Fingerprinting AI Applications and Phases in Edge-Assisted Distributed Learning and Inference using Side-Channel Data

Lawrence Oyaniyi*, Anik Mallik*, Moinul Hossain†, and Wei Yu*
* Department of Computer & Information Sciences, Towson University, MD, USA
† Department of Cyber Security Engineering, George Mason University, VA, USA

***Abstract*—Edge computing has enabled users to experience ubiquitous artificial intelligence (AI) through distributed learning and inference. Continuous efforts to reduce computing burden from the edge devices and increased preservation of privacy have popularized remote inference and federated learning (FL). However, network-level side-channel information can still expose sensitive operational states. In this paper, we demonstrate that network-level telemetry data can be used to fingerprint edge-assisted learning and inference workflows and reveal their operational phases. We propose a hierarchical classification framework, where the first stage separates learning from inference, and the second distinguishes learning phases. In addition, we develop a testbed with convolutional and recurrent neural network-based FL and remote inference systems, alongside an attacker device collecting network sniffing data. Using features derived from flow volumes, transfer speeds, ratios, and latency, the system achieves fingerprinting accuracy of 100% between learning and inference tasks, and 95.9% across different learning phases. These results highlight the vulnerability of edge-assisted distributed AI systems to network-based side-channel fingerprinting.**

***Index Terms*—App fingerprinting, federated learning, edge computing, AI security, side-channel data.**

I. Introduction

Artificial intelligence (AI) has emerged as an essential component of modern computing, transforming various domains such as healthcare, finance, transportation, and smart infrastructure [1]. The growing scale of data generation and the need for real-time analytics prompt the migration of AI workloads from centralized clouds to distributed environments. Among these, edge-assisted distributed learning and inference are attracting significant interest due to their ability to improve privacy, reduce communication latency, and move computation closer to the data source. Federated learning (FL) represents one of the most exciting approaches to achieving distributed model training without direct data sharing [2]. In FL, clients use their private datasets to perform local updates and send only model parameters to the server that aggregates the results. This is complemented by distributed inference that allows devices to offload tasks to powerful edge servers [3].

However, the Quality-of-Service (QoS) of an edge-AI can be degraded when user privacy is violated or a proprietary model is compromised. For example, the neural network (NN) in an FL application is usually trained with sensitive data, and the NN model used in an inference application can be strictly proprietary, which must be protected from external attack [4]. In addition, the attack surface has grown dramatically since protecting user privacy became so critical. Adversaries now leverage techniques such as model extraction, data and model poisoning, evasion, or information leakage. These can compromise model integrity, confidentiality, and availability [3].

Recent studies show that adversaries can infer sensitive information from side channel and network telemetry data, even when communication is encrypted [5]–[8]. Side-channel data include observable signals such as packet timing, message size, throughput, latency, energy usage, and cache activity that unintentionally disclose information about the model's structure or operation [9]–[14]. Prior research has demonstrated that machine learning classifiers can identify the type of neural model or the task being performed by analyzing such signals [5]. Other researchers reported that microarchitectural behavior, power consumption, and electromagnetic emissions could also reveal the nature of computations in AI systems [11]–[14].

Despite these advances, existing fingerprinting techniques are primarily designed to identify AI architectures, but no research has captured the execution phases that characterize FL workflows. In practice, the FL process consists of multiple distinct stages: sending the global model to the client, fitting it locally, evaluating the updated model, and sending the new parameters to the server. Each phase produces a unique communication pattern that can serve as a fingerprint of the FL system.

Motivation: First, to ensure security and compliance in FL, it is essential to identify which *model* is active and its operational phases (*training or evaluation*), to detect attacks, verify history, and identify tampering. Second, prior work shows robust fingerprints from behavioral probes and side channels [9], [11], [13] for FL traffic motifs [5], but not phase identification. Third, no lightweight method provides distinctive, repeatable signatures that distinguish learning from evaluation. Thus, we aim to build and validate phase-aware fingerprints from network traffic data.

Assumptions: The following assumptions define the boundaries of the analysis: (i) the learning or inference process proceeds without detection of the observer, (ii) all payloads may be encrypted, but network metadata remains visible, (iii) the attacker has adequate visibility of the communication channel to collect traffic traces, and (iv) no other active attacks are going on.

Scope: This work examines whether side-channel telemetry collected from network sniffing can reveal the operational states of learning and inference workflows in edge-assisted FL environments. The analysis is restricted to passive observation of encrypted communication between Jetson-class devices orchestrated with the Flower framework, without interfering with traffic or accessing model parameters or raw data. The scope is limited to distinguishing between learning and inference, identifying the associated phases solely from network communication

This work was supported in part by the US National Science Foundation (NSF) under Grant No. 2451736. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the NSF.

behavior, assessing information leakage risks, and informing the need for defensive countermeasures.

