of work. Byte-level Large Language Models learn next-byte distributions over raw bytes [36], and entropy-guided segmentation places boundaries where a time series model's predictability shifts [37], [38], an effect we exploit at the byte level where a field transition changes the statistics of the preceding context. **Byte-GM** is pretrained on raw byte sequences with a next-byte prediction objective (See Figure 2 Stage 1). For each valid position i, **Byte-GM** produces logits $z_i^{(n)} \in \mathbb{R}^{256}$ and the next-byte distribution

$$\pi_i^{(n)} = \mathrm{softmax}(z_i^{(n)}), \qquad \pi_i^{(n)}(v) = P_\theta(v \mid b_{\le i}^{(n)}),$$

producing per-position predictive entropy

$$H_i^{(n)} = -\sum_{v=0}^{255} \pi_i^{(n)}(v) \log \pi_i^{(n)}(v).$$

Low entropy indicates that the next byte is highly predictable, as occurs in fixed-header regions or constant fields. High entropy indicates uncertainty, arising in variable payloads, identifiers, checksums, and data-dependent fields. Changes in entropy across a message correlate with field transitions because the statistical structure of the preceding context shifts at field boundaries. This entropy is supplied as an explicit numerical feature to the boundary head; it is not thresholded as a standalone detector.

The **Byte-GM** is frozen during main-model training (Stage 2). In our strictest evaluation setting, the held-out protocol is also excluded from **Byte-GM** pretraining, closing the one unsupervised leakage path that is commonly overlooked in cross-protocol evaluations.

4.2. Cross-Message Byte Encoder (Cross-MBE)

The primary architectural component is a cross-message transformer encoder (See Figure 2 Stage 2). Each byte is embedded and combined with a learned positional embedding:

$$h_i^{(n,0)} = E_b(b_i^{(n)}) + E_p(i).$$

The encoder contains K transformer layers. In layer k, standard self-attention first processes each message independently:

$$\tilde{h}_{1:L}^{(n,k)} = \mathrm{TransformerLayer}^{(k)}(h_{1:L}^{(n,k-1)}).$$

Then, for every byte offset i, NEURINFERNO computes statistics across the N valid messages:

$$\mu_i^{(k)} = \frac{1}{N_i} \sum_{n=1}^{N} m_i^{(n)} \tilde{h}_i^{(n,k)},$$

$$\nu_i^{(k)} = \frac{1}{N_i} \sum_{n=1}^{N} m_i^{(n)} \left(\tilde{h}_i^{(n,k)} - \mu_i^{(k)}\right)^2,$$

$$\rho_i^{(k)} = \max_{n:m_i^{(n)}=1} \tilde{h}_i^{(n,k)},$$

where $N_i = \sum_n m_i^{(n)}$ is the count of valid messages at offset i. The mean $\mu_i^{(k)}$ captures the common representation; the variance $\nu_i^{(k)}$ captures cross-message variability (directly encoding fixed-versus-variable structure); and the max $\rho_i^{(k)}$ captures extreme activations that may correspond to rare but structurally meaningful byte patterns.

These statistics are concatenated with the per-message representation and projected back via a learned residual:

$$h_i^{(n,k)} = \tilde{h}_i^{(n,k)} + W_k \left[\tilde{h}_i^{(n,k)};\, \mu_i^{(k)};\, \nu_i^{(k)};\, \rho_i^{(k)}\right].$$

Every byte representation is thus conditioned on both its within-message context and on how the same offset behaves across all messages in the batch. The final encoder output is $h_i^{(n)} = h_i^{(n,K)}$, and the final cross-message variance $\nu_i^{(K)}$ is retained for the boundary head.

4.3. Boundary Prediction Head

The boundary head classifies each inter-byte gap. For gap $(i, i{+}1)$ in message n, it constructs a feature vector

$$d_i^{(n)} = h_i^{(n)} - h_{i+1}^{(n)},$$

$$r_i^{(n)} = h_i^{(n)} \odot h_{i+1}^{(n)},$$

$$g_i^{(n)} = \left[h_i^{(n)};\, h_{i+1}^{(n)};\, d_i^{(n)};\, r_i^{(n)};\, H_i^{(n)};\, \nu_i^{(K)};\, \nu_{i+1}^{(K)}\right],$$

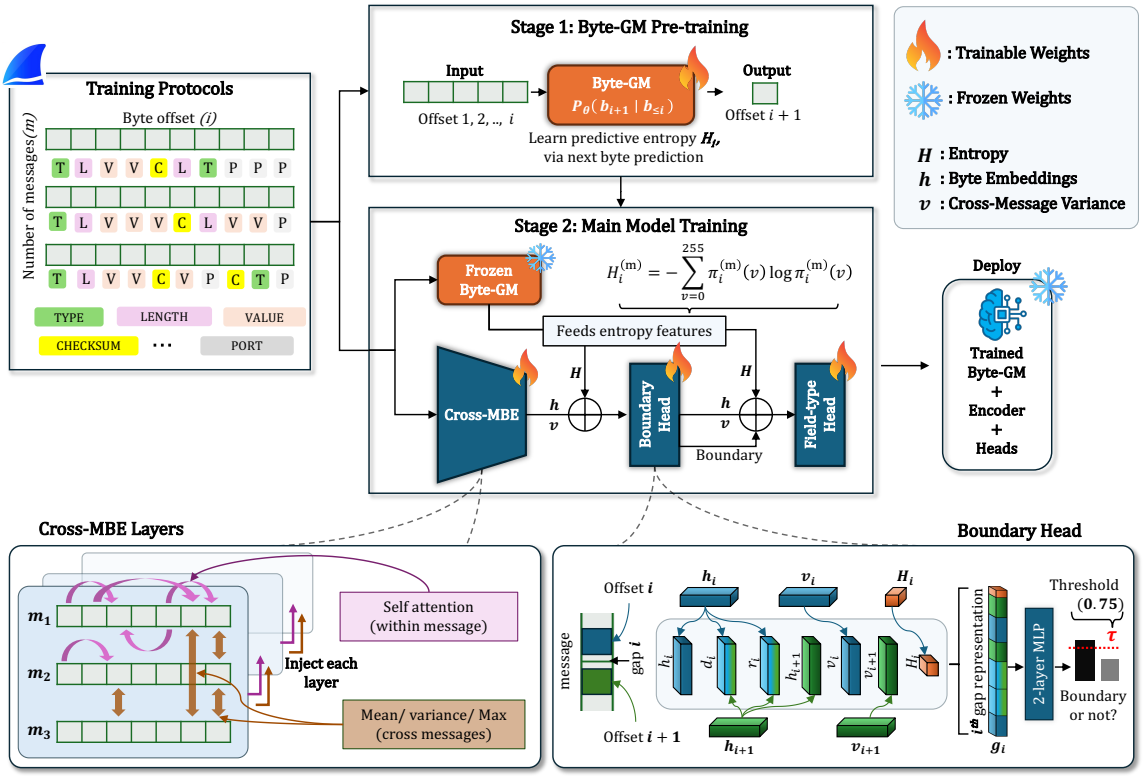

Figure 2. **NEURINFERNO training**. Stage 1, the Byte Generative Model **(Byte-GM: Section 4.1)**, is pretrained on raw unlabeled byte sequences for next-byte prediction. Stage 2, the **Byte-GM** is frozen and supplies per-position predictive entropy H_i while the Cross-Message Byte Encoder **(Cross-MBE: Section 4.2)**, boundary head **(Section 4.3)**, and field-type head **(Section 4.4)** are trained on labeled messages. The encoder inset shows the core mechanism, within-message self-attention combined with mean, variance, and maximum aggregation across messages at the same offset, injected at every layer. The trained **Byte-GM**, **Cross-MBE**, and heads are then frozen for deployment.

where $[\,;\,]$ for vector concatenation, $\odot$ for element-wise multiplication The difference $d_i^{(n)}$ captures representational discontinuity; the element-wise product $r_i^{(n)}$ captures compatibility between adjacent representations. The entropy $H_i^{(n)}$ obtained from Stage 1 pre-trained **Byte-GM** quantifies within-message predictability change, and the two variance vectors $\nu_i^{(K)}$ and $\nu_{i+1}^{(K)}$ indicate whether either side of the gap belongs to a fixed or variable region across the batch. A two-layer MLP produces the boundary logit:

$$s_i^{(n)} = \mathrm{MLP}(g_i^{(n)}), \qquad p_i^{(n)} = \sigma(s_i^{(n)}).$$

and the boundary probability is $p_i^{(n)} = \sigma(s_i^{(n)})$. σ is the Sigmoid Logistic Function.

