Lens: A KNOWLEDGE-GUIDED FOUNDATION MODEL FOR NETWORK TRAFFIC

arXiv:2402.03646v5 [cs.LG] **14 Jan 2026** - **version 5!!**

Department of Software, Sookmyung Women's University
Jisoo Kim

Table of Contents

01 Introduction

ABSTRACT

Network traffic refers to the amount of data being sent and received over the Internet or any system that connects computers. Analyzing network traffic is vital for security and management, yet remains challenging due to the heterogeneity of plain-text packet headers and encrypted payloads. To capture the latent semantics of traffic, recent studies have adopted Transformer-based pretraining techniques to learn network representations from massive traffic data. However, these methods pre-train on data-driven tasks but overlook network knowledge, such as masking partial digits of the indivisible network port numbers for prediction, thereby limiting semantic understanding. In addition, they struggle to extend classification to new classes during fine-tuning due to the distribution shift. Motivated by these limitations, we propose Lens, a unified knowledge-guided foundation model for both network traffic classification and generation. In pretraining, we propose a Knowledge-Guided Mask Span Prediction method with textual context for learning knowledge-enriched representations. For extending to new classes in finetuning, we reframe the traffic classification as a closed-ended generation task and introduce context-aware finetuning to adapt the distribution shift. Evaluation results across various benchmark datasets demonstrate that the proposed Lens achieves superior performance on both classification and generation tasks. For traffic classification, Lens outperforms competitive baselines substantially on 8 out of 12 tasks with an average accuracy of **96.33%** and extends to novel classes with significantly better performance. For traffic generation, Lens generates better high-fidelity network traffic for network simulation, gaining up to **30.46%** and **33.3%** better accuracy and F1 in fuzzing tests. We will open-source the code upon publication.

1. Technique trends being used in recent studies – and some limitations found…
2. Lens Architecture
   - Pretraining – Knowledge-Guided MSP
   - Finetuning – closed-ended generation task & context-aware finetuning
3. Evaluation Results

01 Introduction – 2 limitations & RQs

Recent works have developed foundation models employing various pretraining techniques with promising results, however…

(1) Current data-driven pretraining methods **“overlook” vital network knowledge**
→ incomplete semantic understanding..

<Random Masking>

- port no. 40023 (indivisible)
  → masked as ___23
- TCP → masked as __CP

RQ1: How can we integrate network-specific knowledge into pretraining to learn better network representations?

(2) Most encoder-based models in finetuning for classification use an additional “MLP architecture”
→ **poor performance when extending to new classes** as they need to be re-trained for accommodation..

RQ2: How can we seamlessly extend traffic classification to novel classes while maintaining high performance?

01 Introduction

RQ1: How to address the “network-specific knowledge”?

RQ2: How to extend traffic classification to “novel classes”?

Lens : Unified knowledge-guided foundation model based on an encoder-decoder T5 architecture

- Enc-Dec architecture : better captures the global information → well suited for the header generation task based on its successor payload
  - Dec-only models (next token prediction) struggle with this due to their auto-regressive** nature

- RQ1) Knowledge-Guided Mask Span Prediction (KG-MSP) – which intentionally mask network metadata and payload-related information based on their IMPORTANCE in networking (NOT random masking)
  - Also, incorporate context infos as auxiliary knowledge into model pretraining for generalization

- RQ2) Reframe the network traffic classification as a closed-ended generation task** for adapting distribution shifts**
  - Leverage context-aware finetuning to smoothly extend classification from known to new classes, by training only on new-class data with the updated context

01 Introduction

[Extra Slide]

- **Auto-regressive model?
  - Sequentially predicts/generates the next token one by one by looking at ONLY the “previously created tokens” from left to right when creating a sentence
  - The structure that stably fixes “the entire input” as a condition is weaker than Enc-Dec architecture

- **Closed-ended generation task?
  - A generation problem where the correct answer candidates are predetermined! (closed set)
    - Input: Traffic + “list of label candidates(context)”
    - Output(generation): Exactly 1 label string
  - In other words, the model generates a label, but the correct answer must be one of the predefined labels, such as “Youtube”, “Skype”, “DoH”, or “VPN” → used constraint generation to achieve

- **Distribution Shift?
  - the data distribution seen during training ≠ the data distribution received during actual application
    - New apps/services emerge → Port/pattern/protocol usage changes
    - Changes in attack techniques → Packet length/sequence patterns change
  - Original method (encoder + MLP head) is a “closed classifier with a fixed label space” → the head structure needs to be changed and retrained when a new class is introduced, and performance can fluctuate if the distribution shifts during this process
  - However, Lens proposes a scenario where, by converting classification to generative, label candidates are inserted as “context” (list update allowed) and can be adapted only with new class data

01 Introduction

- Performance assess – 12 network traffic classification tasks & 5 network generation tasks across 6 datasets
- Traffic classification : Outperforms competitive baselined across 8 / 12 tasks with an average acc of 96.31%
  - Extends to classify novel attacks well – gained acc / F1 advantage up to 31.31% / 42.59%
- Traffic generation : achieved 30.46% higher acc and 33.3% higher F1 in network fuzzing tests

<Contribution Summary>

1. KG-MSP + Context for learning generalizable network representations
2. Reframe the “traffic classification” as a “closed-ended generation task” to adapt the distribution shift, which enables excellent extensibility by simply updating the textual context and lightly finetuning only on new classes.
3. Evaluate Lens on both network traffic classification and generation.

- Classification → Lens outperforms competitive baselines on most tasks and extends to novel classes significantly better.
- Generation → Lens generates high-fidelity network traffic that closely mirrors real-world distributions, enhancing the efficacy of subsequent fuzzing tests.

02 Related Works (Briefly)

1. Network Traffic Classification

- Classical ML methods – KNN, SVM, statistical features for RF, MMIRF, etc.
  - Require expert knowledge for feature extraction and lack generalization capability
- DL Techniques – CNN, RNN, LSTM, DeepPacket (SAE w/ CNN)
  - Rely heavily on large amounts of labeled data and have limited generalization ability
- Pre-training Approaches(ENC) – PERT, ET-BERT, netFound, NetMamba(hierarchical Transformer)
  - Pretrain on random masking & only applicable for classification due to their encoder-only structure
- Pre-training Approaches(DEC) – NetGPT, GBC, TrafficGPT
  - Predict tokens based on casual probabilities, inferring the value of network fields based on partial context (Even though these are pretrained on the next token prediction for both traffic classification & generation)
- NLP based Pre-training – TrafficLLM (fine-tuned LLM)
  - Suffer from hallucination & Requires more labeled data to mitigate the domain gap

02 Related Works (Briefly)

1. Network Traffic Classification

Table 1: Comparison of the proposed **Lens** and existing pretraining methods. "KG Pretrain" denotes knowledge-guided pretraining. "IP Masking" means all IPs are anonymized or removed in both pretraining and finetuning for privacy. "Generation" refers to the support of generation tasks.

Method	Encoder	Decoder	KG Pretrain	IP Mask	Generation
PERT [15]	✔	✘	✘	✘	✘
ET-BERT [25]	✔	✘	✘	✔	✘
NetGPT [30]	✘	✔	✘	✘	✔
YaTC [58]	✔	✘	✘	✔	✘
TrafficLLM [8]	✘	✔	✘	✔	✔
Lens(Ours)	✔	✔	✔	✔	✔

→ Let's pre-train a network foundation model with a knowledge-guided task combined with auxiliary context to learn better network representations!!

02 Related Works (Briefly)

2. Network Traffic Generation

- Tool-based Traffic Generation – simulation tools(NS-3, yans, DYNAMO) & structure-based solutions(Iperf, Harpoon, …)
  - Require vast domain expertise and might lack versatility & Often result in rigid traffic patterns
- GAN-based Traffic Generation – NetShare, DoppelGANger, etc.
  - Though these methods are adaptive, their generated results may be inconsistent with target protocols
- Pretraining-based Generation – Decoder-based NetGPT & TrafficGPT
  - Hard to assess their performance since they didn't compare result with the SOTA models like Netshare
- TrafficLLM
  - Requires more data and computational resources to achieve good performance

→ Let's leverage "context-aware finetuning" to generate high-fidelity network traffic available for downstream simulation!!