Threat Model: This study defines a passive adversary positioned as an unauthorized entity within the network infrastructure capable of capturing network-level telemetry such as packet arrival time, size, and direction. Despite this privilege, the attacker is constrained by an inability to access raw data, local gradients, or encrypted payloads. Even with these restrictions, an attacker can perform phase-aware fingerprinting to distinguish between learning and inference tasks and reveal specific sub-phases (e.g., $\text{model}_{\text{down}}$ and model_{up}). Therefore, this action constitutes a direct violation of privacy by exposing sensitive workload patterns. Moreover, this metadata serves as a critical prelude to downstream exploits, including resource denial attacks, phase-specific timing attacks, and targeted jamming that compromise system availability and integrity.

Contributions: Our contributions in this study are summarized below: (i) This study proposes a complete framework for phase-aware fingerprinting of distributed learning and inference. (ii) We design supervised classifiers based on network traffic features that segment FL rounds and separate fit/train from evaluate, allowing phase-aware monitoring without access to raw data. (iii) We provide a unified logging schema and flower-based scripts that others can run on edge devices to reproduce phase-aware fingerprinting studies under realistic constraints.

II. Related Work

Hardware and wireless signal-level fingerprinting: FL is recognized as a central paradigm for balancing privacy and performance in fingerprinting tasks. In the radio-frequency domain, Hawk is proposed by [15], leveraging carrier frequency offset for long-range anomaly detection, achieving 97% accuracy and zero false positives. For localization, [16] proposed FedLoc3D, which decouples building-floor classification from coordinate regression for WiFi fingerprints. Likewise, [17] extended this with CSI-driven FL, combining CNN embeddings with FedProx to improve accuracy and reduce overhead.

Traffic and model-Level fingerprinting: At the traffic level, [18] proposes compressing flows into autoencoder-generated fingerprints before FL classification, yielding strong spoofing detection. Additionally, fingerprinting is studied at the model level. Researchers [9] utilize universal adversarial perturbations (UAPs) as global fingerprints for IP protection, distinguishing between pirated and homologous models. Network traffic metadata are also used to fingerprint deep learning architectures [5].

Anonymity network fingerprinting: Anonymity networks is studied in fingerprinting. Segmented Adversarial Defense (SAD) is proposed [4], which injects per-segment adversarial perturbations into Tor traffic. This highlights the growing interplay between fingerprinting attacks and adversarial defenses.

Novelty of our work: Collectively, previous studies examine RF signals, traffic features, WiFi CSI, adversarial subspaces, and encrypted traffic. While most of these works evaluate accuracy in controlled testbeds, our approach focuses exclusively on network trace fingerprints, capturing flow-level and temporal traffic patterns as a new modality for model attribution, anomaly detection, and resilience in FL environments.

III. Proposed Fingerprinting System

The proposed framework identifies the operational behavior of FL and inference applications by leveraging passive network telemetry. As illustrated in Fig. 1, our approach comprises five components designed to transform raw side-channel data into granular phase-recognition insights.

A. Network Infrastructure

The network infrastructure consists of edge devices connected to the edge server via a wireless access point (e.g., Wi-Fi and cellular). An adversary is assumed to have access to the same local network, with or without valid authentication.

B. Data Collection

During each learning and inference round, the system collects two distinct datasets. The first trains the fingerprinting model, while the second is used for evaluation. The passive collection module captures heterogeneous, multi-modal packet-level telemetry from the communication links between clients and the server to characterize both computation and communication behavior. This telemetry includes packet exchange counts, throughput statistics, transmitted and received byte volumes, and transmission delays. This process does not interfere with learning or modify any model parameters. All metrics are logged into CSV files on both server and client devices to ensure reproducibility.

C. Feature Engineering

We transform raw telemetry collected into feature vectors that characterize the behaviour of each operational phase of distributed learning and inference through the following steps:

1) Raw Data: Network sniffing captures low-level communication traces, including packet timestamps, sizes, directions, and protocol metadata.

2) Data Preprocessing: The raw telemetry is cleaned by removing irrelevant entries and corrupted records, followed by synchronization of timestamps across client-server links. Each telemetry sample is assigned to the corresponding learning or inference phase based on communication events observed in the federated learning pipeline. To create structured behavioral snapshots of the workflow, these traces are windowed into fixed-duration segments where communication activity is summarized for classification.