4.4. Auxiliary Field-Role Head

Once the boundary prediction is learned on the training data, representation vectors can be obtained at the byte and boundary-gap levels. A byte-level field-role head leverages these representations with predicted boundary probabilities and produces a distribution over coarse roles:

$$q_i^{(n)} = f_{\mathrm{role}}(h_i^{(n)}, v_i^{(K)}, H_i^{(n)}, p_i^{(n)}).$$

It enables coarse role assignment during inference. After predicting boundaries, byte-level role probabilities are aggregated within each inferred segment by mean pooling followed by argmax. The coarse role vocabulary is intentionally protocol-agnostic, so the same role (*e.g.*, `LENGTH`) can appear in any protocol regardless of its encoding width or byte values. The implementation includes an optional byte-level field-role head. However, all boundary-detection results reported in this paper are obtained by training only the boundary objective. Therefore, the field-role head is not used to improve the reported boundary results.

4.5. Training Objective

NEURINFERNO is trained with a boundary-detection objective over inter-byte gaps. For each valid gap $(i, i+1)$, let $y_i \in \{0, 1\}$ denote whether the gap is a ground-truth field boundary and let p_i be the predicted boundary probability. The training objective is

$$\mathcal{L} = \mathcal{L}_{\mathrm{bdry}}.$$

Boundary loss. Field boundaries are much sparser than non-boundary gaps. To address this class imbalance, we train the boundary head using focal binary cross-entropy with dynamic positive weighting. For a valid gap, define

$$p_{t,i} = \begin{cases} p_i, & y_i = 1, \\ 1 - p_i, & y_i = 0. \end{cases}$$

The boundary loss is

$$\mathcal{L}_{\text{bdry}} = -\frac{1}{|\mathcal{G}|} \sum_{i \in \mathcal{G}} \alpha_i (1 - p_{t,i})^{\gamma} \log(p_{t,i}),$$

where $\mathcal{G}$ is the set of valid inter-byte gaps and $\gamma = 2$ is the focal parameter. The weight α_i compensates for the imbalance between boundary and non-boundary gaps. In each training batch, we compute the ratio between negative and positive gaps, clamp it to a fixed range, and use it as the positive-class weight:

$$w^+ = \text{clip}\left(\frac{N_-}{\max(N_+, 1)}, 1, 100\right),$$

$$\alpha_i = \begin{cases} w^+, & y_i = 1, \\ 1, & y_i = 0. \end{cases}$$

Here, N_+ and N_- are the number of positive and negative gaps in the batch. This prevents the model from converging to the trivial solution of predicting every gap as non-boundary, while focal reweighting reduces the influence of easy examples.

4.6. Training Procedure

Training proceeds in three stages. First, the **Byte-GM** is pretrained on raw byte sequences with next-byte prediction. This stage does not require boundary or field-role labels and allows the model to learn general byte-level regularities such as fixed prefixes, repeated values, length-related patterns, and high-entropy payload regions.

Second, the main NEURINFERNO model is trained with the **Byte-GM** frozen. In this stage, the language model provides stable entropy features, while the **Cross-MBE** and prediction heads learn to map entropy, representation changes, and cross-message statistics to boundaries and field roles. Batches are constructed as format groups: all messages in a batch come from the same protocol format. This is necessary because cross-message statistics are meaningful only when the messages share a common structure.

Third, we optionally fine-tune the full model end-to-end with a smaller learning rate. This allows the byte generative model representations to adapt to the field-inference objective while preserving the general byte-level knowledge learned during pretraining. In protocol-wise generalization experiments, the held-out inference protocol is excluded from supervised training. Specifically, unlabeled raw bytes from the held-out protocol are also excluded from **Byte-LM** pretraining, preventing leakage through unsupervised byte statistics.

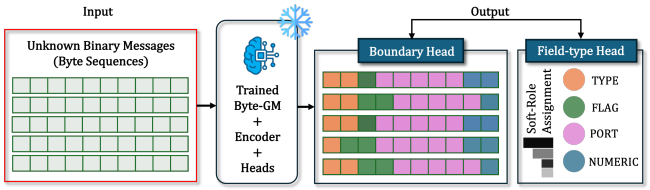

Figure 3. **NEURINFERNO inference**. A batch of messages from an unknown protocol, raw bytes with no specification, priors, or executable, is passed through the frozen **Byte-GM**, **Cross-MBE**, and heads in a single forward pass. The boundary head segments each message, and the field-type head assigns a coarse role to each segment by soft aggregation, here type, constant, payload, and length. No retraining or tuning occurs at inference.

4.7. Inference

At test time, NEURINFERNO receives a batch of messages from an unknown protocol format. The model does not require the protocol name, endianness, capture timestamp, specification, or executable implementation. It first computes byte-level predictive entropy using **Byte-GM**. It then jointly encodes all messages using the **Cross-MBE** and predicts a boundary probability for each valid inter-byte gap. A threshold τ converts probabilities into binary boundary predictions:

$$\hat{y}^{(n)} * i = \mathbb{I}\left[p^{(n)} * i > \tau\right].$$

The predicted boundaries partition each message into contiguous segments. When field-role predictions are used, each segment receives a coarse role by aggregating byte-level role probabilities over the segment:

$$\hat{c} * a : b = \arg\max * c \in \mathcal{C} \frac{1}{b - a + 1} \sum_{i=a}^{b} q_{i,c}.$$

The final output is therefore a parser-like structural description: a sequence of inferred fields, each with byte offsets and, when supported by the model, a coarse structural role. This output is intended to assist downstream reverse-engineering workflows by identifying likely field boundaries and highlighting interpretable regions for analyst inspection.

5. Experiments and Analysis

5.1. Experimental Setup

5.1.1. Performance Evaluation Methodology. Our evaluation targets a single operational question: *"Can our field inference system,* NEURINFERNO*, recover boundaries on a protocol it has never encountered, the case an analyst faces when meeting undocumented malware or proprietary traffic for the first time?"* This is more challenging than it first appears to evaluate, because proving generalization requires withholding a protocol from training and then evaluating it, yet evaluating requires ground truth of field boundaries. We separate these two roles cleanly. Every inference protocol

TABLE 1. DATASET PROTOCOLS AND STRUCTURAL COVERAGE.

Protocol	Use and structural property
ARP	Address resolution; short, mostly fixed-size messages.
BGP	Inter-domain routing; fixed header with variable routing attributes.
DHCP	Host configuration; option-rich, variable-length messages.
DNS	Name resolution; header plus variable question/record sections.
ICMP	Diagnostics/control; compact type/code/checksum structure.
IGMP	Multicast management; short control messages with group addresses.
IP	Network delivery; compact header with length, flags, addresses.
Modbus	Industrial control; short request/response messages with function codes.
NTP	Time synchronization; mostly fixed format with timestamp fields.
OSPF	Intra-domain routing; fixed header with routing-control payloads.
TCP	Reliable transport; ports, sequence numbers, flags, and options.
UDP	Datagram transport; minimal fixed header with variable payload.

we evaluate is withheld from **Byte-GM** self-supervised pre-training, **Cross-MBE** training, and fine-tuning, so it is unseen by the model and serves as an operationally unknown format. Dissector-derived labels for held-out protocols are used only to measure the prediction performance and never enter any training or validation stage.

From the deployed model's standpoint, a withheld protocol is indistinguishable from one that is genuinely undocumented, so performance on held-out protocols is direct evidence for performance on protocols the analyst has never seen. This design choice also distinguishes us from recent self-supervised segmentation methods, such as SLMSP [14]. SLMSP learns from the target protocol's own traffic and is therefore trained and evaluated on the same protocol, even though it doesn't require ground-truth boundaries for self-supervised training. This means it requires traffic from the format under analysis and offers no evidence for a format seen for the first time. A further consequence concerns measurement. Because SLMSP analyzes undocumented protocols for which no ground truth exists, it can report only a coverage proxy on them, the fraction of held-out messages that fit an inferred format, rather than boundary precision and recall. By keeping documented protocols for the held-out role, we report true boundary precision, recall, F1-score, and FPR on every protocol the model has never seen, which provides a stronger, more direct measure of the generalization we claim.

5.1.2. Datasets. We evaluate on twelve protocol-specific datasets: **ARP**, **BGP**, **DHCP**, **DNS**, **ICMP**, **IGMP**, **IP**, **Modbus**, **NTP**, **OSPF**, **TCP**, and **UDP**. Table 1 summarizes their applications and structural properties. For each protocol, we collect up to 2,000 structurally valid messages using public captures following BinaryInferno [7] and protocol-specific packet generation [15]; Modbus yields 1,497 messages after validation. Byte-level role labels and gap-level boundary labels are derived programmatically from packet and dissector layouts, making the benchmark deterministic and reproducible. To include mildly malformed traffic, we apply controlled corruptions to 10% of messages by flipping a small number of bytes or truncating a suffix; corrupted bytes are labeled as UNKNOWN, while remaining valid boundaries are preserved.

