MetaFTT: Meta-Learning-Based FT-Transformer for Few-Shot IoT Intrusion Detection

Seonghoon Jeong

System and Network Security Lab. (SNSec Lab.)
Division of Artificial Intelligence Engineering, Sookmyung Women's University

November 12, 2025

Paper under review for Journal of Network and Computer Applications (JNCA), Elsevier

Impact factor: 8.0 (Clarivate Analytics, 2025), top 4.3% journal in category COMPUTER SCIENCE, SOFTWARE ENGINEERING.

MetaFTT: Meta-Learning-Based FT-Transformer for Few-Shot IoT Intrusion Detection

Seonghoon Jeong[a], Sachin Kaja[b], Hyunjae Kang[b,*], Ulysses Lam[b], Jung Taek Seo[c] and Dan Dongseong Kim[b]

[a]Division of Artificial Intelligence Engineering, Sookmyung Women's University, 04310, Seoul, Republic of Korea
[b]School of Electrical Engineering and Computer Science (EECS), The University of Queensland, 4072, Brisbane, Australia
[c]Department of Smart Security, Gachon University, 13120, Seongnam, Republic of Korea

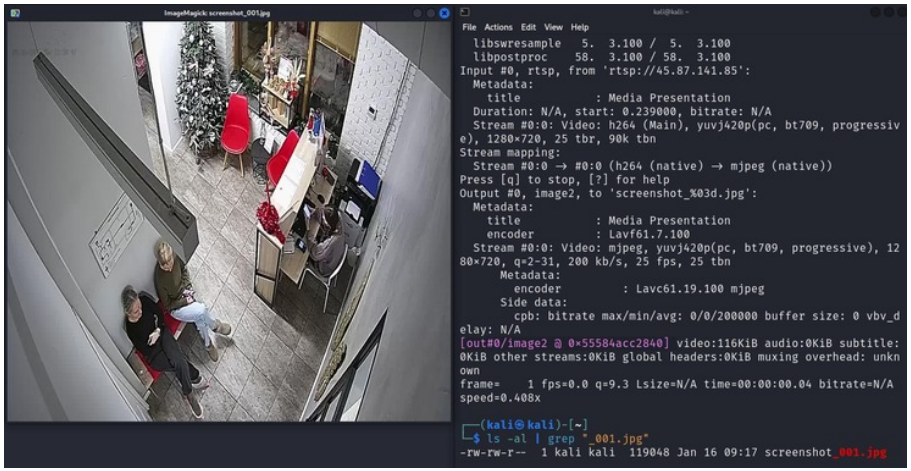
ARTICLE INFO

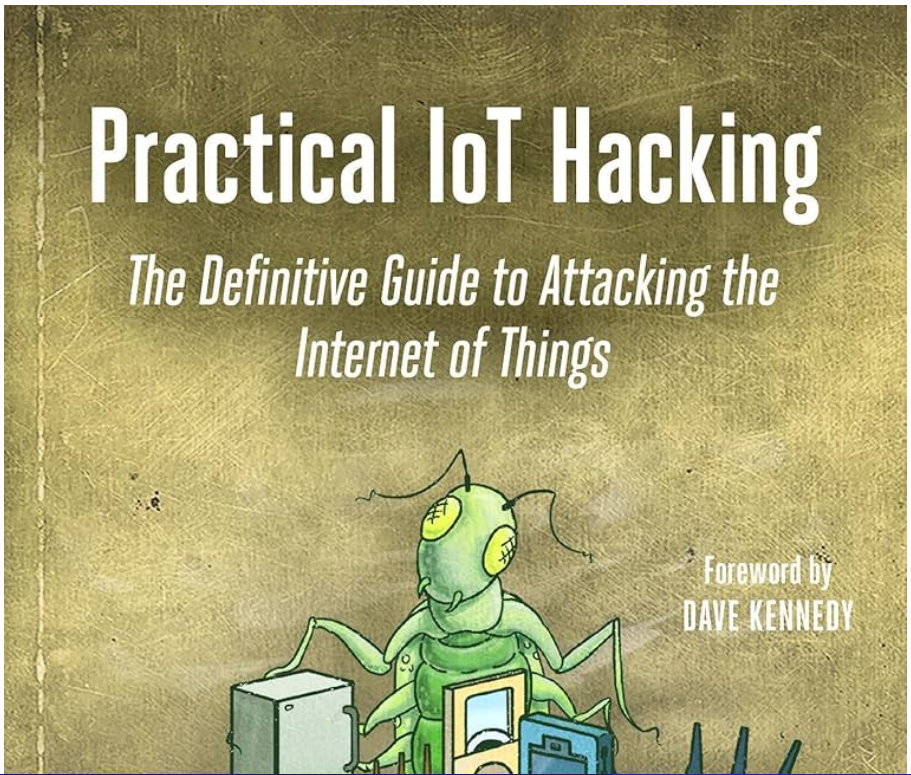
Keywords:
Network Intrusion Detection System (NIDS)
IoT Security
Few-Shot Learning (FSL)
Meta-Learning
Rapid Adaptation

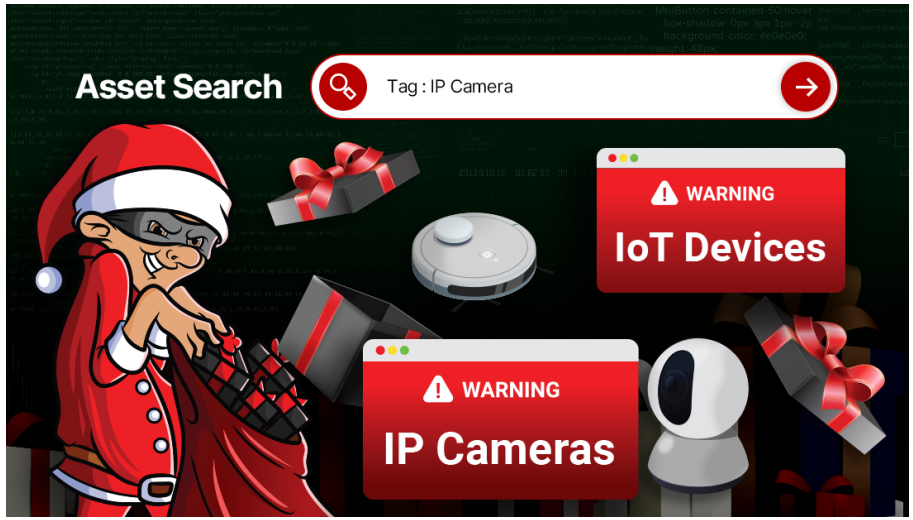
ABSTRACT

Network Intrusion Detection Systems (NIDS) are considered one of the essential components to secure Internet of Things (IoT) networks from complex, evolving, and previously unseen cyberthreats. Recently, deep neural network models act as the backbone of NIDSs. However, they remain several challenges—supervised models require gold (human-annotated) labels which are hard to obtain at scale; unsupervised models typically underperform and cannot handle complex traffic. Although the optimization-based meta-learning enables the rapid adaptation to new tasks (*i.e.,* novel threats), with minimal labeled data, and without extensive retraining, it is not widely adopted for network intrusion detection due to significant inner-loop cost. In this paper, we propose MetaFTT, a few-shot intrusion detection method that integrates an FT-Transformer backbone with episodic meta-learning for rapid adaptation from a small support set. The lightweight inner adaptation and first-order meta-learning reduces the inner-loop cost and allows MetaFTT to find initial parameters for the complex backbone model. We evaluate the zero-day detection performance of MetaFTT using the UQ-IoT-IDS-2021 dataset with a zero-day protocol that holds out one attack family at a time and deploys a binary detector (benign vs. held-out attack). Across nine zero-day IoT attack scenarios, the experimental

Introduction: The IoT Security Challenge







Introduction: The IoT Security Challenge

- The Internet of Things (IoT) has seen massive expansion, but this also broadens the attack surface.

Introduction: The IoT Security Challenge

- The Internet of Things (IoT) has seen massive expansion, but this also broadens the attack surface.
- Network Intrusion Detection Systems (NIDS) are vital for securing these networks.

Introduction: The IoT Security Challenge

- The Internet of Things (IoT) has seen massive expansion, but this also broadens the attack surface.
- Network Intrusion Detection Systems (NIDS) are vital for securing these networks.
- Traditional Deep Learning (DL) for NIDS faces challenges:
  - **Supervised models** require large, labeled datasets, which are scarce for new (zero-day) attacks.
  - **Unsupervised models** often suffer from higher false-positive rates on dynamic IoT traffic.

Introduction: The IoT Security Challenge

- The Internet of Things (IoT) has seen massive expansion, but this also broadens the attack surface.
- Network Intrusion Detection Systems (NIDS) are vital for securing these networks.
- Traditional Deep Learning (DL) for NIDS faces challenges:
  - **Supervised models** require large, labeled datasets, which are scarce for new (zero-day) attacks.
  - **Unsupervised models** often suffer from higher false-positive rates on dynamic IoT traffic.
- **The Core Problem:** How can we effectively detect novel, unseen "zero-day" attacks when labeled data is extremely limited?

Limitations of Traditional DL-based NIDS

Even with advances in DL, major challenges persist:

Supervised Learning

- **Pro:** High accuracy for *known* attacks.
- **Con:** Requires massive, manually-labeled datasets.
- **Con:** Labels for new "zero-day" attacks are scarce or non-existent.
- **Con:** Performance degrades under label noise and domain shift.