3) Feature Construction: Handcrafted behavior descriptors are formed by combining basic metrics into more discriminative ratios and directional measures, such as uplink–downlink traffic ratios, throughput imbalance, and speed difference. These constructed features amplify contrasts among learning phases. Additionally, we generate temporal features by computing the duration of each learning segment from timestamped telemetry, which characterizes both the internal model operations and the overall network responsiveness.

4) Feature Normalization: The extracted telemetry features exhibit different numerical scales and units. Feeding these raw values into the fingerprinting classifier would allow large-magnitude features to dominate decision process, while smaller-magnitude but equally informative features would have reduced influence. Feature normalization is applied to ensure that selected features contribute fairly in learning and prediction.

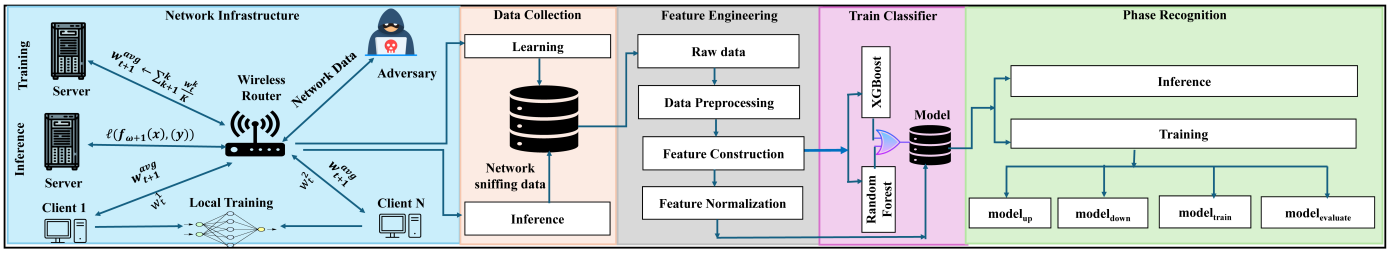


Fig. 1. Overview of the proposed fingerprinting system design.

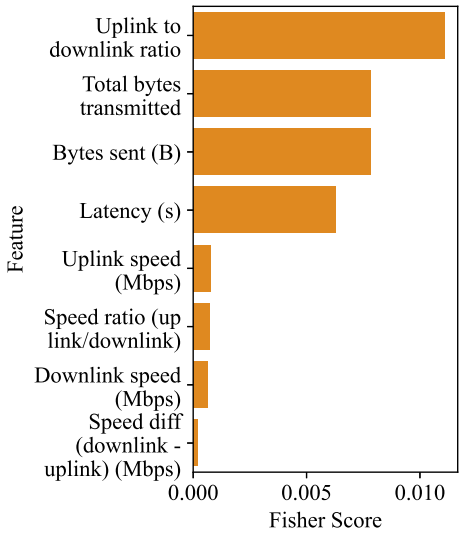


Fig. 2. Top Fisher scores for training versus inference.

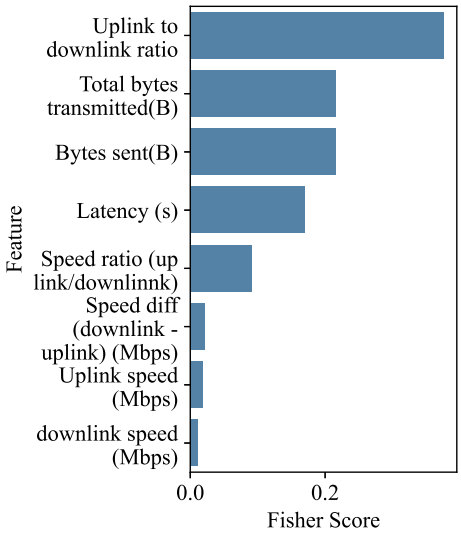


Fig. 3. Top Fisher scores for the training phases.

D. Train Classifier

The system employs a hierarchical classification framework in which models are trained on telemetry features ranked by Fisher scores, ensuring that only the most discriminative attributes are retained. Using stratified sampling, the data are split into training and test sets, and both XGBoost (XGB) and Random Forest (RF) classifiers are evaluated, with XGB selected when performance is comparable. The first classifier distinguishes whether a trace corresponds to learning or inference. In addition, a second classifier is engaged when learning is detected to determine the specific phase of the learning process (i.e., $\text{model}_{\text{down}}$, $\text{model}_{\text{train}}$, model_{up}, and $\text{model}_{\text{evaluate}}$). The overall workflow for this process is detailed in Algorithm 1.