03 Overall Framework of LENS

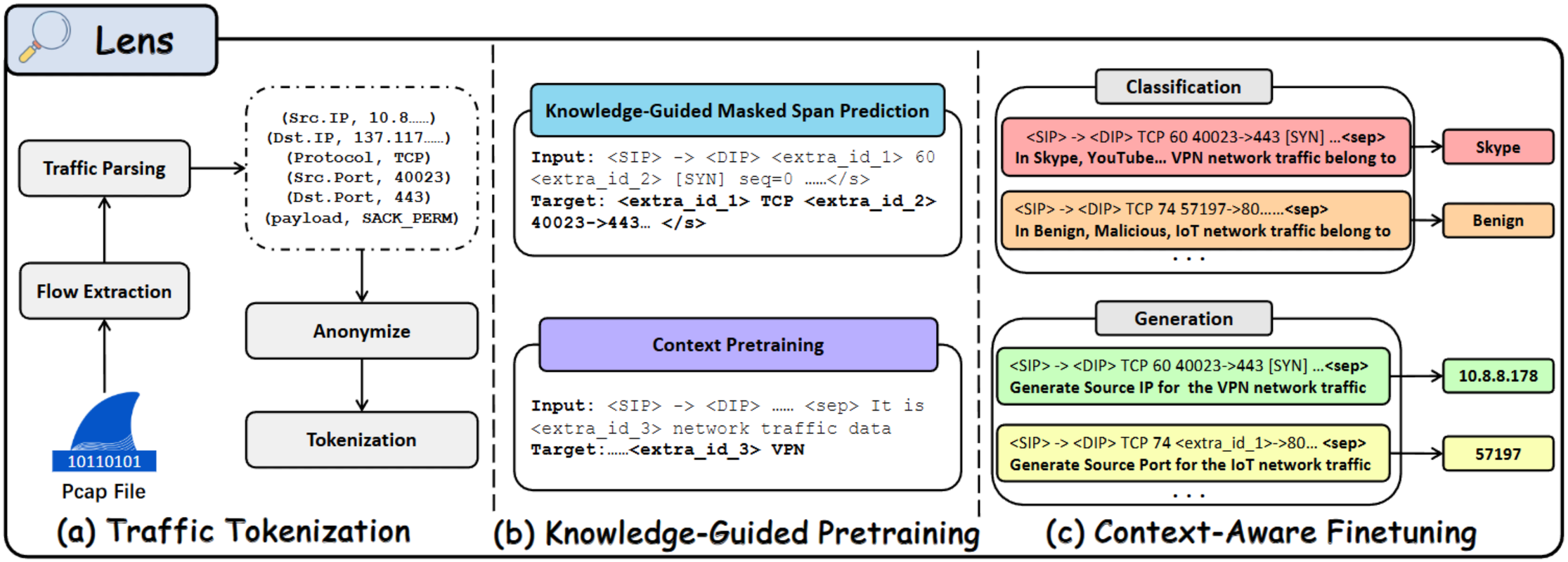
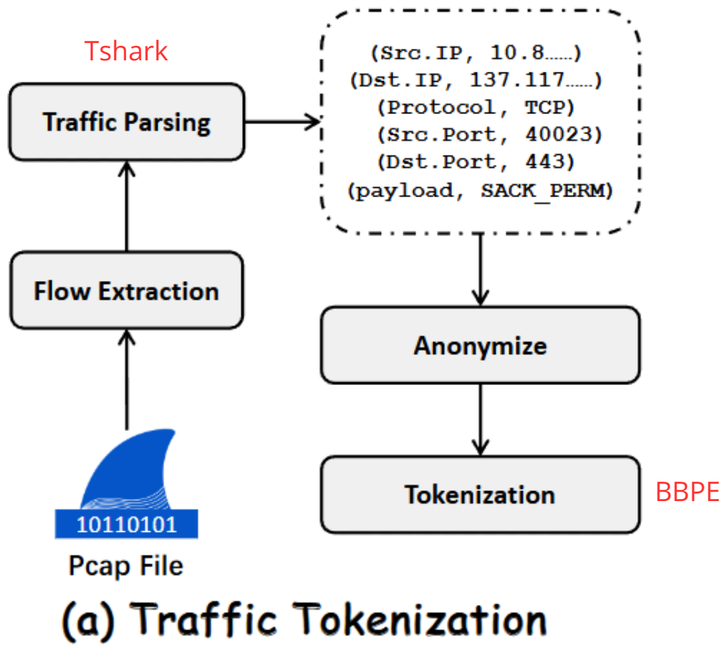


Figure 1: The overall framework of Lens. (a) Network flows are extracted, parsed with Tshark, anonymized, and tokenized using our network-specific tokenizer. (b) Lens is pretrained with Knowledge-Guided Masked Span Prediction (KG-MSP) and auxiliary natural-language context. (c) In finetuning, Lens performs downstream classification and generation tasks via context-aware finetuning.

03 (a) Traffic Tokenization

[Appendix]



Traffic Classification Data Example

Input: <SIP> → <DIP> TCP 60 34665 → 443 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM TSval=4177987 TSecr=0 WS=128
<DIP> → <SIP> TCP 60 443 → 34665 [SYN, ACK] Seq=0 Ack=1 Win=42540 Len=0 MSS=1350 SACK_PERM TSval=1083957698 TSecr=4177987 WS=128
<SIP> → <DIP> TCP 52 34665 → 443 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=4177994 TSecr=1083957698
<SIP> → <DIP> TLSv1 569 Client Hello 1603 0102 0001 0001 fc03 03a5
<DIP> → <SIP> TCP 52 443 → 34665 [ACK] Seq=1 Ack=518 Win=43648 Len=0 TSval=1083957724 TSecr=4177994
<DIP> → <SIP> TLSv1.2 201 Server Hello, Change Cipher Spec, Encrypted Handshake Message 1603 0300 5d02 0000 5903 0355
<SIP> → <DIP> TCP 52 34665 → 443 [ACK] Seq=518 Ack=150 Win=30336 Len=0 TSval=4178001 TSecr=1083957725
<SIP> → <DIP> TLSv1.2 103 Change Cipher Spec, Encrypted Handshake Message 1403 0300 0101 1603 0300 2800
<SIP> → <DIP> TLSv1.2 139 Application Data 1703 0300 5200 0000 0000 0000
<SIP> → <DIP> TCP 1390 [TCP segment of a reassembled PDU] 1703 0306 a000 0000 0000 0000
<SIP> → <DIP> TLSv1.2 415 Application Data ce79 99f6 3ab9 6c90 3ddd 5620
<SIP> → <DIP> TLSv1.2 1103 Application Data 1703 0304 1600 0000 0000 0000
<DIP> → <SIP> TLSv1.2 108 Application Data 1703 0300 3300 0000 0000 0000
<DIP> → <SIP> TLSv1.2 94 Application Data 1703 0300 2500 0000 0000 0000 ……
<sep> In aim email facebook ftps gmail hangout icq netflix scp sftp skype spotify torrent vimeo voipbuster youtube, the VPN network traffic belong to:</s>
Labels: Hangout</s>

Part of the payload hex is appended to the TLS handshake/Application Data!

task context example: In (…label candidates…), the VPN network traffic belong to:

Traffic Generation Data Example

Input: <SIP> → 93.189.89.83 TCP 322 POST /iWEab%20&%26%3DC/kXi_6r+j/Tb HTTP/1.1 [TCP segment of a reassembled PDU] 504f 5354 202f 6957 4561 6225 <sep> Generate source ip for the USTC-TFC2016 network traffic:</s>
Labels: <SIP>10.0.2.108</s>

<sep> is followed by Generation direction(prompt) like “Generate source ip…”

Figure 4: The example input of classification and generation. For classification, the input includes parsed network traffic and a task context listing label options. For generation, the input contains a masked packet and a task description.

- Parse network flows through Tshark into text-like input
- Afterwards, we pretrain a specialized Byte-level Byte Pair Encoding (BBPE) tokenizer on the textual network input to tokenize network terms, like “Seq”, “TCP”, and also the natural language context properly.

03 (a) Traffic Tokenization

[Extra Slide]

Q1. Why does Lens use BBPE Tokenizer? (my thoughts)

1. Network text is a unique language with "mixed character types"
   - Processing things like english token(TCP, TLSv1.2, Client Hello, SACK_PERM) / symbols(→, :, =, [SYN, ACK]) / number(port 34665, length 60, seq=0) / hex payload(1603 0102 ...) with a general NLP tokenizer designed for natural language, can result in stange fragmentation–breaking down meaning units–or an explosion of OOV/rare tokens
   - BBPE is byte-level, allowing any string to be broken down into bytes. BPE merge can then regroup frequently occurring patterns into meaningful chunks. Therefore, it's advantageous for acquiring a vocabulary specialized for network terms/patterns!

2. To enable "field-level masking" in KG-MSP, token boundaries must be cleanly defined
   - Network semantic units, such as port numbers and protocols, should be tokenized in a form that's easy to mask as "spans", without being excessively fragmented
   - In other words, the tokenizer must transform network terms/number patterns into "plausible chunks" for the "whole masking → restoration" objective to work effectively!

3. Since Lens also needs "natural language context", it must properly tokenize both network terms + natural language context

03 (a) Traffic Tokenization

[Extra Slide]

Q2. What does “pretraining BBPE tokenizer” mean? (my thoughts)

- Learning BPE merge rules and vocab to determin which string fragments are good tokens in this data
  - Results : vocab.json(token_to_id mapping), merges.txt(rules for which byte fragments should be merged and in what order)

> **Pretraining Data.** For each dataset (except CrossPlatforms), we randomly sample 60% of flows using a stratified sampling strategy, but only with respect to the coarsest dataset-level categories (e.g., VPN vs non-VPN, benign vs malicious), without using any fine-grained task labels such as application or service types. This ensures that pretraining does not access any downstream classification labels and completely avoids label leakage. The pretraining split is also used to pretrain our network-specific BBPE tokenizer, together with the mini C4 natural language corpus [31].

- network traffic data (use the pretraining split) + miniC4?
  - Training a tokenizer solely on network text can lead to strange fragmentation of natural language contexts (e.g., "It is ... network traffic data", "Generate ...")
  - Training solely on natural language can lead to broken network terms (e.g., SACK_PERM, TLSv1, Seq=)
  - Therefore, Lens combined the two to create a tokenizer that successfully processes both network and natural language.

Human-readable text (Tshark parsing result):
"<SIP> → <DIP> TCP 60 34665 → 443 [SYN] Seq=0 ..."
Internal tokenizer output (conceptually):
["<SIP>", "→", "<DIP>", "TCP", "60", "34665", "→", "443", "[", "SYN", "]", "Seq", "=", "0", ...]
And the final model input is an array of these tokens converted to integer IDs:
[512, 97, 513, 804, 45, 1290, 97, 902, ...]