Both approaches struggle with the rapid emergence of novel attacks.

Limitations of Traditional DL-based NIDS

Even with advances in DL, major challenges persist:

Supervised Learning

- **Pro:** High accuracy for *known* attacks.
- **Con:** Requires massive, manually-labeled datasets.
- **Con:** Labels for new "zero-day" attacks are scarce or non-existent.
- **Con:** Performance degrades under label noise and domain shift.

Unsupervised Learning

- **Pro:** Can detect anomalies without any labels.
- **Con:** Often suffers from a high false-positive rate (FPR).
- **Con:** Struggles to distinguish benign traffic bursts from malicious anomalies.
- **Con:** Highly sensitive to data representation and thresholds.

Both approaches struggle with the rapid emergence of novel attacks.

Background: Few-Shot Learning (FSL) for NIDS

Goal: Build models that can generalize from a tiny number of labeled examples.

Background: Few-Shot Learning (FSL) for NIDS

Goal: Build models that can generalize from a tiny number of labeled examples.

Two main FSL approaches

1. Metric-Based
2. Optimization-Based

Background: Few-Shot Learning (FSL) for NIDS

Goal: Build models that can generalize from a tiny number of labeled examples.

Two main FSL approaches

1. Metric-Based
2. Optimization-Based

1. Metric-Based FSL

- Learns a deep embedding space.
- Classifies based on distance (e.g., Prototypical Nets).
- **Pro:** Computationally efficient.
- **Con:** Static metric space; less flexible for truly novel attacks.

Background: Few-Shot Learning (FSL) for NIDS

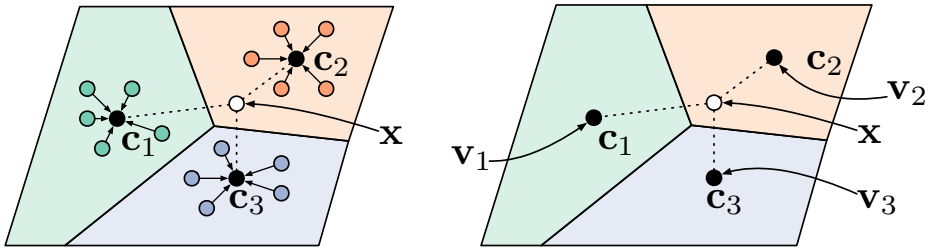
Goal: Build models that can generalize from a tiny number of labeled examples.

Two main FSL approaches

1. Metric-Based
2. Optimization-Based

1. Metric-Based FSL

- Learns a deep embedding space.
- Classifies based on distance (e.g., Prototypical Nets).
- **Pro:** Computationally efficient.
- **Con:** Static metric space; less flexible for truly novel attacks.



(a) Few-shot (b) Zero-shot

Background: Few-Shot Learning (FSL) for NIDS

Goal: Build models that can generalize from a tiny number of labeled examples.

Two main FSL approaches

1. Metric-Based
2. Optimization-Based

2. Optimization-Based FSL

- Learns a model *initialization* that adapts quickly (e.g., MAML).
- **Pro:** Explicit and powerful adaptation.
- **Con:** Extremely high computational and memory cost (second-order gradients).

Background: Few-Shot Learning (FSL) for NIDS

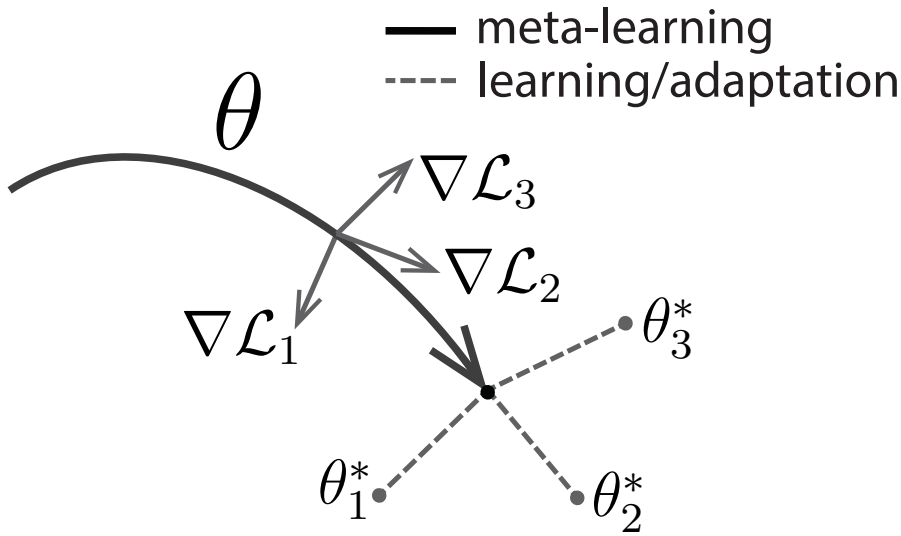
Goal: Build models that can generalize from a tiny number of labeled examples.

Two main FSL approaches

1. Metric-Based
2. Optimization-Based

2. Optimization-Based FSL

- Learns a model *initialization* that adapts quickly (e.g., MAML).
- **Pro:** Explicit and powerful adaptation.
- **Con:** Extremely high computational and memory cost (second-order gradients).



The Research Gap: Our Motivation

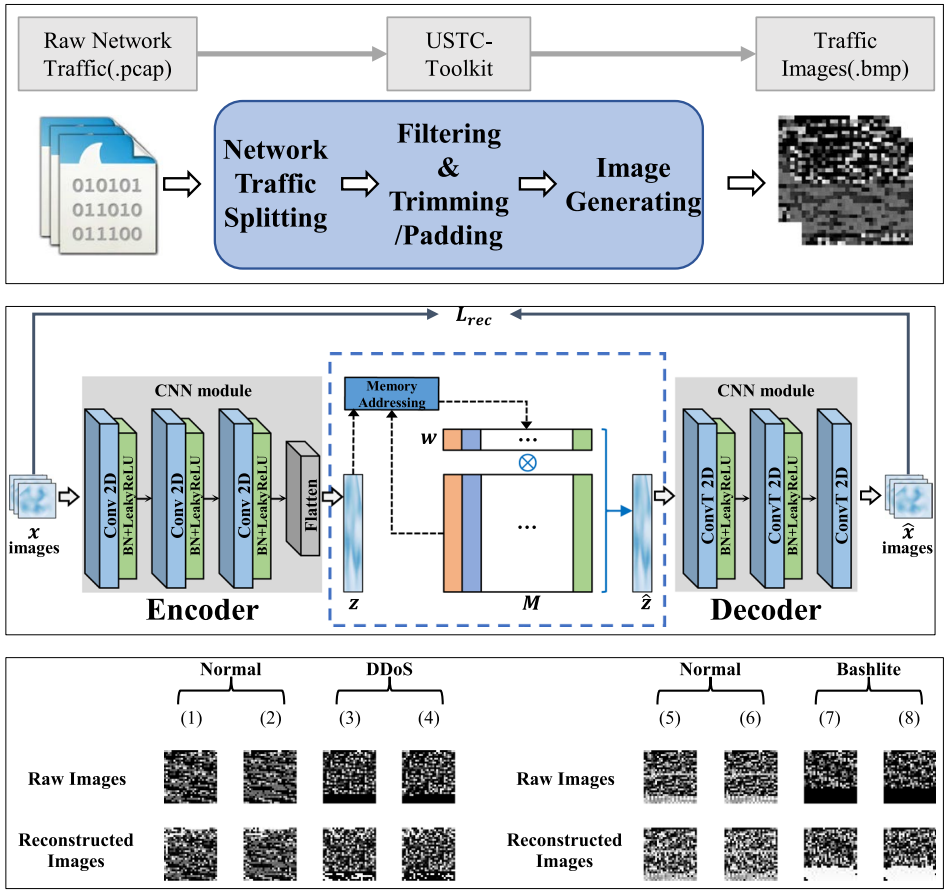
We identified three critical limitations in prior work:

1. **Architectural Mismatch:** Most FSL-NIDS use CNNs, forcing tabular network data into 2D images. This imposes an artificial structure and loses information.

The Research Gap: Our Motivation

We identified three critical limitations in prior work:

1. **Architectural Mismatch:** Most FSL-NIDS use CNNs, forcing tabular network data into 2D images. This imposes an artificial structure and loses information.



The Research Gap: Our Motivation

We identified three critical limitations in prior work:

1. **Architectural Mismatch:** Most FSL-NIDS use CNNs, forcing tabular network data into 2D images. This imposes an artificial structure and loses information.
2. **Adaptability vs. Cost:** A trade-off exists. Metric-based is efficient but weak. Optimization-based is strong but *too computationally expensive* for high-capacity models like Transformers.

The Research Gap: Our Motivation

We identified three critical limitations in prior work:

1. **Architectural Mismatch:** Most FSL-NIDS use CNNs, forcing tabular network data into 2D images. This imposes an artificial structure and loses information.
2. **Adaptability vs. Cost:** A trade-off exists. Metric-based is efficient but weak. Optimization-based is strong but *too computationally expensive* for high-capacity models like Transformers.
3. **Dataset Realism:** Many studies use older datasets (e.g., NSL-KDD) that don't reflect modern, heterogeneous IoT traffic.

Our Goal: Design a framework that is...