The dataset spans link-layer resolution, network-layer control, routing, transport, time synchronization, name resolution, host configuration, and industrial-control communication. It also covers a range of message structures, from short fixed-format protocols such as **ARP**, **IGMP**, **NTP**, and **UDP** to variable or option-rich protocols such as **DHCP**, **DNS**, **TCP**, **OSPF**, and **BGP**. This diversity ensures that leave-out evaluations test protocol-agnostic field-structure learning rather than memorization of a narrow protocol family.

In addition to protocol-specific messages, we use grammar-sampled synthetic messages to expose the model to protocol-agnostic structural motifs. The generator samples abstract binary formats composed of fixed-width fields, length-value fields, type-length-value fields, quantity-controlled repetitions, repeated variable-length patterns, and suffix payloads. These grammar exposures to the model as an initial warm-up process were inspired by the effectiveness of different ensembles of BinaryInferno [7]. Each sampled format is instantiated into multiple messages with varying byte values and field personalities, such as constants, counters, floating-point values, ASCII strings, integers, and addresses.

The grammar-sampled tier is used only for training and ablation studies, never as a held-out evaluation protocol. We therefore report the main results only on the twelve protocol-specific datasets above. To measure whether grammar-sampled data improves transfer or merely induces artificial-pattern memorization, Section 5.3 includes a data-tier ablation comparing protocol-specific data only, grammar-sampled data only, and the combined training setting.

5.1.3. Baselines. We compare against four trace-only baselines that, like NEURINFERNO, operate without any target-protocol data: **BinaryInferno** [7] (BI) in its prior-free configuration (endianness and date priors omitted); **Nemesys** [23]; **NetPlier** [21]; and **Netzob** [3]. All baselines receive the same message sets as NEURINFERNO.

We additionally compare against **SLMSP** [14], the closest learning-based competitor, which we reimplement from its description as no public code is available. Unlike the four baselines above, SLMSP is self-supervised on the target protocol's own traffic. We therefore report it separately, in its native per-protocol setting and in an out-of-design transfer ablation, in Section 5.2 and Table 5. We run it with the recommended β=0.85 and a fixed training budget uniformly across all protocols, with no per-protocol tuning, and score its output with the same boundary scorer used for every other method.

Execution-trace methods such as BinPRE [4], DynPRE [10], ICEPRE [11], and Polyglot [1] are excluded because they require an executable implementation and thus cannot operate in our evaluation setting.

5.1.4. Implementation Settings. We follow two specific setups to ensure the generalization to unseen protocols is fairly evaluated and strictly test our claims.

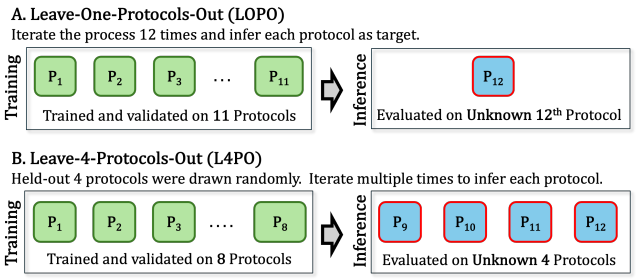

Figure 4. **Evaluation**: we evaluate generalization under Leave-One-Protocol-Out (LOPO, train on 11, test on 1, repeated 12 times) and the stricter Leave-4-Protocols-Out (L4PO, train on 8, test on 4). Inference (held-out) protocols are excluded from every training stage.

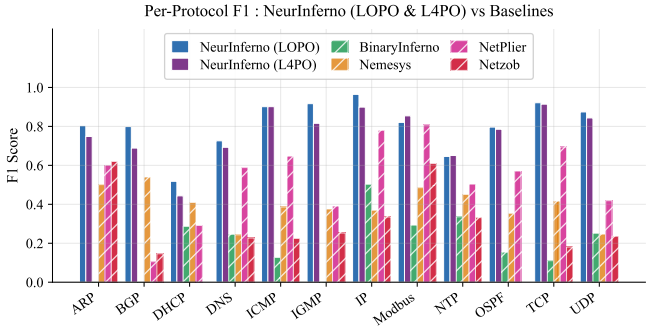

Figure 5. Per-protocol boundary F1 across all twelve protocols. NeurInferno under LOPO (blue) and L4PO (purple) is compared against the four trace-only baselines.

Setup A. Leave-One-Protocol-Out (LOPO) (Figure 4 A) For each of the twelve protocols, one protocol is held out from all supervised training and from **Byte-GM** self-supervised pretraining. The model is trained and validated only on the remaining eleven protocols and evaluated on the held-out protocol. This process is repeated once for each protocol, yielding twelve leave-one-protocol-out folds. For each fold, we train the model with three independent random seeds and report the mean and standard deviation to account for training stochasticity.

Setup B. Leave-Four-Protocols-Out (L4PO). (Figure 4 B) To test a more severe generalization than in the LOPO scenario, we randomly select eight protocols for training and validation and hold out the remaining four for evaluation. The held-out protocols are excluded from all training phases, as in the LOPO scenario. We repeat this procedure with 10 independently drawn protocol-split combinations. Because each combination holds out four protocols, this gives multiple held-out evaluations for each protocol across the ten combinations. We report the mean and standard deviation for each protocol as held out across all L4PO runs to account for both split variability and training stochasticity.

For all training runs, we use a single NVIDIA RTX 6000 Ada GPU. Computational cost measurements also use the CPU and MPS backends described in Section 5.7. Unless otherwise stated, boundary predictions are obtained with a fixed decision threshold of $\tau = 0.75$, chosen a priori and used uniformly across all held-out protocols. All model and training hyperparameters are set once and kept unchanged across inference protocols to ensure a fair protocol-wise evaluation. The full architecture and training hyperparameters are reported in Tables 7 and 8.

5.1.5. Evaluation Metrics. We evaluate boundary detection at the byte-gap level. For a message of length L, there are $L - 1$ candidate boundary offsets, one between each pair of adjacent bytes. A predicted boundary is a true positive (TP) if it exactly matches a ground-truth field boundary offset, and a false positive (FP) otherwise. A ground-truth boundary is a false negative (FN) if no prediction is made at that offset. All remaining non-boundary offsets that are correctly left unpredicted are true negatives (TN). We report boundary-level precision (P), recall (R), F1 score (F1), and false-positive rate (FPR):

$$\mathrm{P} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}}, \quad \mathrm{R} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}, \tag{1}$$

$$\mathrm{F1} = \frac{2 \cdot \mathrm{P} \cdot \mathrm{R}}{\mathrm{P} + \mathrm{R}}, \quad \mathrm{FPR} = \frac{\mathrm{FP}}{\mathrm{FP} + \mathrm{TN}}. \tag{2}$$

Precision measures how many predicted boundaries are correct, recall measures how many true boundaries are recovered, and FPR measures how often the model incorrectly inserts a boundary at a non-boundary byte offset.

5.2. Main Results

Table 2 shows aggregated precision, recall, F1-score and FPR for all methods under both evaluation settings.

5.2.1. LOPO results. NEURINFERNO achieves an average boundary F1 of 0.807 across the 12 unseen protocols, compared with 0.192 for BinaryInferno, 0.398 for Nemesys, 0.533 for NetPlier, and 0.316 for Netzob. The most striking contrast is in recall: NEURINFERNO attains 0.815, compared to BinaryInferno's 0.115, roughly a 7× improvement that directly reflects the recall cost of conservative catalog-based detection. In precision, NEURINFERNO reaches 0.808, comparable to BinaryInferno's 0.736 while recovering far more boundaries, and well above NetPlier (0.565) and Nemesys (0.505). It thus combines the high precision of catalog-based detection with a recall that none of the trace-only baselines approach. See Figure 5 for per-protocol F1-score comparison with baselines.

5.2.2. L4PO results. With only eight training protocols, NEURINFERNO maintains 0.784 average F1 (vs. 0.807 in LOPO), a drop of only 0.023. All baselines are essentially unaffected by the change in the training set, since they use no training data. The per-protocol breakdown for L4PO performance with baselines is shown in Figure 5.

TABLE 2. PER-PROTOCOL BOUNDARY DETECTION RESULTS. NEURINFERNO IS REPORTED UNDER LOPO AND L4PO. BEST PERFORMANCE UNDER EACH METRIC IS IN **BOLD** AND SECOND BEST IS UNDERLINED.