1. Prepare the training text corpus : Lens' "Tshark-texted flow strings" + miniC4 sentences
2. Start with "byte-level basic units" as the initial state
3. Find frequently occurring pairs throughout the corpus
4. Merge the most frequent pairs into a new token
5. Repeat step 3–4 until the vocab size (e.g., 32kb/50kb) is reached, then terminate
6. The resulting vocab/merge rules are established, and new text is deterministically decomposed into tokens according to these rules

03 (a) Traffic Tokenization

[Appendix]

Preprocessing Pipeline

1. Segment raw traffic into 'flow' – using SplitCap based on the standard 5-tuple structure
2. Parse each flow with Tshark – extract the L3/L4/L5 metadata & append a 12 byte hex application payload
3. Anonymize all src/dest IP address – using 2 special placeholder tokens (<SIP>, <DIP>)

Table 9: Statistics of packet numbers and tokenized flow lengths of the dataset.

Dataset	0 percentile	25 percentile	50 percentile	75 percentile	100 percentile
Number of Packets in each flow	5	10	21	**34**	2,429,639
The Length of tokenized flow	175	567	973	**1,466**	126,871,264

- Truncated each flow to its first 34 packets, corresponding to the 75 percentile of the packet-length distribution
- This balances…
  - (1) retaining sufficient semantic information contained in early packets (e.g., 3-way handshake, protocol negotiation, application identification)
  - (2) maintaining a manageable context length for Transformer-based models

→ This strategy preserved >= 75% of classification-relevant signals while keeping the sequence length below 1.5k tokens, which is the maximum context length supported by TurboT5 in their hardware

03 (b) Knowledge-guided Pretraining

- Objective that intentionally masks significant network metadata and payload-related information entirely to learn generalizable representations for downstream tasks
  - What Lens does new is “what criteria do you use to select spans?” → based on network knowledge! (KG)

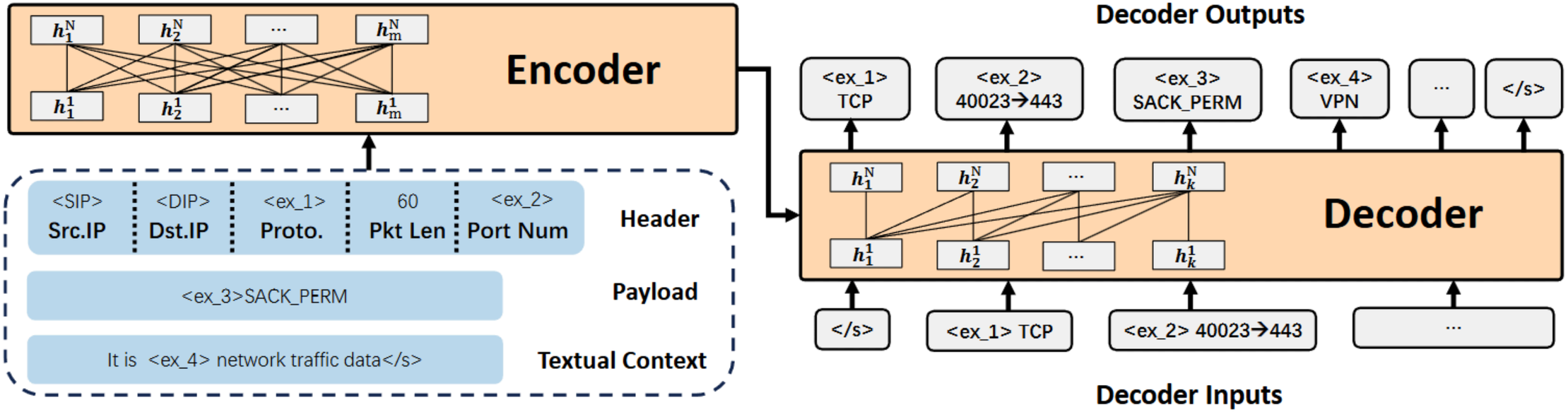


Figure 2: The core model architecture of Lens for pre-training with both the encoder and decoder. 1) The encoder takes in masked network traffic (header and payload) and textual template context. 2) The decoder uncovers the masked span tokens in both traffic and context based on Encoder representations in an auto-regressive way.

03 (b) Knowledge-guided Pretraining

1. Port numbers & protocol are essential for identifying application types
   - Set a θ(p_θ) probability to mask them for learning insightful network representations empirically
   - Both the src/dest port numbers are masked – for better understanding of the network traffic direction

2. Protocol flags and textual payload messages are essential for understanding packet intentions
   - Also set the same θ(p_θ) probability to mask them as a whole

3. Seq and ACK nums are fundamental for analyzing protocol behavior
   - Set a k(p_k) probability to mask seq/ack numbers and the length of packets and payloads

- To preserve the randomness in the masking policy, Lens further masked randomly on the rest of the input to satisfy the overall 15% masking ratio.

B.2 Sensitivity Analysis of Masking Configuration

Table 13a shows that **Lens**achieves its best validation performance when the masking ratio θ is 60%, and overall remains stable across different ratios. Similarly, Table 13b indicates that masking Seq/Ack/Length with ratio k also leads to minimal performance variation. This insensitivity arises because KG-MSP defaults to random masking when fewer than 15% of tokens are masked. For generality, we therefore adopt a 50% masking ratio.

Table 13: Sensitivity Analysis on Masking Vital Network Metadata (left) and Seq/Ack/Length (right).

(a) Analysis of θ for Masking Vital Metadata

Mask Prob.	30%	60%	90%
	AC	AC	AC
Lens (Ours)	0.33	**0.34**	0.32

(b) Analysis of k for Masking Seq/Ack/Length

Mask Prob.	25%	50%	75%
	AC	AC	AC
Lens (Ours)	**0.35**	**0.35**	0.34

03 (b) Knowledge-guided Pretraining

$$\mathcal{L}_{\text{KG-MSP}} = -\sum_{i=1}^{k} \log P(\text{KSPAN}_i = \text{SPAN}_i | \mathbf{x}_{\text{in}}, \text{KSPAN}_{<i}; \theta),$$

- KSPAN_i is generated given x_in and KSPAN<i when parameterized by θ
  - Here, Lens decodes the original span tokens after the special token used for masking in input.

    θ : the model (both encoder and decoder) parameters

    k : the total number of masked spans

    KSPAN_i : i-th span generated by the decoder part of Lens

    SPAN_i : the original span tokens

    x_in : the masked input sequence of the encoder

    KSPAN<i : the generated spans from the decoder before i-th span

03 (b) Knowledge-guided Pretraining

[Appendix]

B.1 Comparison with Random Masking Strategies

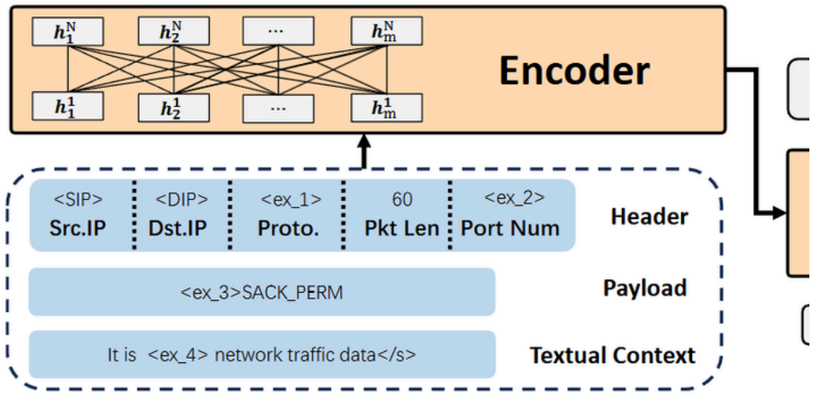
KG-MSP consistently outperforms random masking during pretraining. As shown in Table 12, Lensachieves +1.91% / +8.6% higher accuracy and +3.26% / +4.05% higher F1 on Task 2 and Task 3, respectively. This improvement stems from masking protocol-critical metadata and payload-related information rather than arbitrary spans. Since random masking here corresponds to the MASS-style/T5-style span masking objective [41], these results highlight the benefit of our knowledge-guided design.

Table 12: Ablation study of pretraining strategy with KG-MSP or Random Masking. Models pretrained with KG-MSP perform better than random masking.

Pretraining	Task 2		Task 3	
	AC	F1	AC	F1
Random Masking	0.8788	0.8567	0.7546	0.7732
KG-MSP(Ours)	**0.8979**	**0.8893**	**0.8406**	**0.8137**