- Architecturally appropriate (**FT-Transformer** for tabular data).
- Powerfully adaptive (**Optimization-based** meta-learning).
- Computationally **efficient** and practical.
- Validated on a **realistic IoT dataset**.

Proposed Framework: MetaFTT Overview

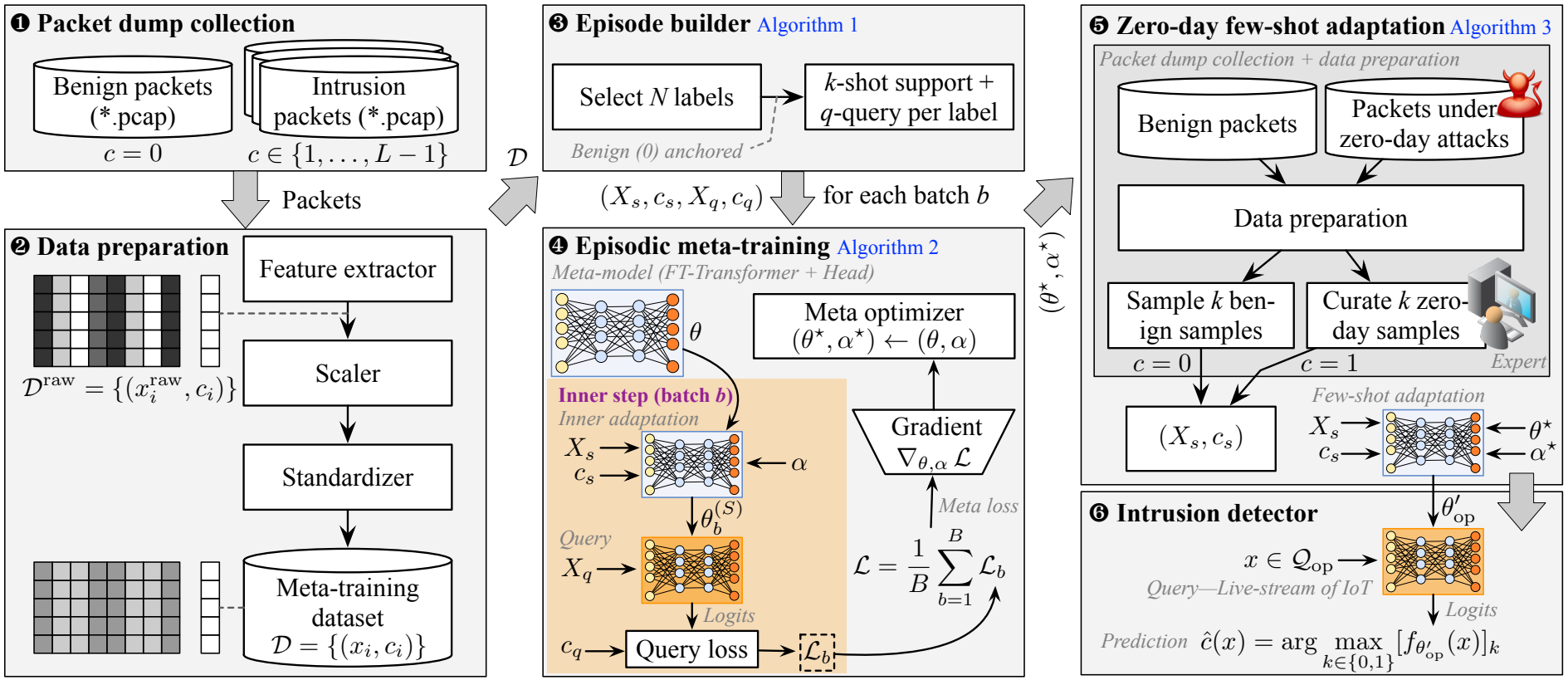


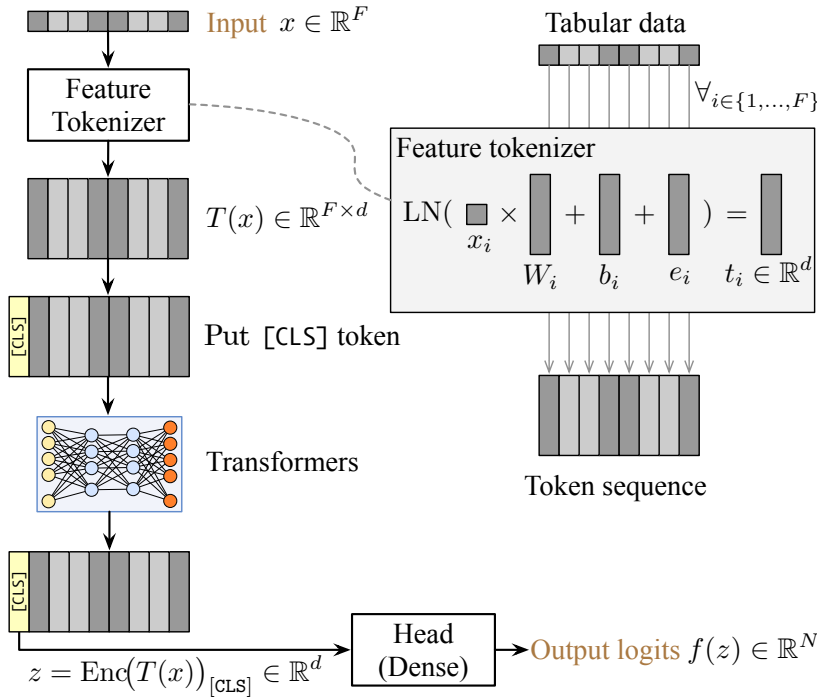
Figure: Proposed framework.

Core Component 1: FT-Transformer

Why FT-Transformer? It is natively designed for tabular data.

- **Addresses the "Architectural Mismatch".** No more forcing tabular data into 2D images.

- **How it works (simplified):**
  - **1. Feature Tokenizer:** Projects each of the F scalar features into a d-dimensional embedding (a vector).
  - **2. Transformer Encoder:** Applies self-attention *across the feature embeddings* to learn deep, complex interactions.
  - **3. [CLS] Token:** Aggregates the full sequence of feature information for a final classification.