	NeurInferno LOPO				NeurInferno L4PO				BinaryInferno				Netzob†				NetPlier				Nemesys			
Protocol	P	R	FPR	F1	P	R	FPR	F1	P	R	FPR	F1	P	R	FPR	F1	P	R	FPR	F1	P	R	FPR	F1
ARP	0.837	**0.774**	0.064	**0.804**	**0.903**	0.646	0.033	0.748	0.000	0.000	**0.000**	0.000	0.800	0.507	0.054	0.620	0.504	0.747	0.314	0.601	0.519	0.490	0.204	0.501
BGP	0.725	**0.890**	0.081	**0.799**	0.696	0.688	0.073	0.689	0.000	0.000	**0.000**	0.000	0.100	0.350	0.507	0.146	0.094	0.120	0.357	0.105	**0.799**	0.412	0.035	0.540
DHCP	0.506	0.530	0.056	**0.518**	0.350	**0.624**	0.136	0.443	**0.833**	0.173	**0.004**	0.286	–	–	–	–	0.252	0.346	0.114	0.291	0.446	0.380	0.052	0.409
DNS	0.692	**0.764**	0.147	**0.726**	0.680	0.715	0.158	0.693	**1.000**	0.140	**0.000**	0.246	0.600	0.140	0.047	0.227	0.582	0.616	0.209	0.589	0.266	0.230	0.297	0.243
ICMP	0.975	0.838	0.019	**0.901**	0.975	**0.839**	0.019	**0.901**	**1.000**	0.068	**0.000**	0.127	0.667	0.136	0.070	0.225	0.795	0.563	0.137	0.647	0.539	0.303	0.261	0.386
IGMP	**1.000**	**0.846**	**0.000**	**0.917**	0.903	0.748	0.072	0.816	0.000	0.000	**0.000**	0.000	0.667	0.156	0.074	0.253	0.565	0.300	0.212	0.391	0.513	0.298	0.269	0.376
IP	0.995	**0.936**	0.005	**0.964**	0.969	0.843	0.026	0.899	**1.000**	0.336	**0.000**	0.503	0.667	0.224	0.105	0.335	0.785	0.776	0.194	0.780	0.533	0.286	0.232	0.369
Modbus	0.821	0.820	0.214	0.821	0.840	**0.869**	0.199	**0.854**	**1.000**	0.172	**0.000**	0.293	0.750	0.516	0.208	0.609	**1.000**	0.679	**0.000**	0.808	0.769	0.363	0.140	0.485
NTP	0.605	**0.693**	0.122	0.646	0.630	0.672	0.107	**0.650**	**1.000**	0.202	**0.000**	0.336	0.303	0.366	0.233	0.332	0.432	0.604	0.218	0.504	0.481	0.428	0.121	0.451
OSPF	0.896	0.717	0.051	**0.796**	0.847	**0.733**	0.082	0.785	**1.000**	0.084	**0.000**	0.154	–	–	–	–	0.667	0.499	0.154	0.571	0.454	0.290	0.215	0.354
TCP	0.854	**1.000**	0.112	**0.921**	0.855	0.982	0.110	0.914	**1.000**	0.059	**0.000**	0.111	0.400	0.117	0.123	0.181	0.690	0.708	0.215	0.695	0.496	0.363	0.249	0.417
UDP	0.797	**0.967**	0.077	**0.874**	0.823	0.869	0.060	0.843	**1.000**	0.144	**0.000**	0.251	0.667	0.144	0.029	0.236	0.411	0.497	0.291	0.419	0.247	0.267	0.307	0.247
Avg	0.808	**0.815**	0.079	**0.807**	**0.811**	0.774	0.079	0.784	0.736	0.115	**0.000**	0.192	0.562	0.266	0.145	0.316	0.565	0.538	0.201	0.533	0.505	0.342	0.198	0.398

† Netzob macro-averaged over 10/12 protocols; DHCP and OSPF failed.

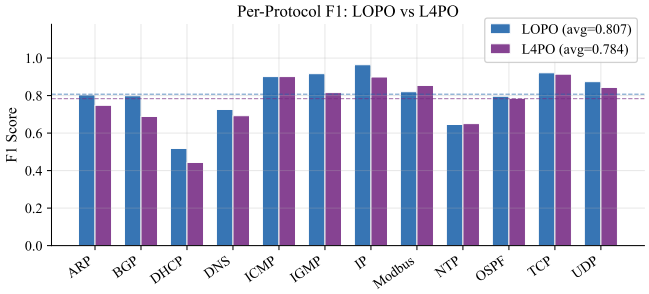

Figure 6. Per-protocol boundary F1 under leave-one-protocol-out (LOPO) and leave-four-protocols-out (L4PO) evaluation. Each bar reports the mean F1 for a held-out protocol. Dashed lines show the macro-average F1 across all protocols for each setting.

5.2.3. LOPO-L4PO comparison. Figure 6 further highlights per-protocol F1 under LOPO and L4PO. Across most protocols, the reduction in F1 is modest despite removing four protocols from the training pool. The macro-average F1 decreases from 0.807 under LOPO to 0.784 under L4PO, while the ranking of protocol difficulty remains largely unchanged. IP, TCP, ICMP, UDP, Modbus, and OSPF maintain strong performance in both settings, whereas DHCP remains the most challenging protocol because of its highly variable option structure. The relatively small gap between LOPO and L4PO provides evidence that the model captures protocol-agnostic structural regularities rather than memorizing the specific formats observed during training.

5.2.4. Comparison to learning-based segmentation. The closest learning-based baseline, SLMSP, operates in a different and strictly less challenging regime than the methods presented in Table 2. SLMSP trains a self-supervised model on the target protocol's own traffic and segments that same traffic. Table 3 compares SLMSP to NEURINFERNO and the trace-only baselines at a macro level, with detailed per-protocol results provided in Table 5. Despite access to the target's own traffic, *however, without any labeled training messages*, SLMSP consistently underperforms relative to NEURINFERNO across all protocols, averaging 0.459 on full messages and 0.350 on the 16-byte truncated view, compared to 0.807 achieved by NEURINFERNO without exposure to the target. When evaluated in NEURINFERNO's regime, trained on eleven protocols and tested on the held-out protocol (SLMSP-transfer), SLMSP's average performance declines further to 0.353, due to its protocol-specific byte-and-offset embeddings that do not generalize. This comparison demonstrates that, even when provided with the target's traffic, SLMSP underperforms and fails to generalize, thereby highlighting the ability of NEURINFERNO to generalize to previously unseen formats, a capability not afforded by target-trained segmentation methods.

TABLE 3. AVERAGE BOUNDARY F1 ACROSS THE TWELVE PROTOCOLS, SITUATING **SLMSP** AGAINST NEURINFERNO AND THE TRACE-ONLY BASELINES. PER-PROTOCOL RESULTS IN APPENDIX TABLE 5.

Method	**Target traffic**	**Avg F1**
NEURINFERNO (LOPO)	none	**0.807**
BinaryInferno	none	0.192
Nemesys	none	0.398
NetPlier	none	0.533
Netzob	none	0.316
SLMSP-full	target	0.459
SLMSP-native	target	0.350
SLMSP-transfer	none	0.353

5.3. Ablation Study

Figure 7 isolates the contribution of each architectural and data component. The full model achieves the highest average F1 in both settings, reaching 0.807 under LOPO and 0.784 under L4PO. Removing cross-message statistics reduces LOPO F1 to 0.741 and L4PO F1 to 0.742, confirming that comparing same-offset behavior across messages is a primary source of boundary evidence. Removing **Byte-GM** entropy produces a similar degradation, reducing F1 to 0.742 under LOPO and 0.733 under L4PO. This shows that

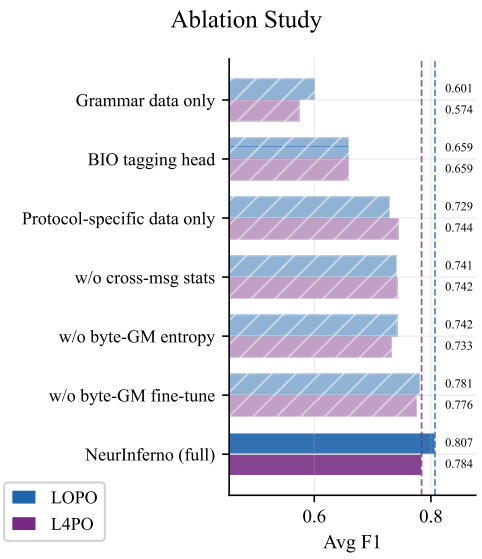

Figure 7. Ablation study under LOPO and L4PO, reporting average boundary F1.

predictive uncertainty contributes complementary information beyond the encoder's representation discontinuity alone.