E. Phase Recognition

During deployment, the normalized network-sniffing data are fed into the hierarchical model, enabling the system to recognize not only whether the device is learning or inferring but also the fine-grained structure of learning activity as reflected in its telemetry. The coarse prediction identifies the global state, and if required, the fine-grain classifier refines the decision. This approach supports accurate recognition of FL behavior using encrypted side-channel communication alone. From a security perspective, this capability demonstrates that an adversary with access to side-channel information can move beyond simple detection of learning activity to infer detailed phase-level behavior. This exposure reveals sensitive workload patterns and opens avenues for model extraction and phase-specific timing attacks, which undermine the confidentiality and integrity of distributed learning and inference systems.

IV. Experimental Setup

The experiment is conducted in a distributed learning environment, with one NVIDIA Jetson AGX Orin configured as the central server and two Jetson Orin Nano devices assigned as clients. The training and evaluation processes are coordinated through the Flower framework. This setup provides a robust and scalable architecture that can seamlessly accommodate additional edge clients. By training models locally and aggregating them at the central server, this configuration emulates a realistic edge learning environment.

To ensure broad workload coverage and prevent hardware-specific overfitting, three distinct neural architectures are employed. A convolutional neural network based on ResNet-18 is trained on the CIFAR-10 dataset for image classification. A recurrent neural network with a long short-term memory architecture is trained on the UCI Human Activity Recognition dataset to demonstrate sequence learning. For inference, the YOLO model trained on the COCO dataset is used. While the testbed utilizes Jetson-class devices, the variety in model complexity (CNN vs. RNN) and the use of the Flower framework allow for a realistic simulation of edge-assisted AI ecosystems.

During both learning and inference stages, telemetry data are recorded in real time. These measurements are processed to generate fingerprints that describe each model's operational characteristics. The generated fingerprints are analyzed to determine their effectiveness in identifying the inference and

Algorithm 1: End-to-End Training and Hierarchical Prediction

Input: $D \in \mathbb{R}^{n \times m}$ (telemetry samples), label set L
Output: Selected coarse model M_1^*, selected phase model M_2^*, and predictions $\hat{Y}$
Step 1: Feature Selection
Compute Fisher score F_j for each feature f_j and sort in descending order.
Select indices of the top k features: $S_k = \arg\text{top k}_j\, F_j$.
Form the reduced dataset $D' = D[:, S_k]$.
Step 2: Data Preparation
Encode labels L. Split D' into D_{train} and D_{test}.
Step 3: Train Coarse Models (Learning vs. Inference)
Train XGB_1 on $(D_{\text{train}}, L_{\text{coarse}})$ and compute $\text{Acc}(\text{XGB}_1)$ on D_{test}.
Train RF_1 on $(D_{\text{train}}, L_{\text{coarse}})$ and compute $\text{Acc}(\text{RF}_1)$ on D_{test}.

$$M_1^* = \begin{cases} \text{XGB}_1, & \text{if Acc}(\text{XGB}_1) \geq \text{Acc}(\text{RF}_1), \\ \text{RF}_1, & \text{otherwise.} \end{cases}$$

Step 4: Train Phase Models (Learning Subphases)
Train XGB_2 on $(D_{\text{train}}, L_{\text{phase}})$ and compute $\text{Acc}(\text{XGB}_2)$ on D_{test}.
Train RF_2 on $(D_{\text{train}}, L_{\text{phase}})$ and compute $\text{Acc}(\text{RF}_2)$ on D_{test}.

$$M_2^* = \begin{cases} \text{XGB}_2, & \text{if Acc}(\text{XGB}_2) \geq \text{Acc}(\text{RF}_2), \\ \text{RF}_2, & \text{otherwise.} \end{cases}$$

Step 5: Hierarchical Prediction
foreach *new sample* x **do**
  $\hat{y}_1 \leftarrow M_1^*(x)$
  if $\hat{y}_1 =$ *Learning* **then**
    $\hat{y} \leftarrow M_2^*(x)$; `// modelup, modeldown, modeltrain, or modelevaluate`
  else
    $\hat{y} \leftarrow$ Inference
  Append $\hat{y}$ to $\hat{Y}$
Step 6: Reporting
If ground truth Y exists, compute Accuracy, Precision, Recall, and F_1.
return $M_1^*,\ M_2^*,\ \hat{Y}$

learning phases based on the observed behavioral patterns. The testbed setup and devices utilized are presented in Fig. 4.

V. Performance Evaluation

The fingerprinting framework is evaluated using network sniffing data as inputs for prediction, with ground-truth phase labels derived from learning and inference data. The evaluation follows a hierarchical design in which the first classifier distinguishes between learning and inference, and upon detecting learning, a second classifier identifies the phases: $\text{model}_{\text{down}}$, $\text{model}_{\text{train}}$, model_{up}, and $\text{model}_{\text{evaluate}}$. Performance is quantified using accuracy, precision, recall, and f1-score, with confusion matrices illustrating correct and misclassified results.