Core Component 1: FT-Transformer—implementation details

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
FTTransformer                            [1, 5]                    160
├─FeatureTokenizer: 1-1                  [1, 100, 160]             48,000
│    └─LayerNorm: 2-1                    [1, 100, 160]             320
├─ModuleList: 1-2                        --                        --
│    └─TransformerBlock: 2-2             [1, 101, 160]             --
│    │    └─LayerNorm: 3-1               [1, 101, 160]             320
│    │    └─MultiheadAttention: 3-2      [1, 101, 160]             103,040
│    │    └─Dropout: 3-3                 [1, 101, 160]             --
│    │    └─LayerNorm: 3-4               [1, 101, 160]             320
│    │    └─Sequential: 3-5              [1, 101, 160]             205,600
│    │    └─Dropout: 3-6                 [1, 101, 160]             --
│    └─TransformerBlock: 2-3             [1, 101, 160]             --
│    │    └─LayerNorm: 3-7               [1, 101, 160]             320
│    │    └─MultiheadAttention: 3-8      [1, 101, 160]             103,040
│    │    └─Dropout: 3-9                 [1, 101, 160]             --
│    │    └─LayerNorm: 3-10              [1, 101, 160]             320
│    │    └─Sequential: 3-11             [1, 101, 160]             205,600
│    │    └─Dropout: 3-12                [1, 101, 160]             --
├─LayerNorm: 1-3                         [1, 101, 160]             320
├─Linear: 1-4                            [1, 5]                    805
==========================================================================================
Total params: 668,165
Trainable params: 668,165
Non-trainable params: 0
Total mult-adds (Units.MEGABYTES): 0.41
==========================================================================================
Input size (MB): 0.00
Forward/backward pass size (MB): 2.07
Params size (MB): 1.66
Estimated Total Size (MB): 3.72
==========================================================================================
```

Core Component 2: Efficient Meta-Learning

Problem: Full optimization-based meta-learning on a Transformer is computationally infeasible (e.g., predicted >80GB VRAM).

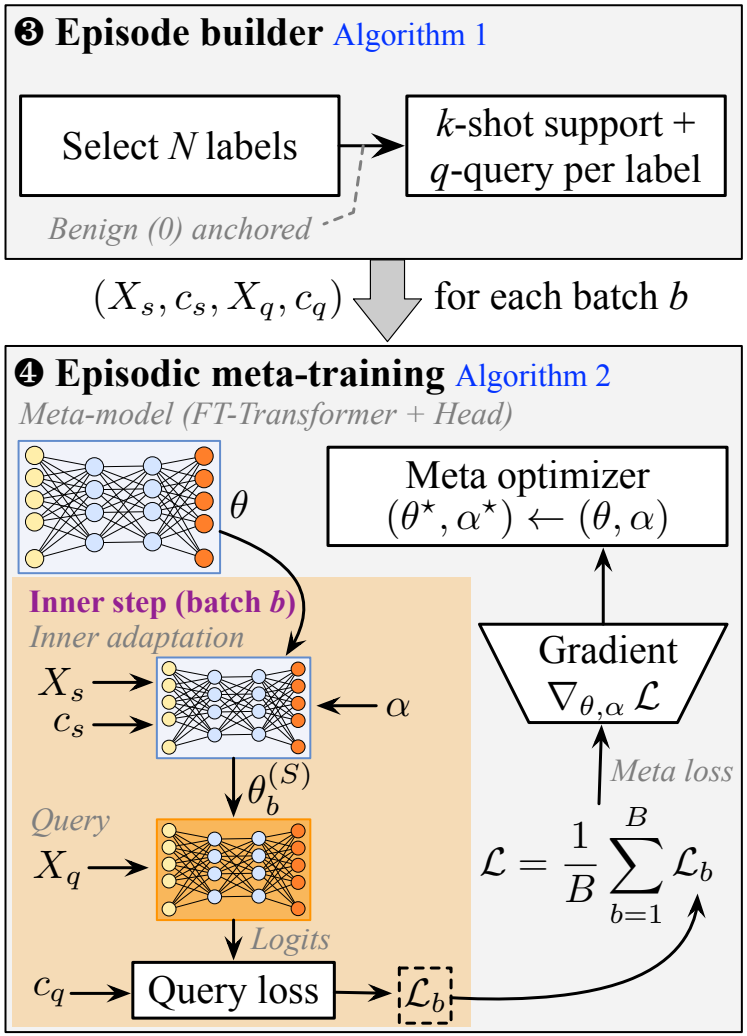
Our Solution: A two-part efficiency strategy.

1. **First-Order Approximation (FO-MAML):**
   - We omit the expensive second-order (Hessian) calculations during the outer-loop update.
2. **Lightweight Inner Adaptation (ANIL-inspired):**
   - In the inner loop (adaptation), we **freeze** the heavyweight Transformer blocks (ψ).
   - We **only** adapt the lightweight parameters (ω):
     - The final Classification Head
     - All LayerNorm parameters

Result: Makes meta-learning a high-capacity Transformer practical and tractable.

Core Component 2: Efficient Meta-Learning—in detail