03 (b) Knowledge-guided Pretraining

4. Context Pretraining – combine textual description of the input as “auxiliary knowledge”!

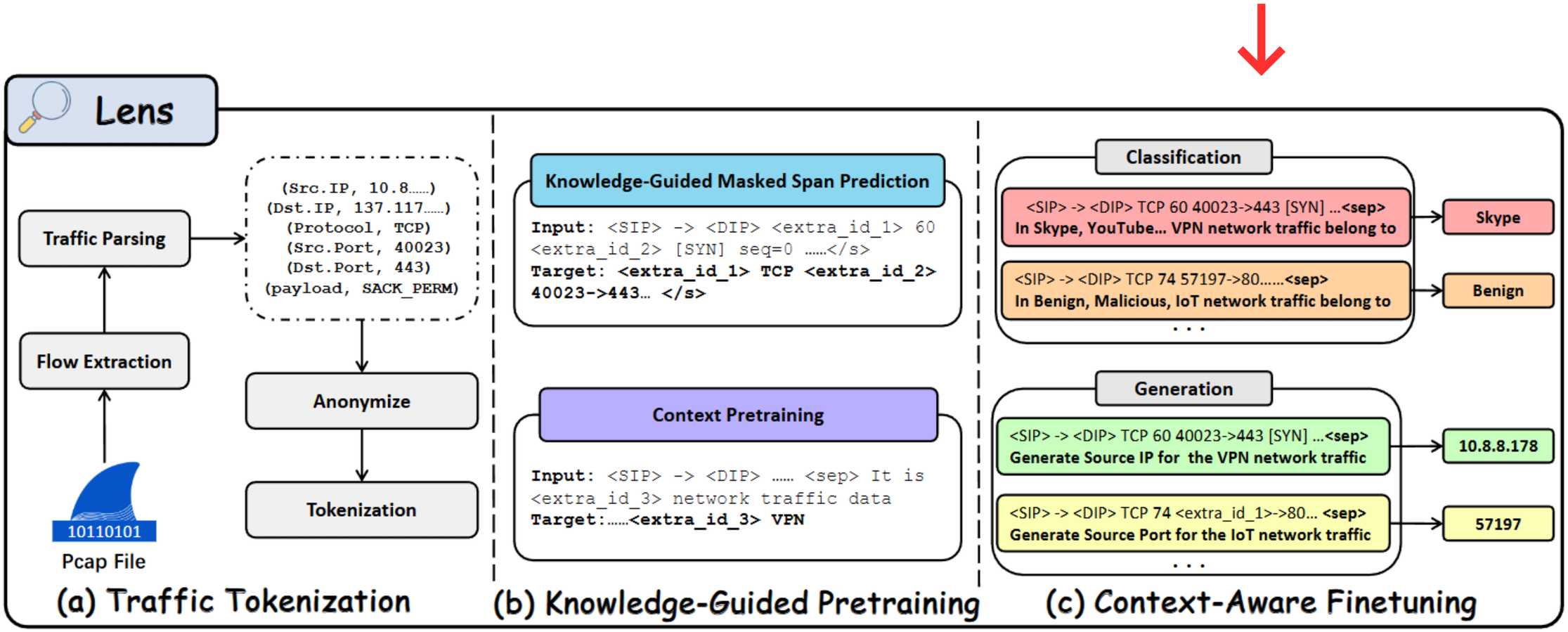


Encoder Input : [network traffic] <sep> [textual description]

- The textual description → derived solely from the dataset source and does not contain any downstream task labels, ensuring that no task-specific information is leaked!!

- Two-sided benefits:
  - The auxiliary description bridges the gap between similar network traffic via its sources(e.g, VPN, DoHBrw, IoT, ...)
  - The textual description bridges the gap between network traffic and natural language, preparing basic natural language understanding for downstream context-aware finetuning.
- Then How can we pretrain these auxiliary knowledge?
  - Randomly masked 15% of the textual description as NLP pretraining!

03 (c) Context-aware Finetuning



03 (c) Context-aware Finetuning

Skype → [**Sk**, ype]
Youtube→ [**You**, tube]
Yelp → [**Ye**, lp]
Zoom → [**Zo**, om]
First token candidate: {**Sk, You, Ye, Zo**}

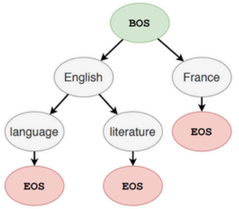
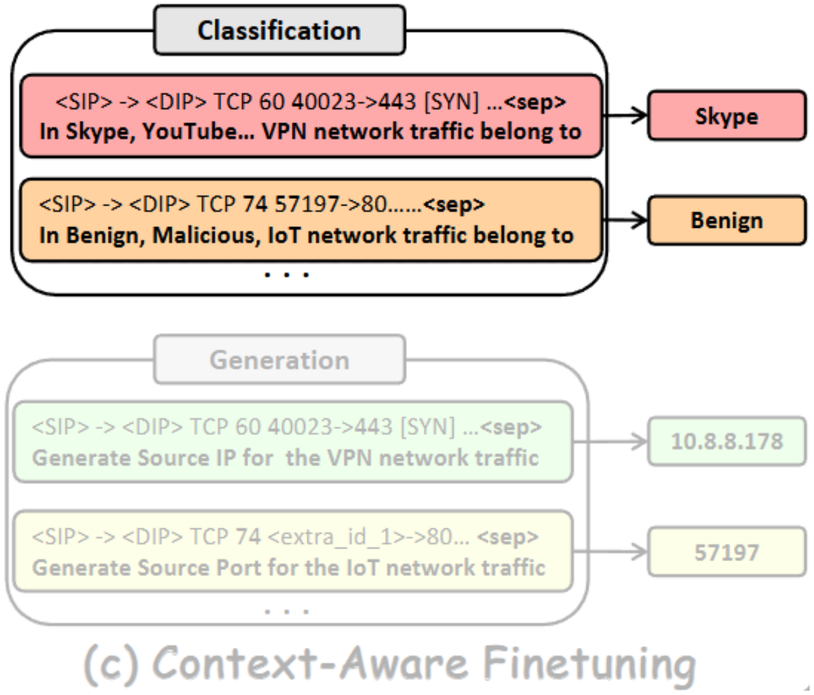


Figure 9: Example of prefix tree (trie) structure where the allowed entities identifiers are 'English language', 'English literature' and 'France'. Note that at the root there is the start-of-sequence token SOS and all leaves are end-of-sequence tokens EOS. Since more that one sequence has the same prefix (i.e., 'English'), this end up being an internal node where branches are the possible continuations.

1. Network Traffic Classification

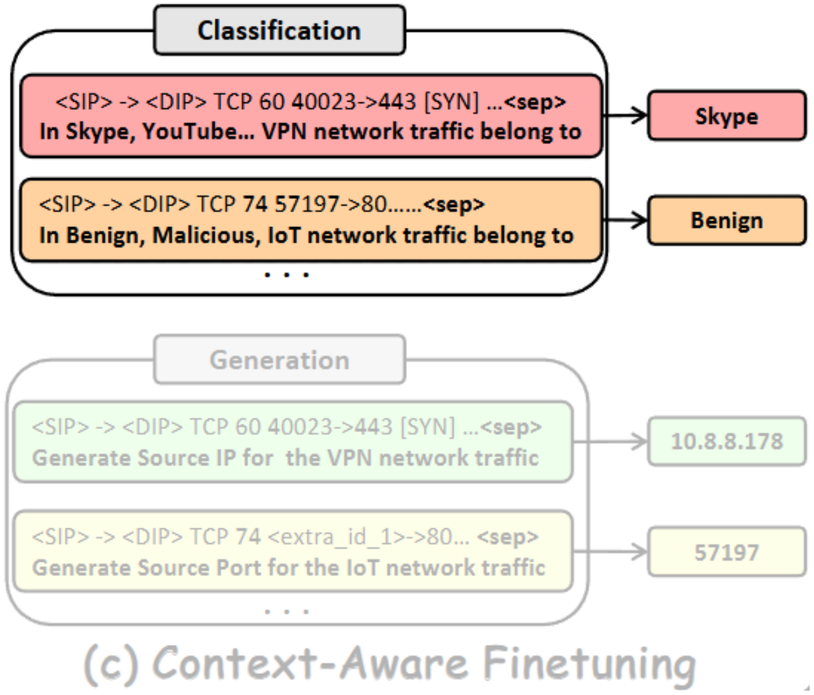


- Input format: [network traffic] <sep> [task description]

- Provide all application labels to select in the task context
  - potential for extensibility to unseen classes through adding more label options in the context
- To prevent generating labels out of choices, Lens guarantee the validity of the output by using a prefix trie as GENRE to constrain the generation process, ensuring the output sequence always matches a known label
  - Constrained decoding: filter "possible next token candidates" by trie, then eliminate tokens that are not in label string from the choices

03 (c) Context-aware Finetuning

2. Extending Network Classification to Novel Classes

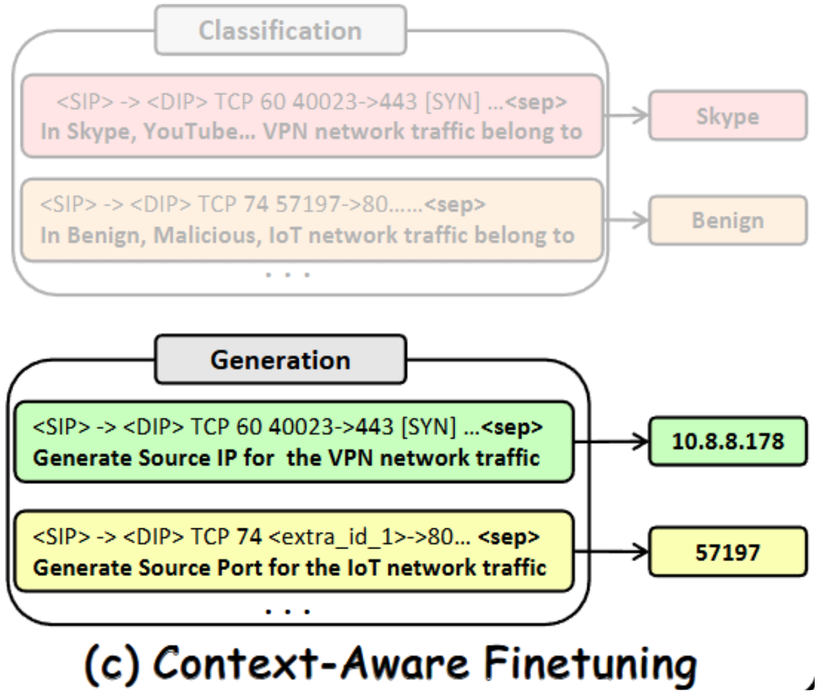


- Reforming traffic classification into a closed-ended generation task, Lens alleviates the distribution shifts and seamlessly extends to new classes via context-aware finetuning
- Process) Encounter unseen data from new classes → Reload the saved checkpoint to resume the learned knowledge on known classes first → add new label options in the task context and only fine-tune on unseen data(correlate new labels to unseen data)

→ Reuse learned knowledge for classifying new classes seamlessly through discerning the differences in the task context!