Classification performance is assessed based on accuracy, precision, recall, and F1-score. Accuracy indicates overall correctness, precision reflects the reliability of predictions, and recall measures sensitivity to actual instances. F1-score balances precision and recall. Confusion matrices further illustrate the distribution of predictions and highlight misclassifications.

The classification results in Table I demonstrate that network-level telemetry contains highly discriminative fingerprints for identifying the operational states of learning and inference in FL. The learning and inference phases are fingerprinted with

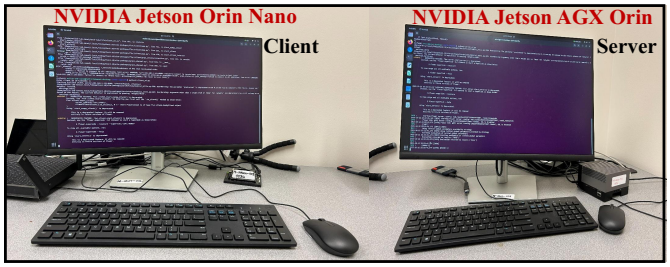


Fig. 4. Experimental setup used to evaluate the proposed fingerprinting framework, consisting of Nvidia Jetson AGX Orin and Orin Nano.

TABLE I
Top-level classification results: Learning vs. Inference

Class	Precision	Recall	F1
Inference	1.00±0.00	1.00 ±0.00	1.00±0.00
Learning	1.00±0.00	1.00 ±0.00	1.00±0.00
Accuracy		100% ±0.00	

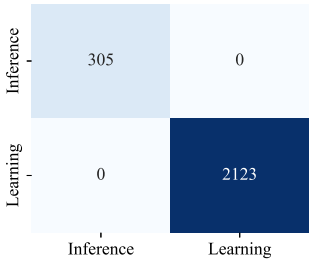
100% accuracy, indicating that their communication patterns are completely distinct. This outcome highlights the reliability of network traces for phase detection without relying on system-level measurements. Achieving 100% accuracy at the top-level is a reflection of the model's ability to distinguish between learning and inference rather than an artifact of a control laboratory setup. Their behaviors on the network are naturally distinct, which makes them easier to separate.

As shown in Table II, the classifier achieves an overall accuracy of 95.9% across learning phases. $\text{Model}_{\text{train}}$ and model_{up} are detected almost perfectly (F1 $=\sim$ 1.00), reflecting their distinctive traffic patterns, while $\text{model}_{\text{down}}$ and $\text{model}_{\text{evaluate}}$ achieve F1 scores of 0.92. $\text{Model}_{\text{down}}$ shows higher precision (0.96) but lower recall (0.89), while $\text{model}_{\text{evaluate}}$ shows the reverse, indicating occasional confusion between the two phases. These results confirm that network-level telemetry provides sufficient detail to distinguish between different phases of the learning process. At the same time, the occasional lower performance (95%) observed in sub-phase analysis, especially between model $_{\text{down}}$ and model $_{\text{evaluate}}$, is a reflection of the fact that the model is not blindly fitting to one environment. Those phases genuinely share some behavioral similarities, and the system responds to those overlaps. In essence, it captures genuine patterns rather than memorizing a configuration.

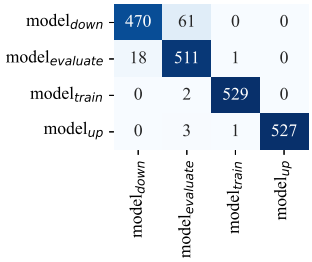
TABLE II
Training phase classification results

phase	Precision	Recall	F1
$\text{model}_{\text{down}}$	0.96±0.02	0.89±0.03	0.92±0.02
$\text{model}_{\text{evaluate}}$	0.89±0.04	0.96±0.01	0.92±0.03
$\text{model}_{\text{train}}$	1.00±0.00	1.00±0.00	1.00±0.00
model_{up}	1.00±0.00	0.99±0.01	1.00±0.00
Accuracy		95.9%±0.60	

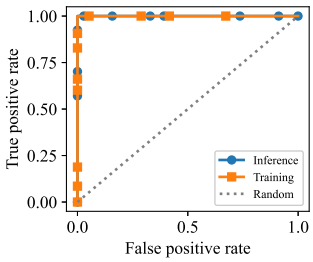
Figs. 5(a)–5(e) provide further evidence of this result. The confusion matrix in Fig. 5(a) shows that all inference and learning samples are correctly classified with no overlap. The



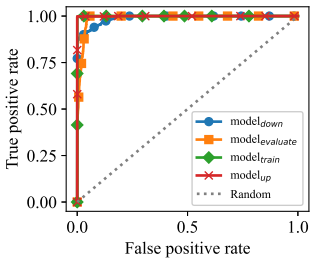
(a) Confusion matrix (training vs. inference).