The data-tier ablations further show that both training sources matter. Training only on protocol-specific data achieves 0.729 LOPO F1 and 0.744 L4PO F1, while grammar-sampled data alone performs substantially worse, at 0.601 and 0.574, respectively. Thus, the synthetic grammar tier is not sufficient on its own, but it improves performance when combined with protocol-specific data by exposing the model to additional protocol-agnostic structural motifs. Finally, removing **Byte-GM *fine-tuning*** causes only a small drop($\approx$ 1%), suggesting that most of the gain comes from the entropy signal itself and from cross-message encoding rather than from extensive adaptation of the byte generative model. Replacing the gap-level boundary head with BIO tagging performs poorly, indicating that direct gap prediction is better aligned with the evaluation target.

5.4. Sensitivity to Cross-Message Context

A practical question for trace-only field inference is how many messages of the same format are needed before cross-message statistics become useful. Figure 8 varies the inference context size N, while keeping the trained model fixed. Performance improves as N increases, confirming that additional messages provide useful evidence for distinguishing fixed fields, variable fields, and payload-like regions. However, the gains saturate around $N = 64 - 100$, and increasing the context to $N = 500$ provides only marginal or no benefit. This justifies the default operating point of $N = 100$ used in the main experiments.

The model also remains effective when only a small number of messages are available. Even at $N = 32$, both LOPO and L4PO retain most of their final F1, indicating that NEURINFERNO does not require large traffic collections to operate. Instead, it exploits whatever cross-message evidence is available, while benefiting from larger batches when available.

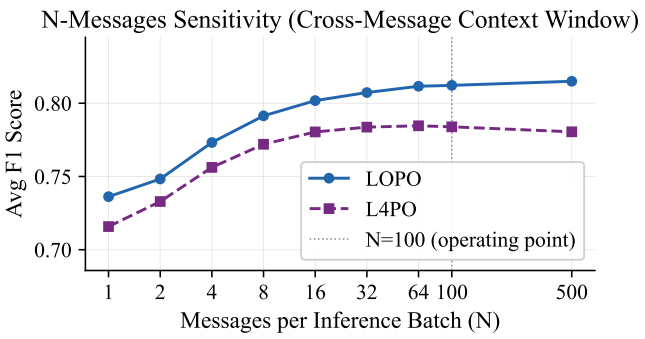

Figure 8. Effect of the number of messages per inference batch N on average boundary F1, with the training batch size ($N = 100$) marked.

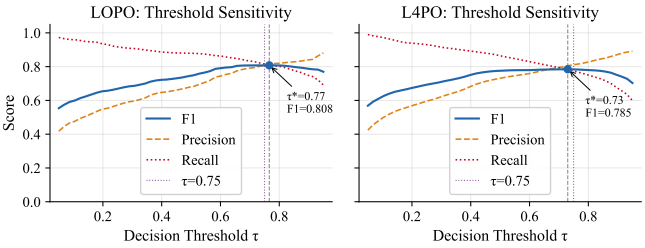

Figure 9. Boundary metrics as a function of decision threshold τ. The default threshold $\tau = 0.75$ was fixed.

5.5. Threshold Sensitivity

The boundary head produces a probability for each candidate byte-gap, which is converted into a boundary prediction using a decision threshold τ. Figure 9 evaluates the effect of varying τ under both LOPO and L4PO. As expected, increasing the threshold makes the model more conservative: precision increases steadily while recall decreases. The resulting F1 curve remains relatively flat across a broad range of thresholds, indicating that performance is not highly sensitive to the exact operating point. The optimal threshold is $\tau = 0.77$ for LOPO and $\tau = 0.73$ for L4PO, while the default value $\tau = 0.75$ lies close to the optimum in both settings. This stability suggests that the boundary probabilities are reasonably calibrated and that NEURINFERNO does not rely on careful threshold tuning to achieve strong performance.

5.6. Auxiliary Field-Role Head Analysis

Field-role prediction is the auxiliary task of NEURINFERNO. To characterize role inference, we train the role head separately on the frozen backbone learned representations and evaluate it under LOPO against coarse labels derived from dissector field names. The head reaches $\approx 55\%$ field-role accuracy on protocols with balanced role diversity

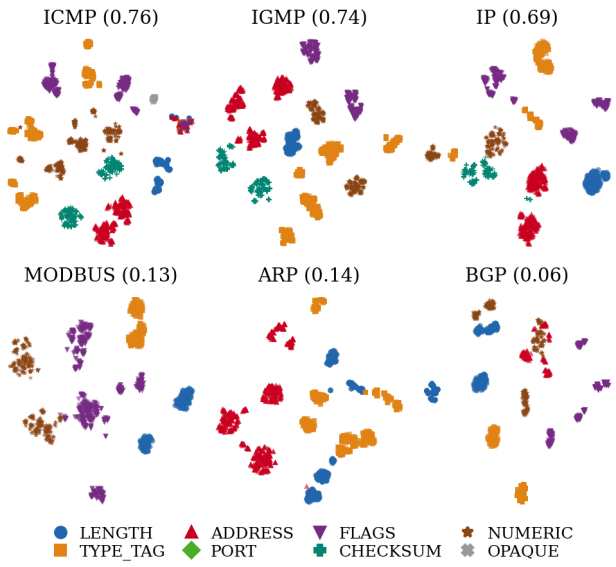

Figure 10. t-SNE of frozen backbone byte representations, colored by ground-truth role, for six held-out protocols under LOPO with per-protocol field-role accuracy in parentheses.

(DNS, ICMP, IGMP, IP) but collapses on protocols dominated by a single role (ARP, Modbus, BGP), defaulting to its training-majority class.

This collapse is a property of the head, not the backbone. Figure 10 projects the frozen representations with t-SNE and observes that ARP, Modbus, and BGP form clean, well-separated role clusters even though the head misclassifies most of their bytes. A balanced linear probe on the same representations confirms this, recovering 4.8-5.6× higher accuracy than the head on these three protocols, including `TYPE_TAG` on Modbus (F1 0.79 vs. near-zero) and `ADDRESS` and `LENGTH` on ARP. The structure is present; however, the deployed head simply fails to read it when the test protocol's role prior differs from training.

Three caveats bound this result. The role labels are heuristic, assigned from dissector field names rather than verified semantics, and are least reliable for protocols like DHCP, whose dominant role is option payload, so we draw no conclusions about DHCP role accuracy. The probe is a diagnostic of representational content, not a deployment method, since its class balancing uses the test protocol's distribution, which is unavailable under LOPO. Finally, BGP's constant-byte `FLAGS` marker, sixteen identical bytes with no within-message variation, is a genuine representational limit that neither the head nor the probe recovers. The takeaway is that the boundary-trained backbone encodes coarse role structure that a lightweight head recovers under matched priors and fails gracefully by defaulting rather than scattering under shifted ones.

5.7. Computational Cost

Once trained, NEURINFERNO is deployed frozen, so the cost that matters in practice is inference: a newly encountered protocol is analyzed with a single forward pass and no target-protocol training. Table 4 reports inference cost for the full **1.44M**-parameter model across four backends, measured over 6,000 held-out messages. Inference is fast on every platform, ranging from **1.4,ms** per message on an RTX 6000 Ada GPU to **6.6,ms** per message on a commodity Xeon CPU. Thus, the system remains practical even without an accelerator.

TABLE 4. INFERENCE PERFORMANCE ACROSS HARDWARE PLATFORMS. TIMING IS MEASURED OVER 6,000 MESSAGES USING WARM-CACHE, STEADY-STATE EXECUTION. MODEL SIZE: 1.44M PARAMETERS.

Backend	Hardware	Time	ms/msg	msg/s
CPU	Xeon w5-3423	39.6 s	6.6	151
CPU	M4 Pro CPU	10.7 s	1.8	560
MPS	M4 Pro GPU	12.8 s	2.1	467
GPU	RTX 6000 Ada	8.6 s	1.4	694

Setup. CPU timing on the Xeon W5-3423 used PyTorch intra-op parallelism across 24 cores. Mac measurements used PyTorch MPS on an Apple M4 Pro with 12-core CPU, 16-core GPU, and 24 GiB unified memory. Linux measurements used Python 3.11.15, PyTorch 2.5.1, CUDA 12.4, cuDNN 9.1.0, and Debian kernel 6.1.

Training is offline and amortized. The complete leakage-free experimental suite, including fold-specific **Byte-GM** pretraining, main-model training, and fine-tuning for all LOPO and L4PO runs, required approximately 4 GPU-hours on a single NVIDIA RTX 6000 Ada (10 minutes per single held-out protocol model). This is an upper bound on the training cost used for evaluation, because the deployment setting trains a checkpoint once and then reuses it for all future protocols. In contrast, target-specific self-supervised methods such as SLMSP [14] learn from each new protocol's own traffic and therefore incur a fresh per-protocol training cost, plus a sequence-merging pass, for every protocol analyzed. NEURINFERNO pays its training costs offline and amortizes them over repeated low-cost inferences.