03 (c) Context-aware Finetuning

3. Network Traffic Generation



- Input format: [network traffic] <sep> [task description]
- Substitute the task context with various generation targets
- Mask the corresponding fields in the network traffic input for generation
- After generating, evaluate the fidelity of synthetic network traffic on FUZZING tests to verify its usefulness

- Generating encrypted payloads is lower priority...
  - Instead, replay captured payloads or apply standard encryption algorithm(e.g., AES)

04 Experiments

[Appendix]

4.1 Implementation details & experimental settings (Quite Detailed…!!)

- 6 publicly available datasets in NetBench
  - ISCX-VPN, ISCX-Tor, USTC-TFC-2016, Cross Platform, CIC-DoHBrw-2020, CIC-IoT-2023
- 12 downstream network traffic classification & 5 network generation tasks

Dataset	Pretraining	Finetuning
ISCX-VPN[11]	60%	24%
ISCX-Tor[14]	60%	24%
USTC-TFC-2016[51]	60%	5%
Cross Platform (Android)[46]	NA	15%
Cross Platform (IOS)[46]	NA	26%
CIC-DoHBrw-2020[32]	60%	6%
CIC-IoT-2023[35]	60%	5%

- Cross Platform dataset → excluded from pretraining; generalization evaluation in finetuning

- Pretraining Data
  - Randomly sample 60% of flows using stratified sampling strategy – only with respect to the COAREST dataset-level categories (e.g., VPN vs non-VPN / benign vs malicious)
  - Ensures that pretraining doesn't access any downstream classification labels and completely avoid label leakage
  - Also used to pretrain the network-specific BBPE tokenizer

04 Experiments

[Appendix]

4.1 Implementation details & experimental settings (Quite Detailed...!!)

- 6 publicly available datasets in NetBench
  - ISCX-VPN, ISCX-Tor, USTC-TFC-2016, Cross Platform, CIC-DoHBrw-2020, CIC-IoT-2023
- 12 downstream network traffic classification & 5 network generation tasks

Dataset	Pretraining	Finetuning
ISCX-VPN[11]	60%	24%
ISCX-Tor[14]	60%	24%
USTC-TFC-2016[51]	60%	5%
Cross Platform (Android)[46]	NA	15%
Cross Platform (IOS)[46]	NA	26%
CIC-DoHBrw-2020[32]	60%	6%
CIC-IoT-2023[35]	60%	5%

- Cross Platform dataset → excluded from pretraining; generalization evaluation in finetuning

- Finetuning Data
  - Randomly select around 10k labeld flows per dataset/task
    - This stage requires sufficient data to learn various downstream label strings (e.g., “YouTube”, “Skype”)
  - Preserved the imbalanced nature in both training and testing like real-world situation

1. flow-level classification
2. packet-level generation

04 Experiments

[Appendix]

4.1 Implementation details & experimental settings (Quite Detailed...!!)

	Hyperparameter	Value
Architecture	Transformer Encoder Layer number	12
	Transformer Decoder Layer number	12
	Attention heads number	12
	Attention heads dimension	64
	Hidden dimension	768
	MLP hidden	2048
	MLP activation	Gated-GeLU
Pre-training	Total gradient steps	780k
	Batch size	48
	Learning rate	1×5^{-4}
	Dropout rate	0.1
	Optimizer	AdamW
	Scheduler	Warmup cosine
	Grad clip norm	1.0
	Scheduler warmup steps	13k
Fine-tuning	Total Max epochs	40
	Batch size	32
	Learning rate	1×5^{-5}
	Dropout rate	0.1
	Optimizer	AdamW
	Grad clip norm	1.0
	Scheduler	Warmup linear

- Google's T5-v1.1-base model – contains roughly 0.25B parameters
  - TurboT5's implementation to process the input length up to 1,500
- GPU server with 4 NVIDIA A6000 48G GPUs, Ubuntu (20.04.6)
- Pretraining
  - batch size: 48
  - gradient accumulation over 6 training steps
  - lr: 5e-4 | total 130,000 steps | 10% steps for lr warmup
- Finetuning
  - batch size: 32
  - dropout rate: 0.1 | lr: 5e-5
  - train each downstream task for 20 epochs using AdamW

04 Experiments

4.1 Implementation details & experimental settings (Quite Detailed…!!)

1. Traffic Classification:

- Baselines:
  - DL baselines: FS-Net, BiLSTM_Att, Datanet, DeepPacket, TSCRNN
  - Competitive pretraining-based methods: YaTC, TrafficLLM, ET-BERT
- Metrics: Accuracy(AC), Macro F1 Score (F1)

2. Traffic Generation:

- Baselines: GAN-based Netshare, pretraining-based TrafficLLM
- Metrics: Jensen-Shannon Divergence (JSD), Total Variation Distance (TVD) – followed by NetGPT's setting
  - JSD : "How similar two probability distributions are"
  - TVD : "Largest difference in probabilities between two distributions"
  - GOOD performance? → low JSD & TVD !!

4.2.1 Traffic Classification Performance

Table 11: Overview of 12 Classification Tasks Across Datasets

Task #	Dataset	Task Description	Classes
Task 1	ISCX-VPN	VPN detection	2
Task 2	ISCX-VPN	VPN Service detection	6
Task 3	ISCX-VPN	VPN application classification	16
Task 4	ISCX-Tor and USTC-TFC-2016	Tor service detection	7
Task 5	USTC-TFC-2016	Application Classification	16
Task 6	Cross Platform (Android)	Application classification	209
Task 7	Cross Platform (Android)	Country detection	3
Task 8	Cross Platform (iOS)	Application classification	196
Task 9	Cross Platform (iOS)	Country detection	3
Task 10	CIC-DoHBrw-2020 (DoH)	DoH query method classification	5
Task 11	CIC-IoT-2023	IoT attack detection	2
Task 12	CIC-IoT-2023	IoT attack method detection	7

04 Experiments

NOTE: Bold (best results) / Underlined (second best)

4.2.1 Traffic Classification Performance

Method	Task 1		Task 2		Task 3		Task 4		Task 5		Task 6	
	AC	F1	AC	F1	AC	F1	AC	F1	AC	F1	AC	F1
FS-Net	0.9785	0.9537	0.7360	0.6732	0.5681	0.5910	0.9271	0.7606	0.8074	0.8817	0.2822	0.1219
BiLSTM_Att	0.9798	0.9551	0.8009	0.7719	0.6103	0.6635	0.9421	0.6095	0.9463	0.9568	0.8091	0.6023
Datanet	0.9726	0.9405	0.7755	0.7209	0.5762	0.5717	0.9362	0.5483	0.9397	0.9540	0.7081	0.4566
DeepPacket	0.9603	0.9178	0.7934	0.7585	0.6137	0.6834	0.9410	0.6405	0.9372	0.9456	0.7993	0.5622
TSCRNN	0.9668	0.9233	0.7975	0.7568	0.6028	0.6483	0.9368	0.6049	0.9412	0.9513	0.9072	0.8319
YaTC	0.9834	0.9627	0.8078	0.7805	0.6420	0.6990	0.9362	0.6956	0.9533	**0.9707**	0.9409	0.8173
TrafficLLM	0.9083	0.8084	0.6757	0.4826	0.5092	0.4650	0.9415	0.7076	0.6598	0.6363	NA	NA
ET-Bert	0.9863	0.9692	0.8130	0.8033	0.6484	0.6662	0.9296	0.6336	0.9462	0.9604	**0.9800**	**0.8925**
Lens (Ours)	**0.9942**	**0.9870**	**0.8979**	**0.8893**	**0.8406**	**0.8137**	**0.9692**	**0.8120**	**0.9538**	0.9676	0.9660	0.8847

Method	Task 7		Task 8		Task 9		Task 10		Task 11		Task 12		Avg.
	AC	F1	AC	F1	AC	F1	AC	F1	AC	F1	AC	F1	AC
FS-Net	0.8552	0.5157	0.2457	0.1052	0.3605	0.1767	0.5275	0.1381	0.3808	0.2758	0.9448	0.4270	0.6345
BiLSTM_Att	0.9440	0.8519	0.9080	0.8414	0.9563	0.9567	0.9781	0.7024	0.9809	0.9798	0.9695	0.4283	0.9021
Datanet	0.9141	0.7855	0.6527	0.4508	0.9343	0.9347	0.9742	0.6564	0.9799	0.9788	0.9454	0.4518	0.8591
DeepPacket	0.9291	0.8210	0.6330	0.3435	0.9412	0.9414	0.9774	0.7321	0.9814	0.9804	0.9655	0.4714	0.8727
TSCRNN	0.9267	0.7953	0.9072	0.8319	0.9510	0.9513	0.9781	0.7024	0.9800	0.9790	0.9698	0.4604	0.9054
YaTC	0.9960	0.9896	0.9552	0.9160	0.9963	0.9962	0.9905	0.9083	0.9864	0.9857	0.9586	0.5885	0.9289
TrafficLLM	0.9473	0.8519	NA	NA	0.9865	0.9864	0.4680	0.2461	0.7927	0.7439	0.5472	0.1844	0.7436
ET-Bert	0.9944	0.9855	**0.9788**	0.9456	**0.9988**	**0.9987**	0.9837	0.8151	0.9851	0.9842	0.9455	0.4296	0.9325
Lens (Ours)	**0.9960**	**0.9898**	0.9752	**0.9492**	0.9951	0.9951	**0.9963**	**0.9610**	**0.9877**	**0.9870**	**0.9878**	**0.6802**	**0.9633**

Table 2, 3 : Performance on traffic classification

Pretraining-based models mostly outperform deep learning models → benefits of pretrained network representations!