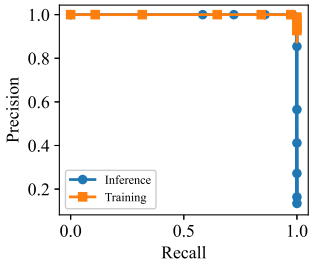
(b) Confusion matrix (learning phase).



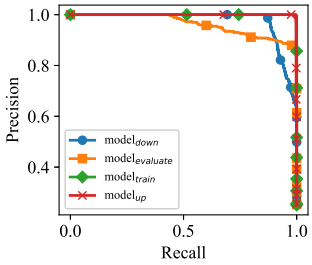
(c) ROC curve (training vs. inference).



(d) ROC curve (training sub-phase).



(e) Precision-Recall curve (training vs inference).



(f) Precision–Recall curve (training sub-phase).

Fig. 5. Comparison of training vs. inference (top row) and training sub-phases (bottom row) results: confusion matrix, ROC, and precision–recall curves.

receiver operating characteristic (ROC) curve in Fig. 5(c) and the precision-recall curve in Fig. 5(e) both reach the ideal area under the curve, confirming that the classifier achieves perfect fingerprinting for learning and inference.

A deeper investigation of the learning phase shows that the classifier achieves an overall accuracy of 95.9% across the $\text{model}_{\text{down}}$, $\text{model}_{\text{train}}$, $\text{model}_{\text{evaluate}}$, and model_{up}. As summarized in Table II, $\text{model}_{\text{train}}$ and model_{up} are detected with near-perfect accuracy, while $\text{model}_{\text{down}}$ and $\text{model}_{\text{evaluate}}$ exhibit slightly lower precision and recall values. This is because their network behaviors overlap with those of other phases and are less distinctive. The occasional lower performance (95.9%) observed in sub-phase analysis, especially between model $_{\text{down}}$ and model $_{\text{evaluate}}$, is a reflection of the fact that the model is not blindly fitting to one environment. Those phases genuinely share some behavioral similarities, and the system responds to those overlaps. In other words, it is reacting to real patterns in the data rather than just memorizing a specific configuration. This observed behaviour shows that when there is a clear distinguishable patterns, the framework separate them flawlessly and when they overlap, it makes small and predictable mistakes.

These findings are supported by Figs. 5(b)–5(f). The confusion matrix in Fig. 5(b) reveals that most errors occur between $\text{model}_{\text{down}}$ and $\text{model}_{\text{evaluate}}$, while $\text{model}_{\text{train}}$ and model_{up} are almost always recognized correctly. The ROC curves in Fig. 5(d) remain close to the ideal upper-left boundary for all four classes, and the precision-recall curves in Fig. 5(f) show consistently high precision across different recall levels.

TABLE III
RANDOM FOREST AND XGBOOST IN TRAINING AND INFERENCE

Metric	Random Forest	XGBoost
Accuracy	0.9991 ± 0.0003	**0.9996 ± 0.0002**
Macro F1	0.9989 ± 0.0005	**0.9994 ± 0.0004**
Training latency (s)	**1.15 ± 0.05**	3.97 ± 0.10
Fingerprint latency (ms)	0.039 ± 0.001	**0.003 ± 0.0001**
Model Size (MB)	**0.68 ± 0.01**	0.70 ± 0.01
Training Memory (MB)	**173 ± 2**	**173 ± 2**

In Table III, a comparison between Random Forest (RF) and XGBoost (XGB) is presented for both learning and inference phases. XGB achieves higher accuracy (0.9996 vs. 0.9991) and macro F1-score (0.9994 vs. 0.9989), along with significantly faster fingerprinting latency (0.003 ms/sample vs. 0.039 ms/sample). On the other hand, RF completes training more quickly (1.15 s vs. 3.97 s) and shows similar memory consumption. Additionally, XGB offers better predictive performance and reduced fingerprinting latency, making it more suitable for deployment, though RF is faster in training [19].