5.8. Qualitative Analysis

Figure 11 compares boundary predictions produced by NEURINFERNO and four trace-only baselines across representative messages from held-out protocols. These visualizations also follow our strict leave-out method, similar to numerical evaluation.

Several consistent trends emerge. NEURINFERNO recovers most interior field boundaries across both fixed-format protocols (e.g., ARP, ICMP, IGMP, NTP) and highly variable protocols (e.g., DHCP, DNS, TCP, and BGP). In contrast, BinaryInferno is highly conservative and frequently misses valid boundaries, particularly inside long structured regions. This behavior is consistent with the quantitative results, where BinaryInferno achieves low false-positive rates at the expense of substantially lower recall. Alignment-based methods such as Nemesys, NetPlier, and Netzob often over-segment variable regions, producing numerous false-positive boundaries. This effect is particularly visible in

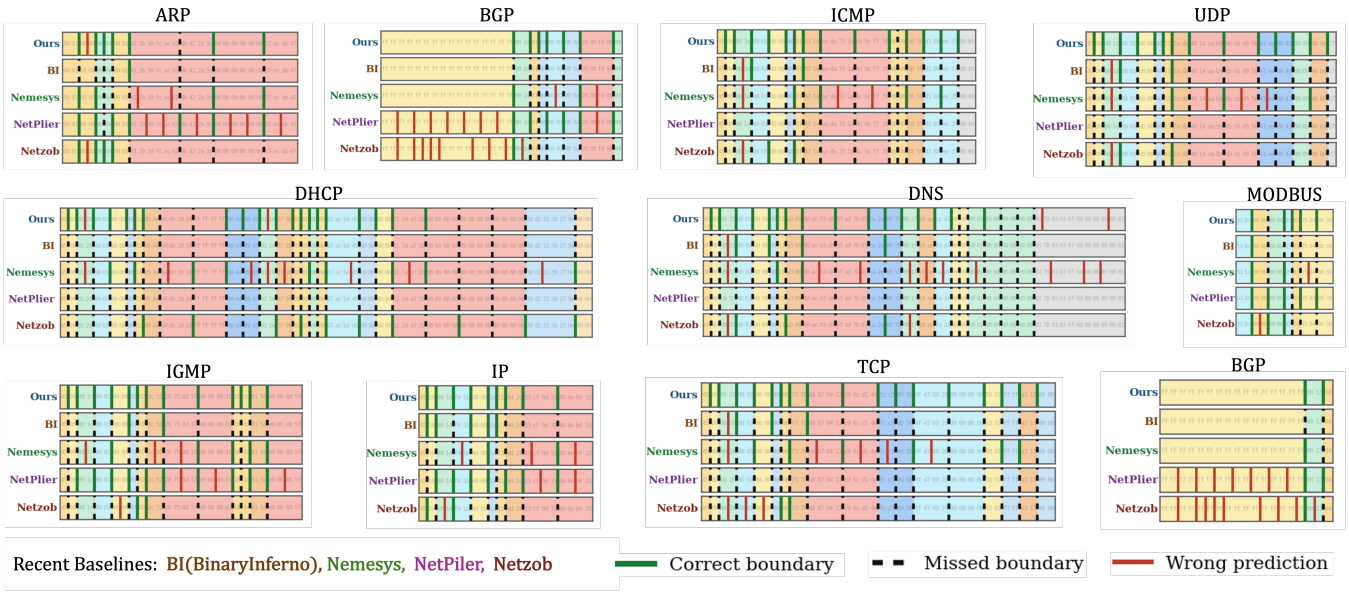

Figure 11. Qualitative comparison of field-boundary predictions across held-out binary protocols. Each panel shows one message instance, with rows corresponding to NEURINFERNO and recent baselines. Green lines mark correctly recovered boundaries, black dashed lines mark missed ground-truth boundaries, and red lines mark false-positive predictions.

DNS, DHCP, TCP, and BGP, where variable-length fields and payload regions create complex cross-message patterns. While these methods sometimes recover major structural transitions, they frequently insert boundaries within a single semantic field. The advantage of NEURINFERNO is not confined to a single protocol family. Improvements are visible across link-layer (ARP), network-layer (IP, ICMP, IGMP), transport-layer (TCP, UDP), routing (BGP, OSPF), infrastructure (DNS, DHCP, NTP), and industrial-control (Modbus) protocols. This qualitative consistency supports the paper's central claim: the model learns protocol-agnostic structural regularities rather than protocol-specific parsing rules.

The visual evidence aligns with the quantitative evaluation. NEURINFERNO achieves its higher F1 score by simultaneously recovering more true boundaries than BinaryInferno while avoiding the systematic over-segmentation exhibited by alignment-based baselines.

5.9. Limitations

In this study, we identify the following potential limitations, which will be investigated in future work.

Byte granularity. Protocols that use bit-packed headers, including Modbus exception codes, IPv4 flags, and fragment offset, inherently include unrecoverable boundaries by design. NEURINFERNO operates at byte granularity and is unable to recover boundaries within bytes, such as sub-byte fields.

Fixed-size messages. For protocols in which all messages are of uniform length, such as pure fixed-size ARP, cross-message variance in representations does not provide additional information beyond the context of individual messages. In these cases, NEURINFERNO does not experience significant performance degradation and functions similarly to a per-message transformer, although the benefits of cross-message analysis are reduced.

Encrypted traffic. TLS (Transport Layer Security) and other encryption schemes randomize byte distributions, thereby eliminating statistical structure. Since NEURINFERNO requires access to plaintext bytes, our approach is not applicable to encrypted payloads.

Semantic precision. The coarse role vocabulary ($|\mathcal{C}| = 8$) offers useful structural labels but is insufficient for recovering protocol-specific field names or subtype semantics without additional context. Users requiring more detailed semantic information must supplement NEURINFERNO's output with further analysis.

Benchmark scope. Our 12-protocol benchmark covers a representative range but excludes highly domain-specific formats, such as custom ICS protocols, firmware update protocols, and encrypted variants. Generalization to these cases and scaling of experiments will be addressed in future research. Additionally, as a foundation model, it remains to be seen how our solution performs when trained on a large, known corpus and subsequently fine-tuned for various downstream tasks that are currently underexplored in the PRE literature. We intend to investigate these aspects in future work.

6. Conclusion

We have introduced NEURINFERNO, a trace-only neural field inference system for analyzing unknown binary protocols. By integrating Byte Generative Model entropy with

explicit cross-message representation statistics injected at each encoder layer, NEURINFERNO identifies field boundaries based on protocol-agnostic fingerprints present across batches of same-format messages. This approach does not require protocol specifications, endianness or timestamp priors, or executable implementations; it operates solely on raw byte streams.

When evaluated using strict protocol-wise leave-one-out validation across ten widely used binary protocols, NEURINFERNO achieves an average boundary F1 score of 0.81. It increases recall from 0.54 (NetPlier) to 0.82 while maintaining competitive precision. The model's near-identical performance when the training pool is reduced from eleven protocols to eight demonstrates its ability to generalize a protocol-agnostic concept of field structure, rather than memorizing specific formats. This finding directly addresses the security-relevant question of whether a field inference system can operate on previously unseen protocols.

A fully reproducible benchmark suit with dissector-derived ground truth labels has been released to facilitate future research. We anticipate that NEURINFERNO will provide a robust foundation for data-driven, trace-only protocol reverse engineering in security contexts where neither executable implementations nor expert analysts are available.

References

[1] J. Caballero, H. Yin, Z. Liang, and D. Song, "Polyglot: Automatic extraction of protocol message format using dynamic binary analysis," in *Proceedings of the 14th ACM conference on Computer and communications security*, 2007, pp. 317–329.

[2] I. Bermudez, A. Tongaonkar, M. Iliofotou, M. Mellia, and M. M. Munafo, "Automatic protocol field inference for deeper protocol understanding," in *2015 IFIP Networking Conference (IFIP Networking)*. IEEE, 2015, pp. 1–9.

[3] G. Bossert, F. Guihéry, and G. Hiet, "Towards automated protocol reverse engineering using semantic information," in *Proceedings of the 9th ACM symposium on Information, computer and communications security*, 2014, pp. 51–62.

[4] J. Jiang, X. Zhang, C. Wan, H. Chen, H. Sun, and T. Su, "Binpre: Enhancing field inference in binary analysis based protocol reverse engineering," in *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, 2024, pp. 3689–3703.

[5] Z. Luo, J. Yu, F. Zuo, J. Liu, Y. Jiang, T. Chen, A. Roychoudhury, and J. Sun, "Bleem: Packet sequence oriented fuzzing for protocol implementations," in *32nd USENIX Security Symposium (USENIX Security 23)*, 2023, pp. 4481–4498.