** TrafficLLM → suffers from the modality gap between network input and NL & imbalanced label distribution
→ necessity of pre-training network foundation models on network traffic data

04 Experiments

NOTE: Bold (best results) / Underlined (second best)

4.2.1 Traffic Classification Performance

Method	Task 1		Task 2		Task 3		Task 4		Task 5		Task 6	
	AC	F1	AC	F1	AC	F1	AC	F1	AC	F1	AC	F1
FS-Net	0.9785	0.9537	0.7360	0.6732	0.5681	0.5910	0.9271	0.7606	0.8074	0.8817	0.2822	0.1219
BiLSTM_Att	0.9798	0.9551	0.8009	0.7719	0.6103	0.6635	0.9421	0.6095	0.9463	0.9568	0.8091	0.6023
Datanet	0.9726	0.9405	0.7755	0.7209	0.5762	0.5717	0.9362	0.5483	0.9397	0.9540	0.7081	0.4566
DeepPacket	0.9603	0.9178	0.7934	0.7585	0.6137	0.6834	0.9410	0.6405	0.9372	0.9456	0.7993	0.5622
TSCRNN	0.9668	0.9233	0.7975	0.7568	0.6028	0.6483	0.9368	0.6049	0.9412	0.9513	0.9072	0.8319
YaTC	0.9834	0.9627	0.8078	0.7805	0.6420	0.6990	0.9362	0.6956	0.9533	**0.9707**	0.9409	0.8173
TrafficLLM	0.9083	0.8084	0.6757	0.4826	0.5092	0.4650	0.9415	0.7076	0.6598	0.6363	NA	NA
ET-Bert	0.9863	0.9692	0.8130	0.8033	0.6484	0.6662	0.9296	0.6336	0.9462	0.9604	**0.9800**	**0.8925**
Lens (Ours)	**0.9942**	**0.9870**	**0.8979**	**0.8893**	**0.8406**	**0.8137**	**0.9692**	**0.8120**	**0.9538**	0.9676	0.9660	0.8847

Method	Task 7		Task 8		Task 9		Task 10		Task 11		Task 12		Avg.
	AC	F1	AC	F1	AC	F1	AC	F1	AC	F1	AC	F1	AC
FS-Net	0.8552	0.5157	0.2457	0.1052	0.3605	0.1767	0.5275	0.1381	0.3808	0.2758	0.9448	0.4270	0.6345
BiLSTM_Att	0.9440	0.8519	0.9080	0.8414	0.9563	0.9567	0.9781	0.7024	0.9809	0.9798	0.9695	0.4283	0.9021
Datanet	0.9141	0.7855	0.6527	0.4508	0.9343	0.9347	0.9742	0.6564	0.9799	0.9788	0.9454	0.4518	0.8591
DeepPacket	0.9291	0.8210	0.6330	0.3435	0.9412	0.9414	0.9774	0.7321	0.9814	0.9804	0.9655	0.4714	0.8727
TSCRNN	0.9267	0.7953	0.9072	0.8319	0.9510	0.9513	0.9781	0.7024	0.9800	0.9790	0.9698	0.4604	0.9054
YaTC	0.9960	0.9896	0.9552	0.9160	0.9963	0.9962	0.9905	0.9083	0.9864	0.9857	0.9586	0.5885	0.9289
TrafficLLM	0.9473	0.8519	NA	NA	0.9865	0.9864	0.4680	0.2461	0.7927	0.7439	0.5472	0.1844	0.7436
ET-Bert	0.9944	0.9855	**0.9788**	0.9456	**0.9988**	**0.9987**	0.9837	0.8151	0.9851	0.9842	0.9455	0.4296	0.9325
Lens (Ours)	**0.9960**	**0.9898**	0.9752	**0.9492**	0.9951	0.9951	**0.9963**	**0.9610**	**0.9877**	**0.9870**	**0.9878**	**0.6802**	**0.9633**

Table 2, 3 : Performance on traffic classification

#2, 3 → Len's robust performance on the imbalanced test set

#4, 10, 12 → Len's pretraining on KG-MSP with the context has learned generalizable network representations, better capturing networking semantics, and the context-aware finetuning

#6, 8 → In contrast to Lens, TrafficLLM deteriorates as finetuning on limited data fails to mitigate the modality gap and learns the large label sets well

#12 → it can not adapt well here also, due to the skewed test label distribution

04 Experiments

NOTE: Bold (best results) / Underlined (second best)

4.2.1 Traffic Classification Performance

Method	Task 12	
	AC	F1
FS-Net	0.9448	0.4270
BiLSTM_Att	0.9695	0.4283
Datanet	0.9454	0.4518
DeepPacket	0.9655	0.4714
TSCRNN	0.9698	0.4604
YaTC	0.9586	0.5885
TrafficLLM	0.5472	0.1844
ET-Bert	0.9455	0.4296
Lens (Ours)	**0.9878**	**0.6802**

Table 3 – Task 12

Attack Type	Sample Size (n)	ET-BERT	YaTC	Lens (Ours)	
DDoS	11055	0.973	0.979	**0.995**	Head
DoS	1216	0.746	0.795	**0.968**	
Spoofing	55	0.486	0.667	**0.779**	Long-tail
Web-based	10	–	–	**0.308**	
BruteForce	5	–	0.500	**0.571**	

Table 4 : The case study on Task 12 (IoT method detection)

- Lens outperforms baselines well on both head and long-tailed classes [n: 5-11k]
  - accurately distinguishes DDoS from DoS
  - effectively recognizes long-tailed classes, such as Web-based and BruteForce (whereas other baselines malfunctioned on these classes)

04 Experiments

4.2.2 Traffic Extensibility Performance

Method	Task 6		Task 8	
	AC	F1	AC	F1
FS-Net	0.2822	0.1219	0.2457	0.1052
BiLSTM_Att	0.8091	0.6023	0.9080	0.8414
Datanet	0.7081	0.4566	0.6527	0.4508
DeepPacket	0.7993	0.5622	0.6330	0.3435
TSCRNN	0.9072	0.8319	0.9072	0.8319
YaTC	0.9409	0.8173	0.9552	0.9160
TrafficLLM	NA	NA	NA	NA
ET-Bert	**0.9800**	**0.8925**	**0.9788**	0.9456
Lens (Ours)	0.9660	0.8847	0.9752	**0.9492**

*LwF(Learning without Forgetting)?

: Regularization method that keeps the output (soft target) of the old model like distillation when learning new data, so that previous knowledge is less forgotten.

- First, finetune models on old classes until convergence → finetune only on data samples from new classes → incorporate new-class options into Lens's finetuning context (whereas ET-BERT extends its MLP classifier by adding output dimensions for the new classes)
  → close-ended generation!
  → more natural!
- ET-BERT-LwF: incorporate LwF mechanism into ET-BERT's classifier

Table 5 : Extensibility performance on #6, #8

Scenarios	Task 6 (209 classes)						Task 8 (196 classes)					
	1 new		3 new		5 new		1 new		3 new		5 new	
	AC	F1	AC	F1	AC	F1	AC	F1	AC	F1	AC	F1
ET-BERT	0.8536	0.7832	0.6387	0.4400	0.7137	0.5906	0.8257	0.8026	0.8477	0.8695	0.6079	0.5762
ET-BERT-LwF	0.9378	**0.8688**	0.8783	0.8625	0.8261	**0.8269**	0.9211	**0.8941**	0.8416	**0.9174**	0.7862	**0.8940**
Lens(Ours)	**0.9565**	0.8578	**0.9518**	**0.8659**	**0.9199**	0.8264	**0.9397**	0.8861	**0.8962**	0.8801	**0.8730**	0.8407

- Lens always achieve better accuracy than ET-BERT-LwF
- They aim to show their model's extensibility to new classes rather than developing new continual learning methods
  → future work!

04 Experiments

4.3 Lens's traffic generation capability

Datasets	Method	JSD ↓					TVD ↓				
		Src IP	Dst IP	Src Port	Dst Port	Len	Src IP	Dst IP	Src Port	Dst Port	Len
ISCX-VPN	NetShare	0.3591	0.3787	0.6539	0.5893	0.6793	0.5802	0.5948	0.9632	0.9137	0.9893
	TrafficLLM	**0.0946**	0.1175	0.5742	0.0430	0.0513	0.1849	0.1772	0.5920	0.0600	0.0845
	Lens (Ours)	0.0974	**0.0905**	**0.5574**	**0.0271**	**0.0338**	**0.1719**	**0.1245**	**0.5789**	**0.0343**	**0.0469**
ISCXTor	NetShare	0.3084	0.4160	0.5835	0.5736	0.6531	0.4930	0.6436	0.8813	0.8807	0.9756
	TrafficLLM	0.0023	**0.3629**	**0.5635**	0.1770	**0.0359**	0.0047	**0.4519**	**0.5838**	0.2339	**0.0500**
	Lens (Ours)	**0.0022**	0.4842	0.5826	**0.1337**	0.0398	**0.0038**	0.5620	0.6133	**0.1877**	0.0560
USTC-TFC	NetShare	0.4415	0.5065	0.5731	0.5885	0.6826	0.6876	0.7794	0.9037	0.9114	0.9933
	TrafficLLM	**0.3702**	**0.4186**	**0.3496**	0.2746	0.0159	0.3915	**0.4731**	**0.3656**	0.3261	0.0259
	Lens (Ours)	0.3783	0.4361	0.3864	**0.2685**	**0.0143**	**0.3910**	0.4748	0.4076	**0.2901**	**0.0203**
Cross Platform (IOS)	NetShare	0.3304	0.5267	0.6056	0.6104	0.6120	0.3389	0.5188	0.9289	0.9318	0.9491
	TrafficLLM	**0.0003**	**0.2433**	**0.5784**	0.0134	**0.0521**	**0.0006**	**0.3374**	**0.6006**	0.0265	**0.0662**
	Lens (Ours)	**0.0003**	0.3241	0.6508	**0.0083**	0.0608	**0.0006**	0.4523	0.6746	**0.0132**	0.0780
Cross Platform (AN)	NetShare	0.3459	0.3964	0.6219	0.6269	0.6312	0.5437	0.6205	0.9353	0.9443	0.9598
	TrafficLLM	0.0016	**0.1969**	**0.6011**	0.0082	**0.0672**	0.0216	**0.3101**	**0.6220**	0.0377	**0.0835**
	Lens (Ours)	**0.0003**	0.2809	0.6531	**0.0046**	0.0690	**0.0041**	0.4166	0.6796	**0.0130**	0.0872
CIRA-CIC-DoHBrw	NetShare	0.3955	0.6117	0.5446	0.4630	0.6566	0.3886	0.6315	0.8825	0.7754	0.9776
	TrafficLLM	0.0162	**0.3782**	**0.4858**	**0.0001**	0.0614	0.0838	**0.4676**	**0.5258**	**0.0003**	0.1069
	Lens (Ours)	**0.0041**	0.4105	0.6915	**0.0001**	**0.0481**	**0.0246**	0.4896	0.7065	**0.0003**	**0.0728**
CIC-IoT-2023	NetShare	0.0732	0.0804	0.5939	0.1188	0.6807	0.2155	0.2273	0.8920	0.2936	0.9904
	TrafficLLM	0.0598	0.0345	0.5544	0.0586	**0.0039**	0.1523	0.1072	0.5844	0.0833	0.0061
	Lens (Ours)	**0.0146**	**0.0098**	**0.0179**	**0.0262**	**0.0039**	**0.0779**	**0.0325**	**0.0217**	**0.0295**	**0.0058**