TABLE IV
RANDOM FOREST AND XGBOOST ON LEARNING PHASE

Metric	Random Forest	XGBoost
Accuracy	0.937 ± 0.006	**0.984 ± 0.004**
Macro F1 score	0.936 ± 0.007	**0.984 ± 0.003**
Training latency (s)	**1.54 ± 0.05**	45.8 ± 0.9
Fingerprint latency (ms)	0.052 ± 0.002	**0.019 ± 0.001**
Model size (MB)	11.2 ± 0.3	**3.8 ± 0.2**
Training memory (MB)	**180 ± 2.5**	199 ± 3.1

In Table IV, XGB consistently surpasses RF throughout all stages. While RF achieves an overall accuracy of 0.937, XGB enhances this to 0.984, attaining higher F1-scores in every phase. Although XGB requires a longer training duration (45.8 s compared to 1.54 s) than RF, it results in a smaller model size (3.8 MB versus 11.2 MB) and provides faster fingerprinting (0.019 ms/sample against 0.052 ms/sample). These findings support choosing XGBoost as more favorable, as it offers superior accuracy, faster inference, and reduced model size, even though it has a higher training cost [19].

TABLE V
LEARNING AND INFERENCE FINGERPRINTING PERFORMANCE

AI Application	TP	FP
Inference	694 ± 8	0 ± 0
Synchronous Learning	2123 ± 45	1 ± 0
Asynchronous Learning	3676 ± 58	1 ± 0
Accuracy (%)	**97.8 ± 0.3**	**98.3 ± 0.2**

Table V shows the comparison of our fingerprinting performance under various network conditions. The model accurately distinguishes inference and training operations with minimal

degradation under a delay scenario. This phenomenon results from a temporal spacing between client inferences that reduces overlapping resource contention. Overall, the inference pipeline demonstrates strong consistency and robustness, maintaining accuracy above 96% even in a non-ideal, delayed FL system.

TABLE VI
COMPARISON OF LEARNING-PHASE FINGERPRINTING PERFORMANCE UNDER DIFFERENT FL SYSTEMS

Federated learning phases	**Synchronous FL**		**Asynchronous FL**	
	TP	**FP**	**TP**	**FP**
model_{down}	235 ± 8	6 ± 4	836 ± 41	836 ± 80
$\text{model}_{evaluate}$	239 ± 9	44 ± 37.5	905 ± 27	83 ± 37.5
model_{train}	207 ± 7	0 ± 0	885 ± 35	0 ± 0
model_{up}	244 ± 8	0 ± 0	919 ± 27	0 ± 0
Accuracy (%)	97.8 ± 0.3		97.3 ± 0.5	

Table VI compares the learning phase recognition performance of the proposed fingerprinting system under asynchronous and synchronous network conditions as explained above. Across all operational phases ($\text{model}_{\text{down}}$, $\text{model}_{\text{evaluate}}$, $\text{model}_{\text{train}}$, and model_{up}), the model maintains highly consistent predictions, with total true positives showing minimal deviation between synchronous and asynchronous federated learning and inference. The overall classification accuracy remains above 97% in all cases, confirming that the proposed fingerprinting is effective for both synchronous and asynchronous FL systems.

VI. DISCUSSION

The hierarchical classification results reveal that network traffic alone provides highly discriminative fingerprints of activities in edge-AI. The perfect separation of the inference and learning phases highlights the deterministic nature of their communication patterns, with each phase leaving distinct traces in the flow statistics. While fingerprinting in the training phase, the classifier achieves near-perfect recognition of $\text{model}_{\text{train}}$ and model_{up}, reflecting their characteristic traffic profiles. For instance, large outbound model updates dominate the model_{up} phase, while $\text{model}_{\text{train}}$ produces stable and consistent flows tied to computation-driven synchronization. Meanwhile, $\text{model}_{\text{down}}$ and $\text{model}_{\text{evaluate}}$ show minor misclassifications, which may result from overlaps in flow size and timing, since small model downloads can resemble evaluation exchanges. Nevertheless, the overall accuracy of 95.9% demonstrates that even fine-grained learning phases are reliably fingerprinted. In order to ensure the attack was not only effective in a clean, ideal lab setting, the framework was also tested under asynchronous configurations that mimic real-world FL challenges. Our proposed fingerprinting framework is scalable, and is applicable in scenarios with any number of clients and servers.

VII. CONCLUSION

In this paper, we presented a novel two-hierarchy fingerprinting framework for distributed learning and inference applications. The proposed framework shows how an adversary can access side-channel data that can expose sensitive information about learning and inference in a distributed edge-assisted AI application. Through an empirical study, we demonstrated that network-level telemetry alone provides sufficiently rich fingerprints to identify operational states in learning and inference. The hierarchical classification framework achieves an average of 100% accuracy in distinguishing between learning and inference and an average of 95.9% accuracy in distinguishing between FL phases. These findings confirm that communication patterns capture stable and discriminative signatures of learning and inference workflows. Based on fingerprinting classification, an attacker can successfully execute a customized attack on AI applications, compromising data integrity, privacy, and overall QoS. This work paves the path for future AI fingerprinting and targeted jamming attack research.