[6] J. Pohl and A. Noack, "Automatic wireless protocol reverse engineering," in *13th USENIX Workshop on Offensive Technologies (WOOT 19)*, 2019.

[7] J. Chandler, A. Wick, and K. Fisher, "Binaryinferno: A semantic-driven approach to field inference for binary message formats." in *NDSS*, 2023.

[8] Z. Lin, X. Jiang, D. Xu, and X. Zhang, "Automatic protocol format reverse engineering through context-aware monitored execution." in *NDSS*, vol. 8, 2008, pp. 1–15.

[9] W. Cui, M. Peinado, K. Chen, H. J. Wang, and L. Irun-Briz, "Tupni: Automatic reverse engineering of input formats," in *Proceedings of the 15th ACM conference on Computer and communications security*, 2008, pp. 391–402.

[10] Z. Luo, K. Liang, Y. Zhao, F. Wu, J. Yu, H. Shi, and Y. Jiang, "Dynpre: Protocol reverse engineering via dynamic inference." in *NDSS*, 2024.

[11] Y. Qu, D. Fang, Z. Wang, J. Cheng, S. Si, Y. Chen, and L. Sun, "Icepre: Ics protocol reverse engineering via data-driven concolic execution," *Proceedings of the ACM on Software Engineering*, vol. 2, no. ISSTA, pp. 2384–2406, 2025.

[12] S. Zhao, J. Wang, S. Yang, Y. Zeng, Z. Zhao, H. Zhu, and L. Sun, "Prosegdl: Binary protocol format extraction by deep learning-based field boundary identification," in *2022 IEEE 30th International Conference on Network Protocols (ICNP)*. IEEE, 2022, pp. 1–12.

[13] S. Zhao, S. Yang, Z. Wang, Y. Liu, H. Zhu, and L. Sun, "Crafting binary protocol reversing via deep learning with knowledge-driven augmentation," *IEEE/ACM Transactions on Networking*, vol. 32, no. 6, pp. 5399–5414, 2024.

[14] J. Li, G. Cheng, H. Tang, Y. Hu, and Q. Shang, "Private protocol reverse engineering via self-supervised learning-based message segmentation," *IEEE Transactions on Information Forensics and Security*, 2026.

[15] R. Rohith, M. Moharir, G. Shobha *et al.*, "Scapy-a powerful interactive packet manipulation program," in *2018 international conference on networking, embedded and wireless systems (ICNEWS)*. IEEE, 2018, pp. 1–5.

[16] The Wireshark Team, *Wireshark (Version 4.6.6)*, 2026, accessed: Jun. 16, 2026. [Online]. [Online]. Available: https://www.wireshark.org

[17] J. Narayan, S. K. Shukla, and T. C. Clancy, "A survey of automatic protocol reverse engineering tools," *ACM Computing Surveys (CSUR)*, vol. 48, no. 3, pp. 1–26, 2015.

[18] J. Duchêne, C. Le Guernic, E. Alata, V. Nicomette, and M. Kaâniche, "State of the art of network protocol reverse engineering tools," *Journal of Computer Virology and Hacking Techniques*, vol. 14, no. 1, pp. 53–68, 2018.

[19] S. Kleber, L. Maile, and F. Kargl, "Survey of protocol reverse engineering algorithms: Decomposition of tools for static traffic analysis," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 1, pp. 526–561, 2018.

[20] Y. Huang, H. Shu, F. Kang, and Y. Guang, "Protocol reverse-engineering methods and tools: A survey," *Computer Communications*, vol. 182, pp. 238–254, 2022.

[21] Y. Ye, Z. Zhang, F. Wang, X. Zhang, and D. Xu, "Netplier: Probabilistic network protocol reverse engineering from message traces." in *NDSS*, 2021.

[22] W. Cui, J. Kannan, and H. J. Wang, "Discoverer: Automatic protocol reverse engineering from network traces." in *USENIX Security Symposium*. Boston, MA, USA, 2007, pp. 1–14.

[23] S. Kleber, H. Kopp, and F. Kargl, "{NEMESYS}: Network message syntax reverse engineering by analysis of the intrinsic structure of individual messages," in *12th USENIX Workshop on Offensive Technologies (WOOT 18)*, 2018.

[24] S. Kleber and F. Kargl, "Refining network message segmentation with principal component analysis," in *2022 IEEE Conference on Communications and Network Security (CNS)*. IEEE, 2022, pp. 281–289.

[25] A. Rohl, M. Roughan, M. White, and A. Chambers, "Packet field tree: a hybrid approach for automated protocol reverse-engineering," in *2024 9th International Conference on Communication, Image and Signal Processing (CCISP)*. IEEE, 2024, pp. 160–169.

[26] J. Caballero, P. Poosankam, C. Kreibich, and D. Song, "Dispatcher: Enabling active botnet infiltration using automatic protocol reverse-engineering," in *Proceedings of the 16th ACM conference on Computer and communications security*, 2009, pp. 621–634.

[27] P. M. Comparetti, G. Wondracek, C. Kruegel, and E. Kirda, "Prospex: Protocol specification extraction," in *2009 30th IEEE Symposium on Security and Privacy*. IEEE, 2009, pp. 110–125.

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

[29] Y. A. Farrukh, I. Khan, S. Wali, D. Bierbrauer, J. A. Pavlik, and N. D. Bastian, "Payload-byte: A tool for extracting and labeling packet capture files of modern network intrusion detection datasets," in *2022 IEEE/ACM International Conference on Big Data Computing, Applications and Technologies (BDCAT)*. IEEE, 2022, pp. 58–67.

[30] V. Kiechle, M. Börsig, S. Nitzsche, I. Baumgart, and J. Becker, "Preunn: Protocol reverse engineering using neural networks." in *ICISSP*, 2022, pp. 345–356.

[31] B. Ning, X. Zong, K. He, and L. Lian, "Preiud: An industrial control protocols reverse engineering tool based on unsupervised learning and deep neural network methods," *Symmetry*, vol. 15, no. 3, p. 706, 2023.

[32] J. Garshasbi and M. Teimouri, "Cnnpre: A cnn-based protocol reverse engineering method," *IEEE Access*, vol. 11, pp. 116 255–116 268, 2023.

[33] G. Huo, Y. Huang, F. Kang, and H. Shu, "Chatpre: Knowledge-aware protocol analysis with llms for intelligent segmentation," *Journal of Network and Computer Applications*, vol. 249, p. 104426, 2026.

[34] Y. Huang, F. Kang, H. Shu, and G. Huo, "Malpre: Malware protocol reverse engineering through code slicing and agentic workflow," in *2025 IEEE 36th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2025, pp. 452–465.

[35] J. Li, G. Cheng, Z. Chen, and P. Zhao, "Protocol clustering of unknown traffic based on embedding of protocol specification," *Computers & Security*, vol. 136, p. 103575, 2024.

[36] A. Pagnoni, R. Pasunuru, P. Rodriguez, J. Nguyen, B. Muller, M. Li, C. Zhou, L. Yu, J. E. Weston, L. Zettlemoyer *et al.*, "Byte latent transformer: Patches scale better than tokens," in *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2025, pp. 9238–9258.

[37] S. Abeywickrama, E. Eldele, M. Wu, X. Li, and C. Yuen, "Entrope: Entropy-guided dynamic patch encoder for time series forecasting," *arXiv preprint arXiv:2509.26157*, 2025.

[38] J. Ling, X. Yang, S. Gong, and B. Gu, "Eapformer: Entropy-aware patch transformer for multivariate long-term time series forecasting," in *Proceedings of the 34th ACM International Conference on Information and Knowledge Management*, 2025, pp. 1850–1860.

Appendix

6.1. SLMSP as Data-Driven Baseline

Table 5

6.2. Hand-picked Folds Performance

Table 6

6.3. Hyperparameters

Table 7 and Table 8

TABLE 5. BOUNDARY F1 SCORE ON THE 12-PROTOCOL BENCHMARK. GROUP A: METHODS THAT NEVER SEE THE TARGET PROTOCOL (LOPO / TRACE-ONLY); SLMSP-TRANSFER IS AN OUT-OF-DESIGN ABLATION PLACING SLMSP IN THE SAME REGIME. GROUP B: SLMSP IN ITS INTENDED PER-PROTOCOL SETTING, TRAINING ON THE TARGET'S OWN TRAFFIC (SELF-SUPERVISED, NO LABELS), A STRICTLY EASIER REGIME THAN LOPO. SLMSP IS RUN WITH THE PAPER'S RECOMMENDED β=0.85 AND A FIXED TRAINING BUDGET (EL 30 EPOCHS, GLOVE 30 EPOCHS) UNIFORMLY ACROSS ALL PROTOCOLS [14]; NO PER-PROTOCOL TUNING. SLMSP-FULL EVALUATES FULL MESSAGES (UP TO 64 B); SLMSP-NATIVE TRUNCATES TO 16 B. BOLD: BEST WITHIN EACH GROUP. MEAN ± STD OVER 5 RANDOM SEEDS (SLMSP) OR 4 TRAINING RUNS (NEURINFERNO).