1. Lens's significantly better understanding of destination port numbers
   - by the KG-MSP strategy!
2. Still surpasses TrafficLLM even not pretrained on the Cross platform dataset
   - → Port semantics generalizes well to out-of-domain datasets!

04 Experiments

4.3 Lens's traffic generation capability

Datasets	Method	JSD ↓					TVD ↓				
		Src IP	Dst IP	Src Port	Dst Port	Len	Src IP	Dst IP	Src Port	Dst Port	Len
ISCX-VPN	NetShare	0.3591	0.3787	0.6539	0.5893	0.6793	0.5802	0.5948	0.9632	0.9137	0.9893
	TrafficLLM	**0.0946**	0.1175	0.5742	0.0430	0.0513	0.1849	0.1772	0.5920	0.0600	0.0845
	Lens (Ours)	0.0974	**0.0905**	**0.5574**	**0.0271**	**0.0338**	**0.1719**	**0.1245**	**0.5789**	**0.0343**	**0.0469**
ISCXTor	NetShare	0.3084	0.4160	0.5835	0.5736	0.6531	0.4930	0.6436	0.8813	0.8807	0.9756
	TrafficLLM	0.0023	**0.3629**	**0.5635**	0.1770	**0.0359**	0.0047	**0.4519**	**0.5838**	0.2339	**0.0500**
	Lens (Ours)	**0.0022**	0.4842	0.5826	**0.1337**	0.0398	**0.0038**	0.5620	0.6133	**0.1877**	0.0560
USTC-TFC	NetShare	0.4415	0.5065	0.5731	0.5885	0.6826	0.6876	0.7794	0.9037	0.9114	0.9933
	TrafficLLM	**0.3702**	**0.4186**	**0.3496**	0.2746	0.0159	0.3915	**0.4731**	**0.3656**	0.3261	0.0259
	Lens (Ours)	0.3783	0.4361	0.3864	**0.2685**	**0.0143**	**0.3910**	0.4748	0.4076	**0.2901**	**0.0203**
Cross Platform (IOS)	NetShare	0.3304	0.5267	0.6056	0.6104	0.6120	0.3389	0.5188	0.9289	0.9318	0.9491
	TrafficLLM	**0.0003**	**0.2433**	**0.5784**	0.0134	**0.0521**	**0.0006**	**0.3374**	**0.6006**	0.0265	**0.0662**
	Lens (Ours)	**0.0003**	0.3241	0.6508	**0.0083**	0.0608	**0.0006**	0.4523	0.6746	**0.0132**	0.0780
Cross Platform (AN)	NetShare	0.3459	0.3964	0.6219	0.6269	0.6312	0.5437	0.6205	0.9353	0.9443	0.9598
	TrafficLLM	0.0016	**0.1969**	**0.6011**	0.0082	**0.0672**	0.0216	**0.3101**	**0.6220**	0.0377	**0.0835**
	Lens (Ours)	**0.0003**	0.2809	0.6531	**0.0046**	0.0690	**0.0041**	0.4166	0.6796	**0.0130**	0.0872
CIRA-CIC-DoHBrw	NetShare	0.3955	0.6117	0.5446	0.4630	0.6566	0.3886	0.6315	0.8825	0.7754	0.9776
	TrafficLLM	0.0162	**0.3782**	**0.4858**	**0.0001**	0.0614	0.0838	**0.4676**	**0.5258**	**0.0003**	0.1069
	Lens (Ours)	**0.0041**	0.4105	0.6915	**0.0001**	**0.0481**	**0.0246**	0.4896	0.7065	**0.0003**	**0.0728**
CIC-IoT-2023	NetShare	0.0732	0.0804	0.5939	0.1188	0.6807	0.2155	0.2273	0.8920	0.2936	0.9904
	TrafficLLM	0.0598	0.0345	0.5544	0.0586	**0.0039**	0.1523	0.1072	0.5844	0.0833	0.0061
	Lens (Ours)	**0.0146**	**0.0098**	**0.0179**	**0.0262**	**0.0039**	**0.0779**	**0.0325**	**0.0217**	**0.0295**	**0.0058**

3. Lens generates more aligned source IP with lower TVD on all datasets
   - Benefits from the auxiliary context pretraining that correlates dataset sources with network traffic
4. TrafficLLM performs slightly better for source port and destination IP generation
   - However, even TrafficLLM has way more parameters, Lens excels over TrafficLLM in most packet length generation tasks.

04 Experiments

4.3 Lens's traffic generation capability

Table 6 is quantitative because it is a "distribution distance"…
So, is it practically helpful? → let's check this out with fuzzing tests!

Datasets	Method	JSD ↓					TVD ↓				
		Src IP	Dst IP	Src Port	Dst Port	Len	Src IP	Dst IP	Src Port	Dst Port	Len
ISCX-VPN	NetShare	0.3591	0.3787	0.6539	0.5893	0.6793	0.5802	0.5948	0.9632	0.9137	0.9893
	TrafficLLM	**0.0946**	0.1175	0.5742	0.0430	0.0513	0.1849	0.1772	0.5920	0.0600	0.0845
	Lens (Ours)	0.0974	**0.0905**	**0.5574**	**0.0271**	**0.0338**	**0.1719**	**0.1245**	**0.5789**	**0.0343**	**0.0469**
ISCXTor	NetShare	0.3084	0.4160	0.5835	0.5736	0.6531	0.4930	0.6436	0.8813	0.8807	0.9756
	TrafficLLM	0.0023	**0.3629**	**0.5635**	0.1770	**0.0359**	0.0047	**0.4519**	**0.5838**	0.2339	**0.0500**
	Lens (Ours)	**0.0022**	0.4842	0.5826	**0.1337**	0.0398	**0.0038**	0.5620	0.6133	**0.1877**	0.0560
USTC-TFC	NetShare	0.4415	0.5065	0.5731	0.5885	0.6826	0.6876	0.7794	0.9037	0.9114	0.9933
	TrafficLLM	**0.3702**	**0.4186**	**0.3496**	0.2746	0.0159	0.3915	**0.4731**	**0.3656**	0.3261	0.0259
	Lens (Ours)	0.3783	0.4361	0.3864	**0.2685**	**0.0143**	**0.3910**	0.4748	0.4076	**0.2901**	**0.0203**
Cross Platform (IOS)	NetShare	0.3304	0.5267	0.6056	0.6104	0.6120	0.3389	0.5188	0.9289	0.9318	0.9491
	TrafficLLM	**0.0003**	**0.2433**	**0.5784**	0.0134	**0.0521**	**0.0006**	**0.3374**	**0.6006**	0.0265	**0.0662**
	Lens (Ours)	**0.0003**	0.3241	0.6508	**0.0083**	0.0608	**0.0006**	0.4523	0.6746	**0.0132**	0.0780
Cross Platform (AN)	NetShare	0.3459	0.3964	0.6219	0.6269	0.6312	0.5437	0.6205	0.9353	0.9443	0.9598
	TrafficLLM	0.0016	**0.1969**	**0.6011**	0.0082	**0.0672**	0.0216	**0.3101**	**0.6220**	0.0377	**0.0835**
	Lens (Ours)	**0.0003**	0.2809	0.6531	**0.0046**	0.0690	**0.0041**	0.4166	0.6796	**0.0130**	0.0872
CIRA-CIC-DoHBrw	NetShare	0.3955	0.6117	0.5446	0.4630	0.6566	0.3886	0.6315	0.8825	0.7754	0.9776
	TrafficLLM	0.0162	**0.3782**	**0.4858**	**0.0001**	0.0614	0.0838	**0.4676**	**0.5258**	**0.0003**	0.1069
	Lens (Ours)	**0.0041**	0.4105	0.6915	**0.0001**	**0.0481**	**0.0246**	0.4896	0.7065	**0.0003**	**0.0728**
CIC-IoT-2023	NetShare	0.0732	0.0804	0.5939	0.1188	0.6807	0.2155	0.2273	0.8920	0.2936	0.9904
	TrafficLLM	0.0598	0.0345	0.5544	0.0586	**0.0039**	0.1523	0.1072	0.5844	0.0833	0.0061
	Lens (Ours)	**0.0146**	**0.0098**	**0.0179**	**0.0262**	**0.0039**	**0.0779**	**0.0325**	**0.0217**	**0.0295**	**0.0058**

5. Pretraining-based Lens and TrafficLLM perform better than GAN-based Netshare on all tasks
   - Because pretraining-based models learn better packet-level network representation, while Netshare sacrifices it for global distribution similarity.