REFERENCES

[1] J. Mulo, H. Liang, M. Qian, M. Biswas, B. Rawal, Y. Guo, and W. Yu, "Navigating challenges and harnessing opportunities: Deep learning applications in Internet of Medical Things," *Future Internet*, vol. 17, 2025.

[2] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, "Federated learning: Challenges, methods, and future directions," *IEEE Signal Processing Magazine*, vol. 37, no. 3, pp. 50–60, May 2020.

[3] L. Zhao, J. Zhao, and X. Wang, "Machine learning in transportation: A survey," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 12, pp. 7661–7678, Dec. 2021.

[4] R. Tang, G. Shen, C. Guo, and Y. Cui, "SAD: Website fingerprinting defense based on adversarial examples," *Security and Communication Networks*, pp. 1–55, 2022.

[5] M. N. H. Shuvo and M. Hossain, "Fingerprinting deep learning models via network traffic patterns in federated learning," in *Proc. of ACM WiseML*, 2025, pp. 32–37.

[6] M. N. H. Shuvo, M. Hossain, A. Mallik, J. Twigg, and F. Dagefu, "FLARE: A wireless side-channel fingerprinting attack on federated Learning," in *Proc. IEEE INFOCOM*, 2026.

[7] V. Duddu, D. Samanta, D. V. Rao, and V. E. Balas, "Stealing neural networks via timing side channels," *arXiv preprint arXiv:1812.11720*, 2018.

[8] H. Chen, B. D. Rouhani, Z. J. Fu, C., and F. Koushanfar, "Deepmarks: A secure fingerprinting framework for digital rights management of deep learning models," in *Proc. of ACM ICMR*, 2019, pp. 105–113.

[9] Z. Peng, S. Li, G. Chen, C. Zhang, H. Zhu, and M. Xue, "Fingerprinting deep neural networks globally via universal adversarial perturbations," in *Proc. of the IEEE/CVF CVPR*, 2022, pp. 13 430–13 439.

[10] P. Patel, E. Choukse, C. Zhang, Í. Goiri, B. Warrier, N. Mahalingam, and R. Bianchini, "Proc. of acm asplos, volume 3," in *Characterizing power management opportunities for LLMs in the cloud*, 2024, pp. 207–222.

[11] W. Hua, Z. Zhang, and G. E. Suh, "Reverse engineering convolutional neural networks through side-channel information leaks," in *Proc. of ACM DAC*, 2018, pp. 1–6.

[12] H. T. Maia, C. Xiao, D. Li, E. Grinspun, and C. Zheng, "Can one hear the shape of a neural network?: Snooping the GPU via magnetic side channel," in *Proc. of USENIX Security Symposium*, 2022.

[13] P. Horváth, D. Lauret, Z. Liu, and L. Batina, "SoK: neural network extraction through physical side channels," in *Proc. of USENIX Conference on Security Symposium*. Usenix, 2024.

[14] B. A. D. Kumar, S. C. Teja R, S. Mittal, B. Panda, and C. K. Mohan, "Inferring DNN layer-types through a hardware performance counters based side channel attack," in *Proc. of on AI-ML Systems*, 2021, pp. 1–7.

[15] S. Halder and T. Newe, "Radio fingerprinting for anomaly detection using federated learning in LoRa-enabled Industrial Internet of Things," *Future generation computer systems*, vol. 143, pp. 322–336, 2023.

[16] B. Gao, F. Yang, N. Cui, K. Xiong, Y. Lu, and Y. Wang, "A federated learning framework for fingerprinting-based indoor localization in multi-building and multifloor environments," *IEEE Internet of Things Journal*, vol. 10, no. 3, pp. 2615–2629, 2022.

[17] N. Nagia, M. T. Rahman, and S. Valaee, "Federated learning for WiFi fingerprinting," in *Proc. of IEEE ICC*, 2022, pp. 4968–4973.

[18] H. Wang, D. Eklund, A. Oprea, and S. Raza, "FL4IoT: IoT device fingerprinting and identification using federated learning," *ACM Transactions on Internet of Things*, vol. 4, no. 3, pp. 1–24, 2023.

[19] V. S. Desdhanty and Z. Rustam, "Liver cancer classification using random forest and extreme gradient boosting (XGBoost) with genetic algorithm as feature selection," in *Proc. of IEE DASA*. IEEE, 2021, pp. 716–719.