Episode builder: Episodes are constructed as balanced N-way tasks with disjoint support and query subsets per class.

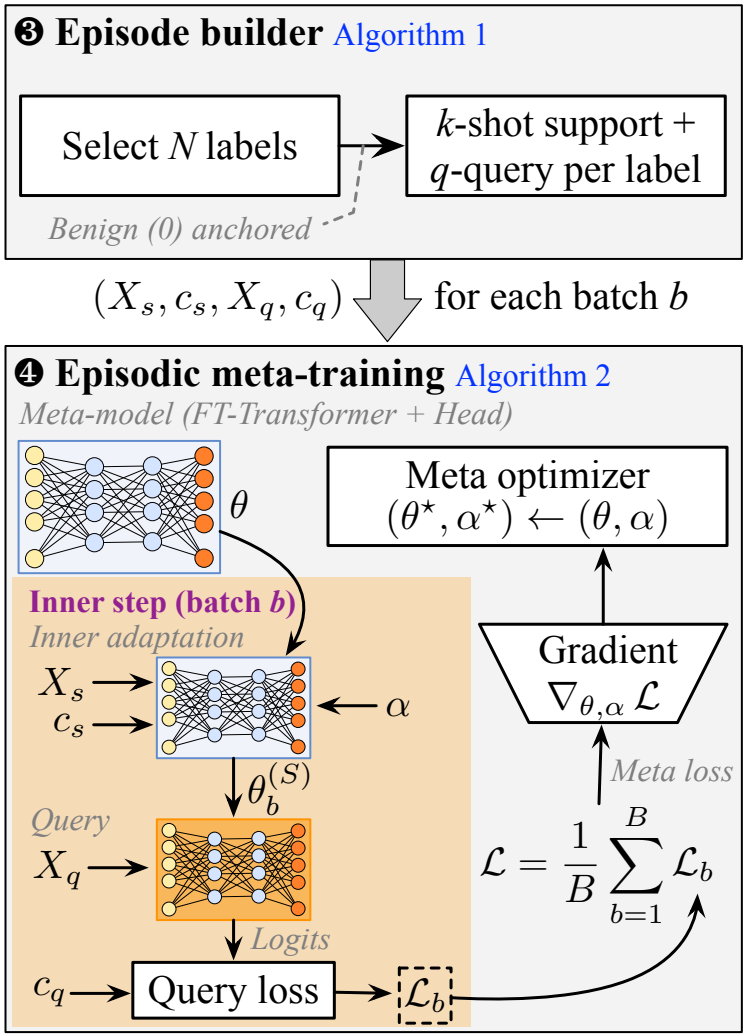


Require: Dataset $\mathcal{D} = \{(x_i, c_i)\}_{i \in \mathcal{I}}$ with $x_i \in \mathbb{R}^F$, $c_i \in \{0, \ldots, L-1\}$
Require: Classes per episode $N \geq 2$, support per class k, query per class q
Convention: all draws are uniform without replacement.

$n_s := N \cdot k$ and $n_q := N \cdot q$
Ensure: $X_s \in \mathbb{R}^{n_s \times F}$, $c_s \in \{0, \ldots, N-1\}^{n_s}$
Ensure: $X_q \in \mathbb{R}^{n_q \times F}$, $c_q \in \{0, \ldots, N-1\}^{n_q}$
function BUILDEPISODE($\mathcal{D}$, N, k, q)

Core Component 2: Efficient Meta-Learning—in detail

Episode builder: Episodes are constructed as balanced N-way tasks with disjoint support and query subsets per class.

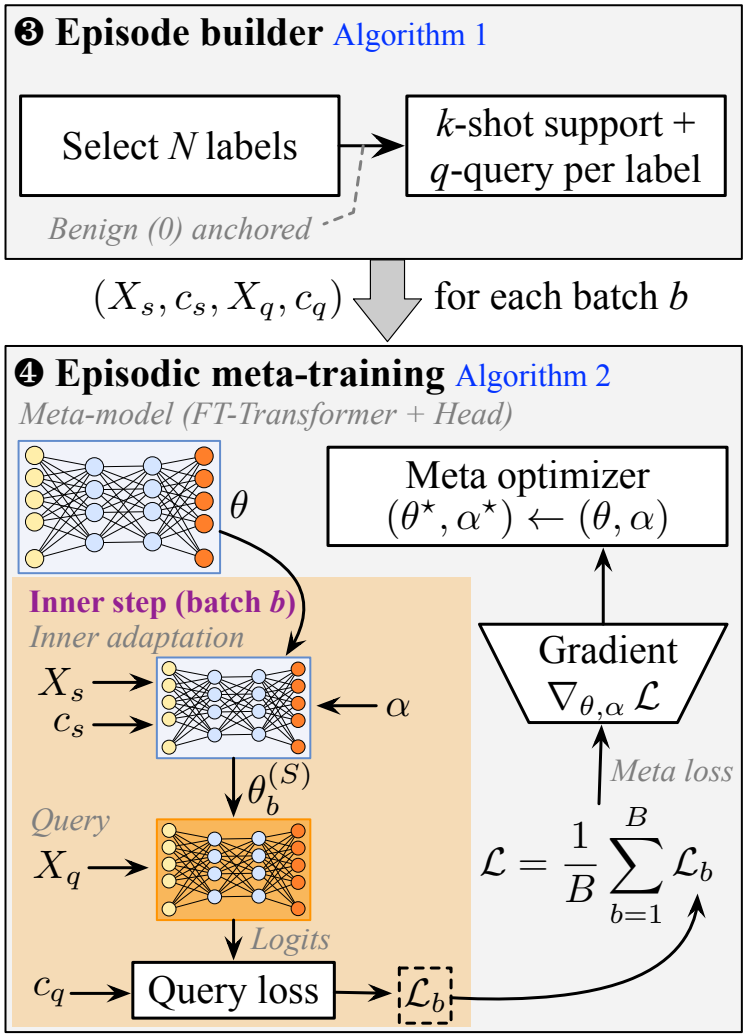


Require: Dataset $\mathcal{D} = \{(x_i, c_i)\}_{i \in \mathcal{I}}$ with $x_i \in \mathbb{R}^F$, $c_i \in \{0, \ldots, L-1\}$
Require: Classes per episode $N \geq 2$, support per class k, query per class q
Convention: all draws are uniform without replacement.

$n_s := N \cdot k$ and $n_q := N \cdot q$
Ensure: $X_s \in \mathbb{R}^{n_s \times F}$, $c_s \in \{0, \ldots, N-1\}^{n_s}$
Ensure: $X_q \in \mathbb{R}^{n_q \times F}$, $c_q \in \{0, \ldots, N-1\}^{n_q}$
function BUILDEPISODE($\mathcal{D}, N, k, q$)
    $\mathcal{C} \leftarrow \{ c \mid \exists i \in \mathcal{I} : c_i = c \}$ ▷ available class IDs
    $\mathcal{C}_{\text{other}} \leftarrow \mathcal{C} \setminus \{0\}$

Core Component 2: Efficient Meta-Learning—in detail

Episode builder: Episodes are constructed as balanced N-way tasks with disjoint support and query subsets per class.

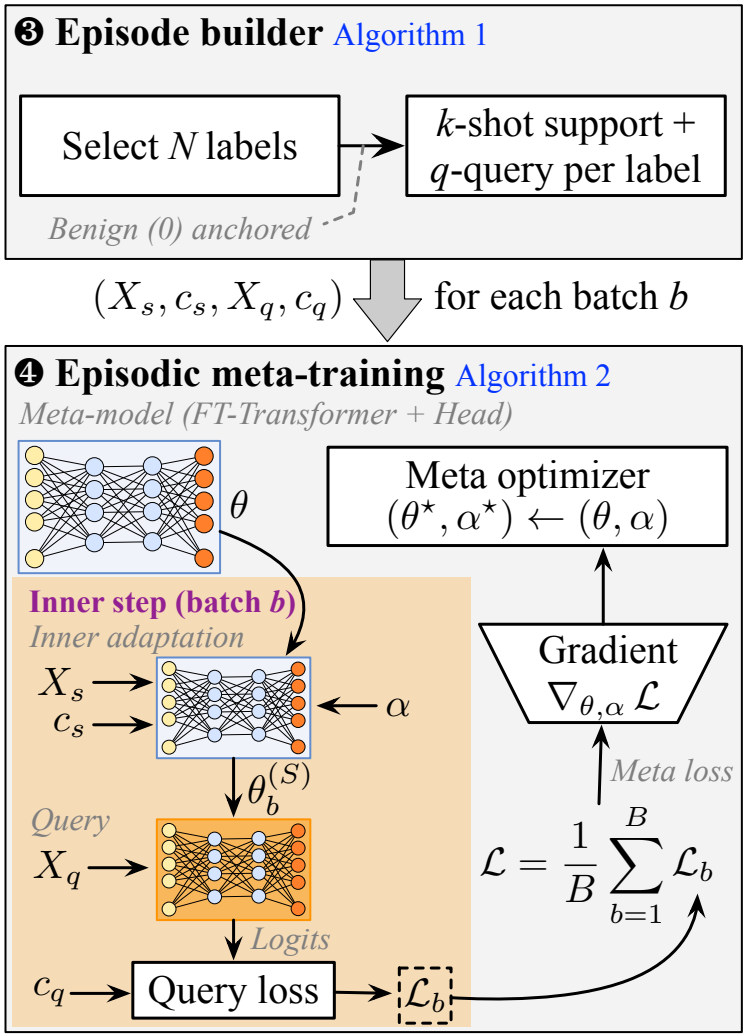


Require: Dataset $\mathcal{D} = \{(x_i, c_i)\}_{i \in \mathcal{I}}$ with $x_i \in \mathbb{R}^F$, $c_i \in \{0, \ldots, L-1\}$
Require: Classes per episode $N \geq 2$, support per class k, query per class q
Convention: all draws are uniform without replacement.

$n_s := N \cdot k$ and $n_q := N \cdot q$
Ensure: $X_s \in \mathbb{R}^{n_s \times F}$, $c_s \in \{0, \ldots, N-1\}^{n_s}$
Ensure: $X_q \in \mathbb{R}^{n_q \times F}$, $c_q \in \{0, \ldots, N-1\}^{n_q}$
function BUILDEPISODE($\mathcal{D}$, N, k, q)
  $\mathcal{C} \leftarrow \{ c \mid \exists i \in \mathcal{I} : c_i = c \}$ ▷ available class IDs
  $\mathcal{C}_{\text{other}} \leftarrow \mathcal{C} \setminus \{0\}$
  Select $A \subset \mathcal{C}_{\text{other}}$ with $|A| = N-1$
  Define $\phi : \{0, \ldots, N-1\} \to \{0\} \cup A$ by $\phi(0)=0$ and a bijection on the rest

Core Component 2: Efficient Meta-Learning—in detail

Episode builder: Episodes are constructed as balanced N-way tasks with disjoint support and query subsets per class.

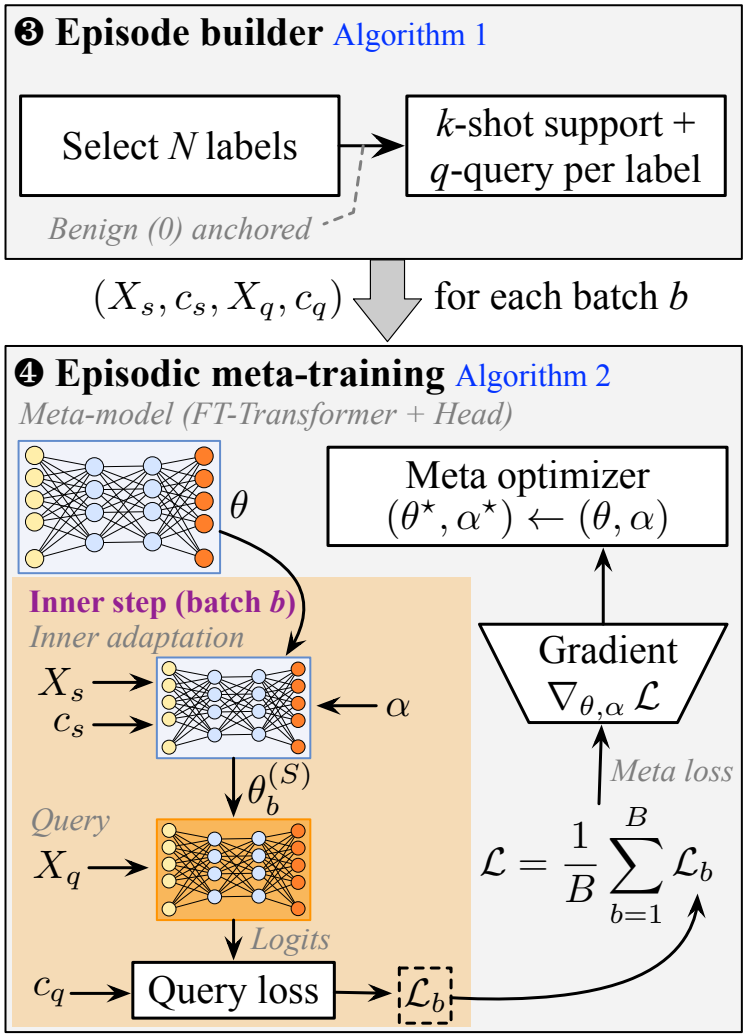


Require: Dataset $\mathcal{D} = \{(x_i, c_i)\}_{i \in \mathcal{I}}$ with $x_i \in \mathbb{R}^F$, $c_i \in \{0, \ldots, L-1\}$