04 Experiments

4.3 Lens's traffic generation capability – Simulation results in fuzzing test

Q. Why only header? → As the payload is usually encrypted, users can either replay saved network payload or generate them randomly with an encrypted algorithm like AES for a more practical use.

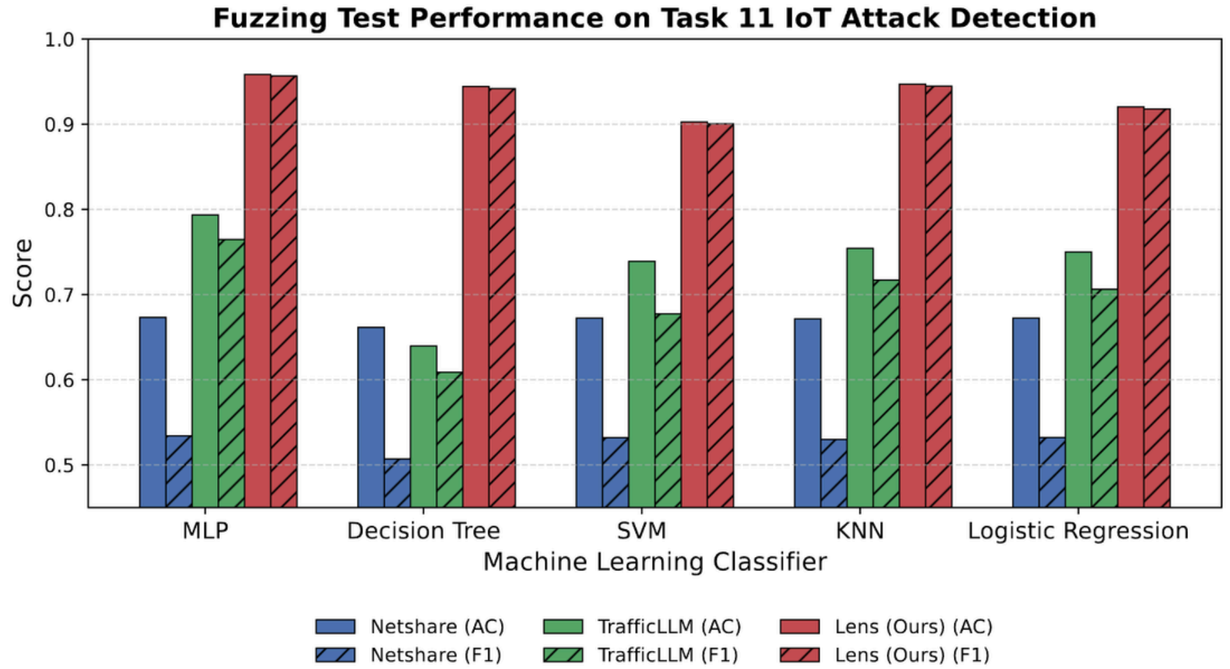


Figure 3: Fuzzing performance on IoT attack detection. Machine-learning models trained on Lens-generated traffic achieve consistently higher accuracy and F1 than those trained on baselines' generated traffic.

1. Resume Lens from its finetuned checkpoint and generate network header fields using the Task 11 training data.
2. Machine learning models are trained on the generated network header fields as a binary classifier to detect malicious network traffic.
3. The trained machine learning models are evaluated on the real Task 11 test set for malicious traffic detection, thereby assessing the quality of the synthetic network data.

- Lens generates high-fidelity network traffic that aligns better with real scenarios and achieves substantially better performance than baselines on all machine learning-based detectors.

04 Experiments

4.4 Ablation studies

<Traffic Classification>

Table 8: Ablation studies on Tor service detection and VPN application classification. Both KG-MSP and FT-Context contribute significantly to performance improvement.

Settings	Tor Service Detection		VPN App. Classification	
	AC	F1	AC	F1
`Lens (Full model)`	**0.9692**	**0.8120**	**0.8406**	**0.8137**
`w/o FT-Context`	0.9577	0.7700	0.7881	0.8111
`w/o KG-MSP`	0.9612	0.7621	0.7246	0.7622

<Traffic Generation>

Table 7: Ablation studies on VPN Destination Port generation. The pretraining KG-MSP and finetuning context improves both JSD and TVD.

Settings	JSD ↓	TVD ↓
`Lens (Full model)`	**0.0271**	**0.0343**
`w/o FT-Context`	0.0294	0.0384
`w/o KG-MSP`	0.0308	0.0437

KG-MSP pretraining and context-aware finetuning contribute to classification and generation.

- KG-MSP strengthens representation learning by masking key metadata and payload information, while FT-Context reduces the modality gap during finetuning.

5 Discussion and Conclusion

In this paper, we proposed Lens, a unified knowledge-guided foundation model for network traffic excelling in both network traffic classification and generation. Through pretraining with Knowledge-guided Mask Span Prediction with textual context, Lens effectively learns generalizable network representations from large-scale unlabeled traffic data. To effectively extend classification to new classes, we reframe traffic classification as a closed-ended generation task and handle the distribution shifts of new-class data through context-aware finetuning. Finally, we evaluated the performance of Lens on 6 real-world datasets, including 12 traffic classification tasks and 5 network generation tasks. For traffic classification, extensive experimental results demonstrated that Lens outperforms the baselines with a large margin in most tasks, showcasing its strong effectiveness and extensibility. For traffic generation, Lens generates better high-fidelity network traffic for effective network simulations, outperforming baselines substantially in fuzzing tests. Furthermore, ablation studies validate the effectiveness of KG-MSP and context-aware finetuning. In the future, we plan to implement the proposed method in real-world computer systems to evaluate its performance.

The End.

06 Appendix

C.1 Performance of per-class in the extensibility of classification

We summarize the setup of the extensibility experiments. For Task 6, the new classes are selected based on their higher sample counts: com.ifeng.news2 (226) for the 1-class setting; additionally, sohu.sohuvideo (200) and xunlei.downloadprovider (190) for the 3-class setting; and further qiyi.video (150) and youku.phone (150) for the 5-class setting. For Task 8, we follow the same criterion: pocket-pool (416) for the 1-class setting; plus aiqiyi (238) and color-ballz (224) for the 3-class setting; and further yy (206) and youku (182) for the 5-class setting. These labels provide clearer performance differences due to their larger test sets.

Across Table 14, 15, and 16, Lenssconsistently outperforms all baselines on both tasks. ET-BERT fine-tunes new classes with a learning rate of 2e-5, whereas Lensuses 5e-6 due to architectural differences, illustrating Lens's stronger extensibility and reduced need for task-specific adjustment.

Table 14: Performance of 1 new class in Task 6 and Task 8.

(a) Task 6 — com.ifeng.news2

Methods	P	R	F1
ET-BERT	0.0474	**1.0000**	0.0900
Lens	**0.9636**	0.9408	**0.9521**

(b) Task 8 — pocket-pool

Methods	P	R	F1
ET-BERT	0.0643	**1.0000**	0.1208
Lens	**0.6753**	**1.0000**	**0.8062**

Table 15: Performance of 3 new classes in Task 6 and Task 8.

Task 6

Methods	com.ifeng.news2			sohu.sohuvideo			xunlei.downloadprovider		
	P	R	F1	P	R	F1	P	R	F1
ET-BERT	0.0484	**0.9941**	0.0924	0.2213	**0.9858**	0.3615	0.5930	0.9440	0.7284
Lens	**0.8195**	**0.9941**	**0.8984**	**0.9577**	0.9645	**0.9611**	**0.9191**	**1.0000**	**0.9579**

Task 8

Methods	pocket-pool			aiqiyi			color-ballz		
	P	R	F1	P	R	F1	P	R	F1
ET-BERT	0.1098	**1.0000**	0.1978	0.0995	0.9888	0.1809	0.5685	0.9881	0.7217
Lens	**0.8041**	**1.0000**	**0.8914**	**0.4120**	**1.0000**	**0.5836**	**0.7000**	**1.0000**	**0.8235**

Table 16: Performance of 5 new classes in Task 6 and Task 8.

Task 6

Methods	com.ifeng.news2			sohu.sohuvideo			xunlei.downloadprovider			qiyi.video			youku.phone		
	P	R	F1	P	R	F1	P	R	F1	P	R	F1	P	R	F1
ET-BERT	0.0572	**1.0000**	0.1083	0.2945	**0.9858**	0.4535	0.4237	**1.0000**	0.5952	0.3839	**1.0000**	0.5548	0.5170	0.9681	0.6741
Lens	**0.7249**	0.9822	**0.8342**	**0.8528**	**0.9858**	**0.9145**	**0.8389**	**1.0000**	**0.9124**	**0.7532**	**1.0000**	**0.8592**	**0.6462**	0.8936	**0.7500**

Task 8

Methods	pocket-pool			aiqiyi			color-ballz			yy			youku		
	P	R	F1	P	R	F1	P	R	F1	P	R	F1	P	R	F1
ET-BERT	0.2188	**1.0000**	0.3590	0.2607	0.9551	0.4096	**0.6484**	0.9881	**0.7830**	0.4444	0.9870	0.6129	0.0000	0.0000	0.0000
Lens	**0.8254**	**1.0000**	**0.9043**	**0.5733**	**0.9663**	**0.7197**	0.6336	**0.9881**	0.7721	**0.5385**	**1.0000**	**0.7000**	**0.6476**	**1.0000**	**0.7861**