	Group A: No target data (LOPO / trace-only)						**Group B: Target traffic (per-protocol)**	
Protocol	NeurInferno	BI	Nemesys	NetPlier	Netzob	SLMSP-xfer‡	SLMSP-full†	SLMSP-nat†
ARP	**0.8130**±0.080	0.0000	0.5010	0.6013	0.6199	0.1337±0.039	**0.2278**±0.015	0.1491±0.058
BGP	**0.7985**±0.092	0.0000	0.5402	0.1054	0.0000	0.0000	**0.5258**±0.033	0.0000
DHCP	**0.5142**±0.088	0.2864	0.4092	0.2913	—	0.2690±0.019	**0.4634**±0.006	0.2642±0.032
DNS	**0.7065**±0.058	0.2458	0.2434	0.5885	0.2189	0.3582±0.012	**0.3744**±0.025	0.2627±0.086
ICMP	**0.9006**±0.003	0.1269	0.3864	0.6472	0.3840	0.3458±0.053	**0.4343**±0.022	0.3698±0.021
IGMP	**0.9167**±0.023	0.0000	0.3759	0.3911	0.2528	0.2783±0.021	**0.5105**±0.010	0.2476±0.021
IP	**0.9586**±0.037	0.5026	0.3694	0.7799	0.2013	0.7083±0.030	0.7150±0.037	**0.7341**±0.008
Modbus	**0.8228**±0.104	0.2930	0.4854	0.8081	0.5529	0.7845±0.009	0.6742±0.017	**0.6835**±0.022
NTP	**0.6753**±0.021	0.3357	0.4506	0.5037	0.0000	0.4545±0.017	0.3147±0.026	**0.5157**±0.038
OSPF	**0.7986**±0.022	0.1544	0.3537	0.5709	0.0803	0.2509±0.007	**0.3896**±0.009	0.2502±0.008
TCP	**0.9251**±0.017	0.1105	0.4168	0.6945	0.1808	0.3252±0.058	**0.5193**±0.023	0.3762±0.007
UDP	**0.8592**±0.054	0.2511	0.2471	0.4186	0.2362	0.3327±0.016	**0.3619**±0.030	0.3516±0.048
Avg.	0.8074±0.049	0.1922	0.3983	0.5334	0.2479	0.3534	0.4592	0.3504

†SLMSP trains on the *target* protocol's own traffic (self-supervised, no labels); NeurInferno withholds the target protocol entirely (LOPO) — a strictly harder regime. ‡SLMSP-transfer: SLMSP trained on the other 11 protocols and applied to the held-out one, matching NeurInferno's LOPO regime; byte#offset embeddings are protocol-specific, so transfer is out-of-design. BGP-raw SLMSP-native F1 = 0.000: bytes 0–15 are an all-0xFF sync marker (zero entropy); vertical correction removes all predictions and GT contains no boundaries in that window. SLMSP-full (0.526) captures the variable header at bytes 19+.

TABLE 6. BOUNDARY PREDICTION RESULTS UNDER LOPO AND L4PO EVALUATION. PRECISION (P), RECALL (R), AND F1 ARE REPORTED OVER THREE RANDOM SPLITS; BOLD INDICATES BEST P, R, AND F1 PER COLUMN PER SETTING.

Setting	**Method**	**Random Split 1**			**Random Split 2**			**Random Split 3**			**Average**		
		P	R	F1	P	R	F1	P	R	F1	P	R	F1
Evaluate on 1 Unknown Protocol (LOPO)	**Ours**	**0.801**	**0.999**	**0.889**	0.845	**0.839**	**0.842**	0.824	**0.797**	**0.810**	**0.823±0.022**	**0.878±0.107**	**0.847±0.040**
	BinaryInferno	0.750	0.176	0.285	**1.000**	0.156	0.270	0.200	0.042	0.070	0.650±0.409	0.125±0.072	0.208±0.120
	Nemesys	0.496	0.363	0.417	0.513	0.298	0.376	0.454	0.290	0.354	0.488±0.030	0.317±0.040	0.382±0.032
	NetPlier	0.690	0.708	0.695	0.565	0.300	0.391	0.667	0.499	0.571	0.641±0.067	0.502±0.204	0.552±0.153
	Netzob	0.400	0.117	0.181	0.667	0.156	0.253	**1.000**	0.042	0.080	0.689±0.301	0.105±0.058	0.171±0.087
Evaluate on 4 Unknown Protocols (L4PO)	**Ours**	0.709	**0.747**	**0.718**	0.689	**0.872**	**0.734**	**0.786**	**0.845**	**0.813**	0.728±0.051	**0.821±0.066**	**0.755±0.051**
	BinaryInferno	**0.757**	0.218	0.310	**0.695**	0.234	0.331	0.738	0.296	0.412	**0.730±0.032**	0.249±0.041	0.351±0.054
	Nemesys	0.591	0.371	0.447	0.649	0.356	0.453	0.498	0.347	0.403	0.579±0.076	0.358±0.012	0.434±0.027
	NetPlier	0.487	0.486	0.469	0.642	0.571	0.597	0.664	0.635	0.645	0.598±0.096	0.564±0.075	0.570±0.091
	Netzob	0.558	0.309	0.338	0.479	0.302	0.318	0.517	0.096	0.149	0.518±0.040	0.236±0.121	0.268±0.104

TABLE 7. NEURINFERNO ARCHITECTURE HYPERPARAMETERS.

Component	Hyperparameter	Value
Byte-GM	Architecture	Causal decoder-only Transformer
	Vocabulary	259 tokens: 256 byte values + PAD/BOS/EOS
	Hidden dimension	64
	Attention heads	4
	FFN dimension	256
	Transformer layers	2
	Maximum length	256 bytes
	Positional encoding	Learned absolute embeddings
	Activation / normalization	GELU / pre-norm Transformer blocks
	Dropout	0.1
ByteEncoder	Architecture	Bidirectional Transformer with cross-message injection
	Vocabulary	259 tokens
	Hidden dimension	128
	Attention heads	4
	FFN dimension	512
	Transformer layers	4
	Maximum length	512 bytes
	Positional encoding	Learned absolute embeddings
	Cross-message statistics	Mean, max, and variance across messages
Boundary head	Architecture	MLP over pairwise gap features
	Input dimension	$6D + 1 = 769$
	Hidden dimension	128
	Output	One boundary logit per byte gap
	Features	Left/right states, difference, product, entropy, variance
Field-type head	Architecture	MLP over byte representation
	Input / hidden dimension	128 / 128
	Output classes	17 field-type classes
	Usage	Present in the model; disabled in boundary-only experiments

TABLE 8. NEURINFERNO TRAINING AND EVALUATION HYPERPARAMETERS.

Stage	Hyperparameter	Value
Stage 1: ***Byte-GM***	Objective	Next-byte cross-entropy
	Optimizer	AdamW
	Learning rate	3×10^{-4}
	Schedule	OneCycleLR with cosine annealing
	Weight decay	10^{-2}
	Gradient clipping	1.0
	Batch size	128 sequences
	Maximum epochs	10
	Early stopping	Patience 5, using validation loss
Stage 2: Main training	Objective	Boundary-only training
	Boundary loss	Focal BCE
	Focal parameter	$\gamma = 2.0$
	Positive weighting	Dynamic negative/positive ratio, clipped to $[1, 100]$
	Optimizer	AdamW
	Learning rate	3×10^{-4}
	Schedule	Linear warmup followed by cosine annealing
	Maximum epochs	10
	ByteLM status	Frozen
Stage 3: LOPO fine-tuning	Initialization	Best Stage-2 checkpoint for the same exclusion set
	Optimizer	AdamW
	Learning rate	3×10^{-4}
	Maximum steps	2000
	Gradient clipping	1.0
	Messages per batch	100
	ByteLM status	Unfrozen
Evaluation	Boundary threshold	$\tau = 0.70$
	Max messages per protocol	100
	LOPO protocols	12 real protocols
	L4PO splits	13 held-out four-protocol splits
	Metrics	Boundary precision, recall, FPR, and F1
Data	Tier-1 data	500 grammar-derived synthetic protocol formats
	Tier-2 data	17 real Scapy-captured protocols
	Sequence lengths	256 bytes for ByteLM; 512 bytes for main model