Require: Classes per episode $N \geq 2$, support per class k, query per class q
Convention: all draws are uniform without replacement.

$n_s := N \cdot k$ and $n_q := N \cdot q$
Ensure: $X_s \in \mathbb{R}^{n_s \times F}$, $c_s \in \{0, \ldots, N-1\}^{n_s}$
Ensure: $X_q \in \mathbb{R}^{n_q \times F}$, $c_q \in \{0, \ldots, N-1\}^{n_q}$
function BUILDEPISODE($\mathcal{D}$, N, k, q)
  $\mathcal{C} \leftarrow \{ c \mid \exists i \in \mathcal{I} : c_i = c \}$ ▷ available class IDs
  $\mathcal{C}_{\text{other}} \leftarrow \mathcal{C} \setminus \{0\}$
  Select $A \subset \mathcal{C}_{\text{other}}$ with $|A| = N-1$
  Define $\phi : \{0, \ldots, N-1\} \to \{0\} \cup A$ by $\phi(0)=0$ and a bijection on the rest
  $X_s \leftarrow [\,]$, $c_s \leftarrow [\,]$, $X_q \leftarrow [\,]$, $c_q \leftarrow [\,]$

Core Component 2: Efficient Meta-Learning—in detail

Episode builder: Episodes are constructed as balanced N-way tasks with disjoint support and query subsets per class.

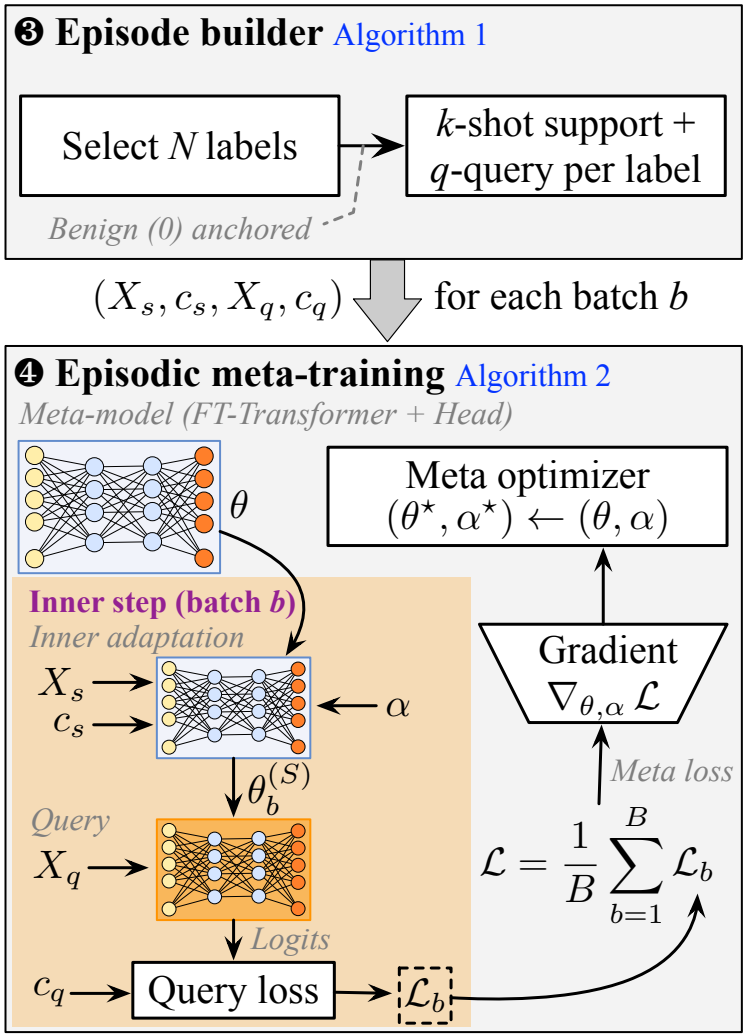


Require: Dataset $\mathcal{D} = \{(x_i, c_i)\}_{i \in \mathcal{I}}$ with $x_i \in \mathbb{R}^F$, $c_i \in \{0, \ldots, L-1\}$
Require: Classes per episode $N \geq 2$, support per class k, query per class q
Convention: all draws are uniform without replacement.

$n_s := N \cdot k$ and $n_q := N \cdot q$
Ensure: $X_s \in \mathbb{R}^{n_s \times F}$, $c_s \in \{0, \ldots, N-1\}^{n_s}$
Ensure: $X_q \in \mathbb{R}^{n_q \times F}$, $c_q \in \{0, \ldots, N-1\}^{n_q}$
function BUILDEPISODE($\mathcal{D}$, N, k, q)
  $\mathcal{C} \leftarrow \{ c \mid \exists i \in \mathcal{I} : c_i = c \}$ ▷ available class IDs
  $\mathcal{C}_{\text{other}} \leftarrow \mathcal{C} \setminus \{0\}$
  Select $A \subset \mathcal{C}_{\text{other}}$ with $|A| = N-1$
  Define $\phi : \{0, \ldots, N-1\} \to \{0\} \cup A$ by $\phi(0)=0$ and a bijection on the rest
  $X_s \leftarrow [\,]$, $c_s \leftarrow [\,]$, $X_q \leftarrow [\,]$, $c_q \leftarrow [\,]$
  for $j = 0$ **to** $N-1$ **do**

Core Component 2: Efficient Meta-Learning—in detail

Episode builder: Episodes are constructed as balanced N-way tasks with disjoint support and query subsets per class.

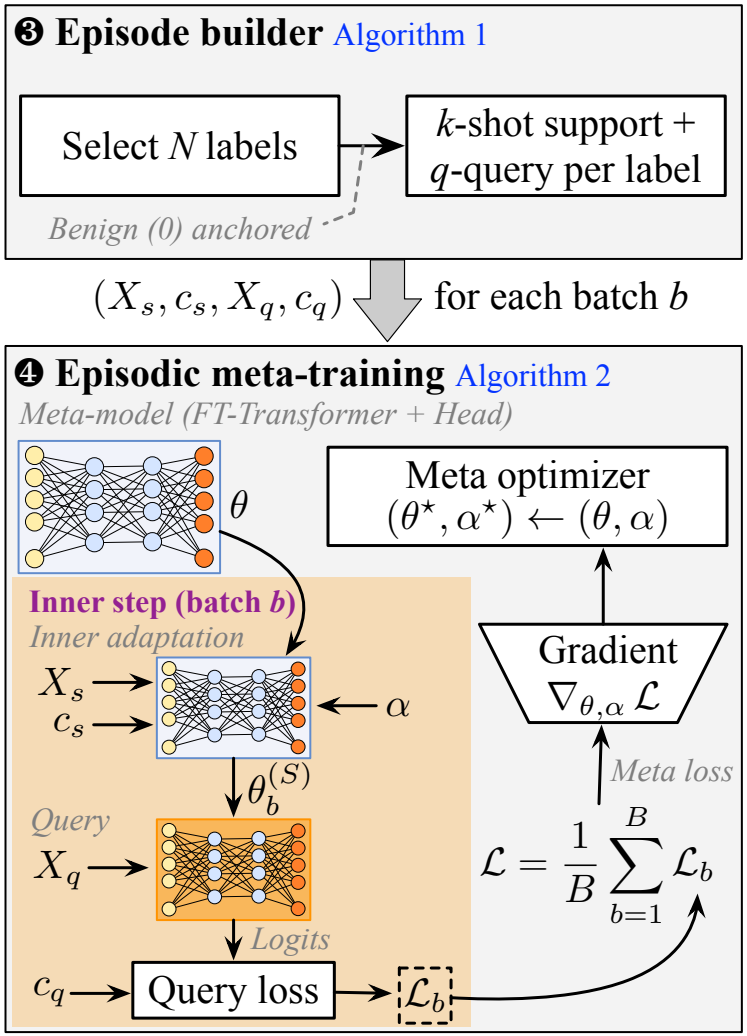


Require: Dataset $\mathcal{D} = \{(x_i, c_i)\}_{i \in \mathcal{I}}$ with $x_i \in \mathbb{R}^F$, $c_i \in \{0, \ldots, L-1\}$
Require: Classes per episode $N \geq 2$, support per class k, query per class q
Convention: all draws are uniform without replacement.

$n_s := N \cdot k$ and $n_q := N \cdot q$
Ensure: $X_s \in \mathbb{R}^{n_s \times F}$, $c_s \in \{0, \ldots, N-1\}^{n_s}$
Ensure: $X_q \in \mathbb{R}^{n_q \times F}$, $c_q \in \{0, \ldots, N-1\}^{n_q}$
function BUILDEPISODE($\mathcal{D}$, N, k, q)
  $\mathcal{C} \leftarrow \{ c \mid \exists i \in \mathcal{I} : c_i = c \}$ ▷ available class IDs
  $\mathcal{C}_{\text{other}} \leftarrow \mathcal{C} \setminus \{0\}$
  Select $A \subset \mathcal{C}_{\text{other}}$ with $|A| = N-1$
  Define $\phi : \{0, \ldots, N-1\} \to \{0\} \cup A$ by $\phi(0)=0$ and a bijection on the rest
  $X_s \leftarrow [\,]$, $c_s \leftarrow [\,]$, $X_q \leftarrow [\,]$, $c_q \leftarrow [\,]$
  for $j = 0$ **to** $N-1$ **do**
    $c \leftarrow \phi(j)$; $\mathcal{I}_c \leftarrow \{ i \in \mathcal{I} \mid c_i = c \}$
    Choose $S_c \subset \mathcal{I}_c$ with $|S_c| = k$
    Choose $Q_c \subset \mathcal{I}_c \setminus S_c$ with $|Q_c| = q$

Core Component 2: Efficient Meta-Learning—in detail

Episode builder: Episodes are constructed as balanced N-way tasks with disjoint support and query subsets per class.

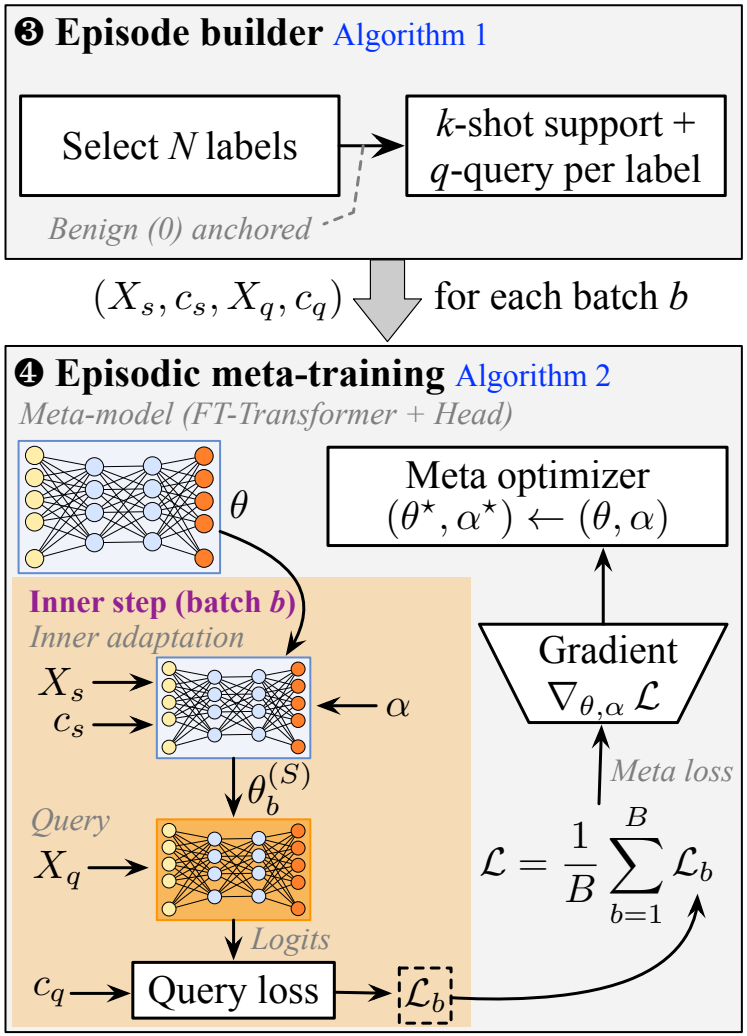


Require: Dataset $\mathcal{D} = \{(x_i, c_i)\}_{i \in \mathcal{I}}$ with $x_i \in \mathbb{R}^F$, $c_i \in \{0, \ldots, L-1\}$
Require: Classes per episode $N \geq 2$, support per class k, query per class q
Convention: all draws are uniform without replacement.

$n_s := N \cdot k$ and $n_q := N \cdot q$
Ensure: $X_s \in \mathbb{R}^{n_s \times F}$, $c_s \in \{0, \ldots, N-1\}^{n_s}$
Ensure: $X_q \in \mathbb{R}^{n_q \times F}$, $c_q \in \{0, \ldots, N-1\}^{n_q}$
function BUILDEPISODE($\mathcal{D}$, N, k, q)
  $\mathcal{C} \leftarrow \{ c \mid \exists i \in \mathcal{I} : c_i = c \}$ ▷ available class IDs
  $\mathcal{C}_{\text{other}} \leftarrow \mathcal{C} \setminus \{0\}$
  Select $A \subset \mathcal{C}_{\text{other}}$ with $|A| = N-1$
  Define $\phi : \{0, \ldots, N-1\} \to \{0\} \cup A$ by $\phi(0)=0$ and a bijection on the rest
  $X_s \leftarrow [\,]$, $c_s \leftarrow [\,]$, $X_q \leftarrow [\,]$, $c_q \leftarrow [\,]$
  for $j = 0$ **to** $N-1$ **do**
    $c \leftarrow \phi(j)$; $\mathcal{I}_c \leftarrow \{ i \in \mathcal{I} \mid c_i = c \}$
    Choose $S_c \subset \mathcal{I}_c$ with $|S_c| = k$
    Choose $Q_c \subset \mathcal{I}_c \setminus S_c$ with $|Q_c| = q$
    $X_s \leftarrow X_s \,\|\, \{x_i : i \in S_c\}$
    $c_s \leftarrow c_s \,\|\, (j, \ldots, j)$ (k times)
    $X_q \leftarrow X_q \,\|\, \{x_i : i \in Q_c\}$
    $c_q \leftarrow c_q \,\|\, (j, \ldots, j)$ (q times)
  end for

Core Component 2: Efficient Meta-Learning—in detail

Episode builder: Episodes are constructed as balanced N-way tasks with disjoint support and query subsets per class.



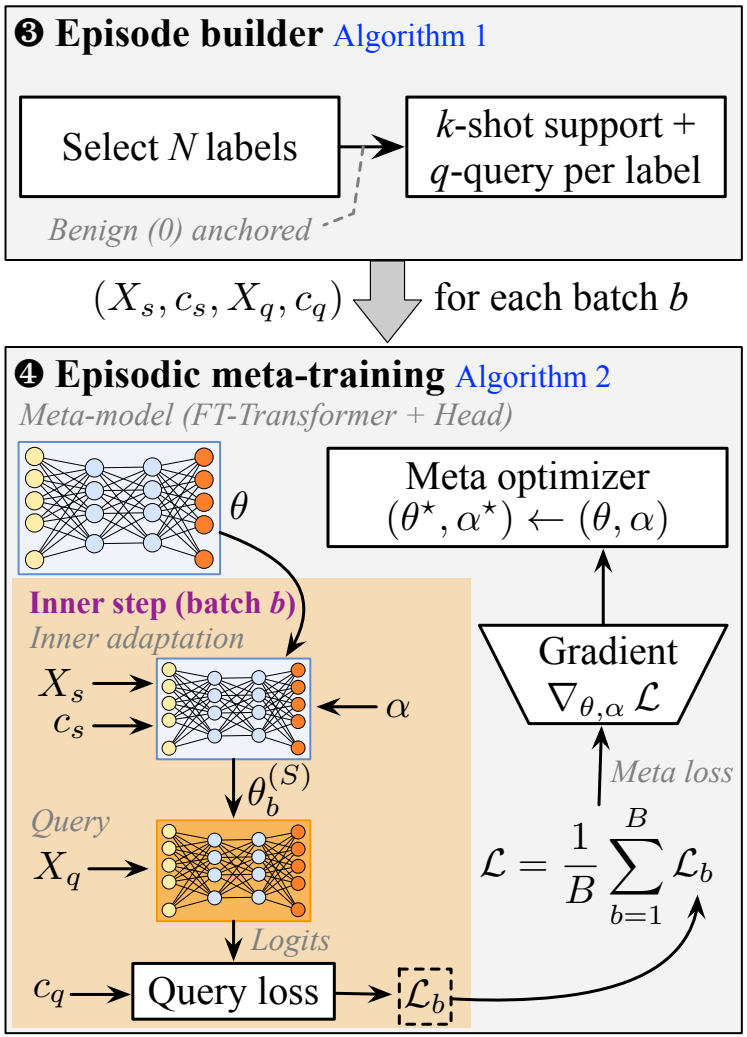
Require: Dataset $\mathcal{D} = \{(x_i, c_i)\}_{i \in \mathcal{I}}$ with $x_i \in \mathbb{R}^F$, $c_i \in \{0, \ldots, L-1\}$
Require: Classes per episode $N \geq 2$, support per class k, query per class q
Convention: all draws are uniform without replacement.

$n_s := N \cdot k$ and $n_q := N \cdot q$
Ensure: $X_s \in \mathbb{R}^{n_s \times F}$, $c_s \in \{0, \ldots, N-1\}^{n_s}$
Ensure: $X_q \in \mathbb{R}^{n_q \times F}$, $c_q \in \{0, \ldots, N-1\}^{n_q}$
function BUILDEPISODE($\mathcal{D}$, N, k, q)
  $\mathcal{C} \leftarrow \{ c \mid \exists i \in \mathcal{I} : c_i = c \}$ ▷ available class IDs
  $\mathcal{C}_{\text{other}} \leftarrow \mathcal{C} \setminus \{0\}$
  Select $A \subset \mathcal{C}_{\text{other}}$ with $|A| = N-1$
  Define $\phi : \{0, \ldots, N-1\} \to \{0\} \cup A$ by $\phi(0)=0$ and a bijection on the rest
  $X_s \leftarrow [\,]$, $c_s \leftarrow [\,]$, $X_q \leftarrow [\,]$, $c_q \leftarrow [\,]$
  for $j = 0$ **to** $N-1$ **do**
    $c \leftarrow \phi(j)$; $\mathcal{I}_c \leftarrow \{ i \in \mathcal{I} \mid c_i = c \}$
    Choose $S_c \subset \mathcal{I}_c$ with $|S_c| = k$
    Choose $Q_c \subset \mathcal{I}_c \setminus S_c$ with $|Q_c| = q$
    $X_s \leftarrow X_s \,\|\, \{x_i : i \in S_c\}$
    $c_s \leftarrow c_s \,\|\, (j, \ldots, j)$ (k times)
    $X_q \leftarrow X_q \,\|\, \{x_i : i \in Q_c\}$
    $c_q \leftarrow c_q \,\|\, (j, \ldots, j)$ (q times)
  end for
  return (X_s, c_s, X_q, c_q)
end function

Core Component 2: Efficient Meta-Learning—in detail

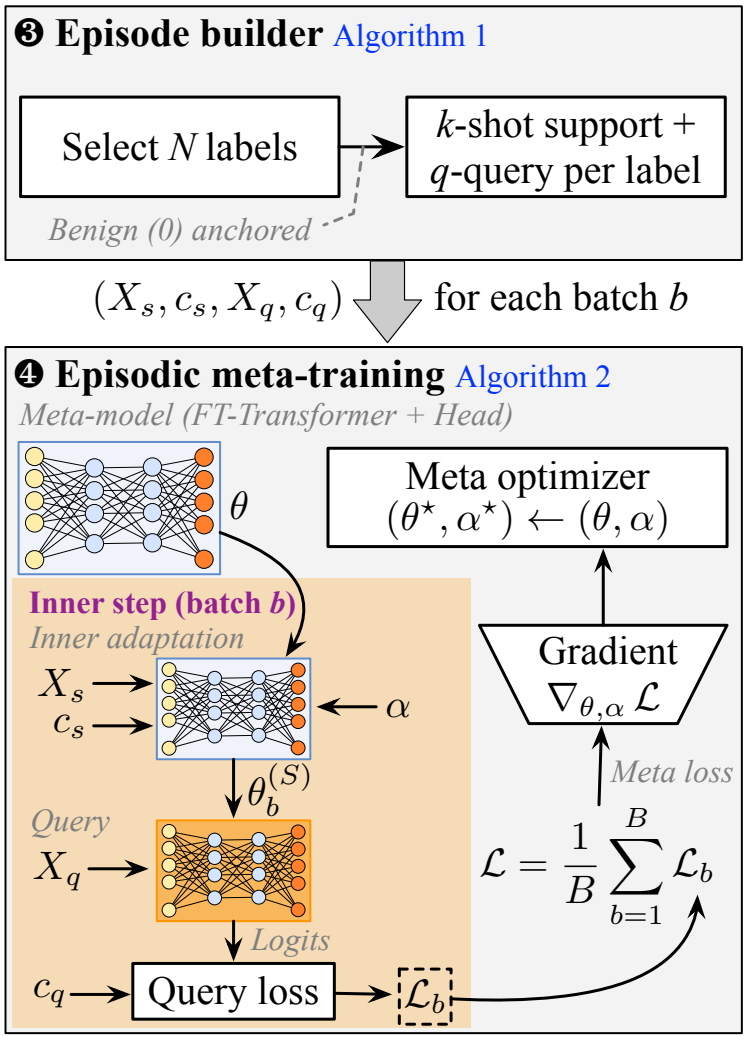
Episodic meta-training Meta-loss

Episodic meta-training Meta-loss



For each eposide $\tau = (X_s, c_s, X_q, c_q)$, the within-episode training loss on the support is the average cross-entropy,

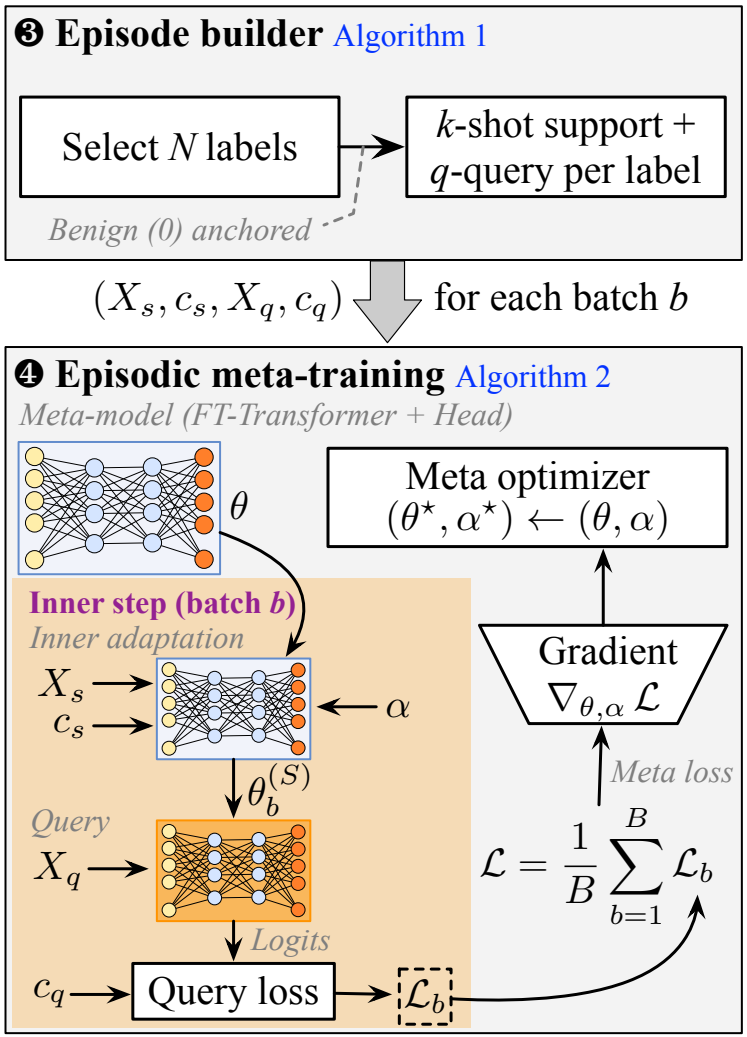
Episodic meta-training Meta-loss



For each eposide $\tau = (X_s, c_s, X_q, c_q)$, the within-episode training loss on the support is the average cross-entropy,

$$\ell_s(\theta; \tau) = \frac{1}{|X_s|} \sum_{(x,c)\in(X_s,c_s)} \mathcal{L}\big(f_\theta(x),\, c\big), \qquad (1)$$

Episodic meta-training Meta-loss



For each eposide $\tau = (X_s, c_s, X_q, c_q)$, the within-episode training loss on the support is the average cross-entropy,

$$\ell_s(\theta;\tau) = \frac{1}{|X_s|}\sum_{(x,c)\in(X_s,c_s)} \mathcal{L}\big(f_\theta(x),\, c\big), \qquad (1)$$

$$\ell_q(\theta;\tau) = \frac{1}{|X_q|}\sum_{(x,c)\in(X_q,c_q)} \mathcal{L}\big(f_\theta(x),\, c\big) \qquad (2)$$

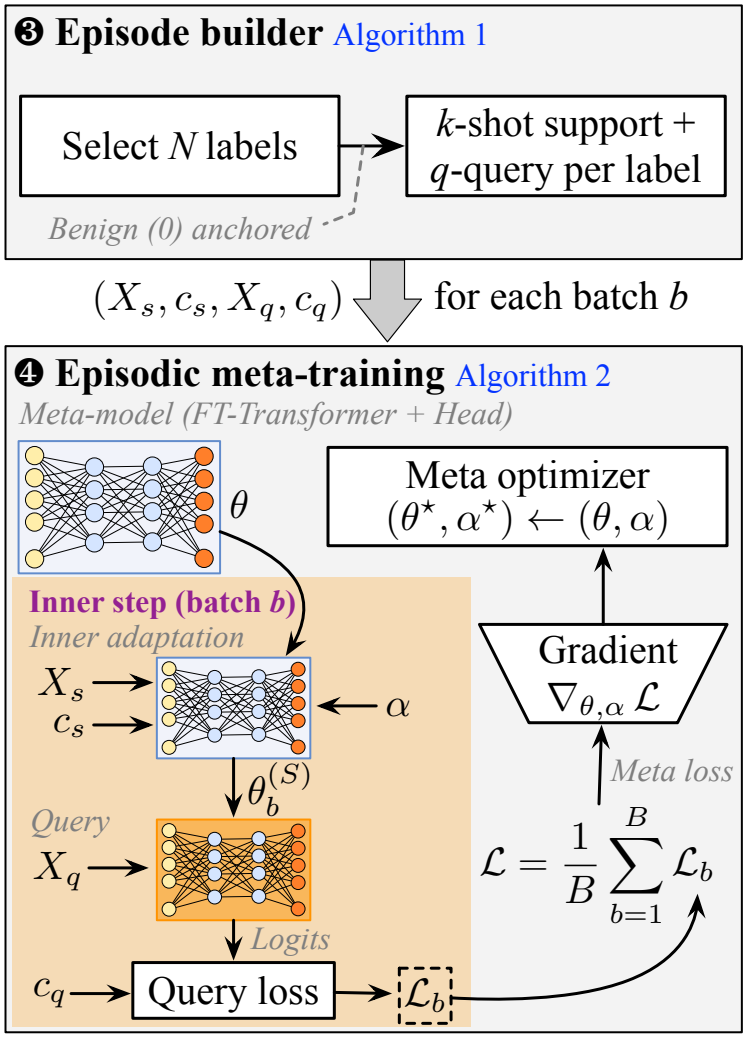
with $\mathcal{L}$ the standard cross-entropy.

Core Component 2: Efficient Meta-Learning—in detail

Episodic meta-training Meta-loss → Inner-loop adaptation

Core Component 2: Efficient Meta-Learning—in detail

Episodic meta-training Meta-loss → Inner-loop adaptation



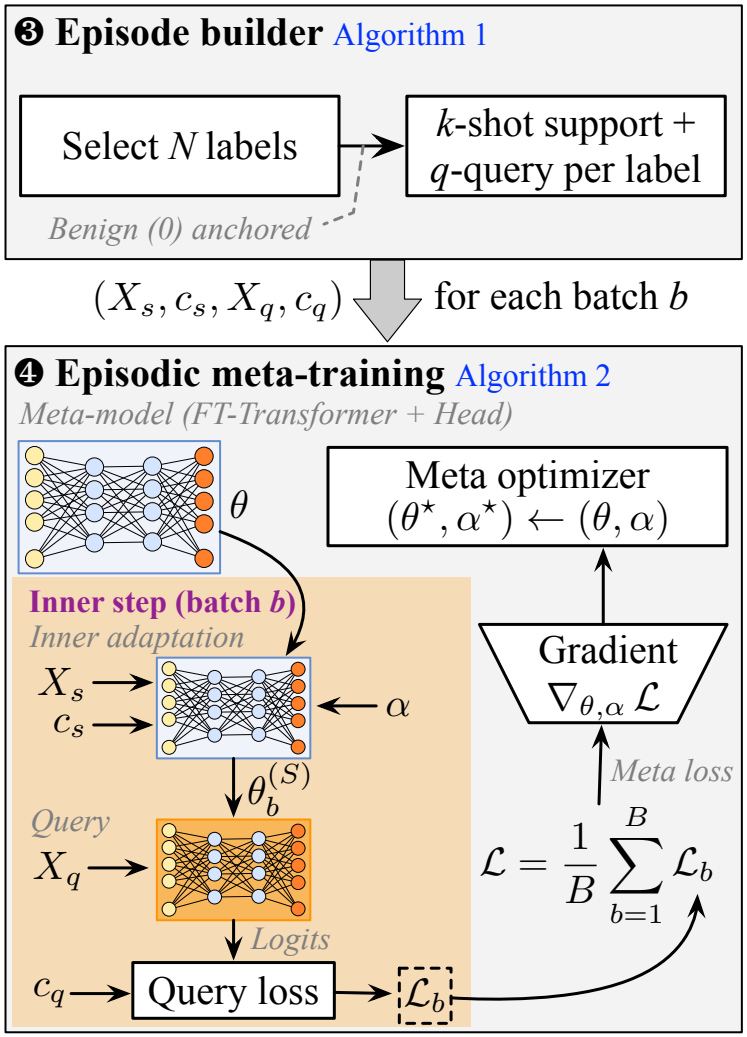
We adapt θ on the support by optimization-based updates and evaluate on the query.

In the MetaSGD formulation, a learnable step-size tensor α of the same structure as θ scales the inner gradients:

$$\theta^{(t+1)} = \theta^{(t)} - \alpha \odot \nabla_{\theta^{(t)}} \ell_s\big(\theta^{(t)}; \tau\big). \quad (3)$$

Core Component 2: Efficient Meta-Learning—in detail

Episodic meta-training Meta-loss → Inner-loop adaptation



We adapt θ on the support by optimization-based updates and evaluate on the query.

In the MetaSGD formulation, a learnable step-size tensor α of the same structure as θ scales the inner gradients:

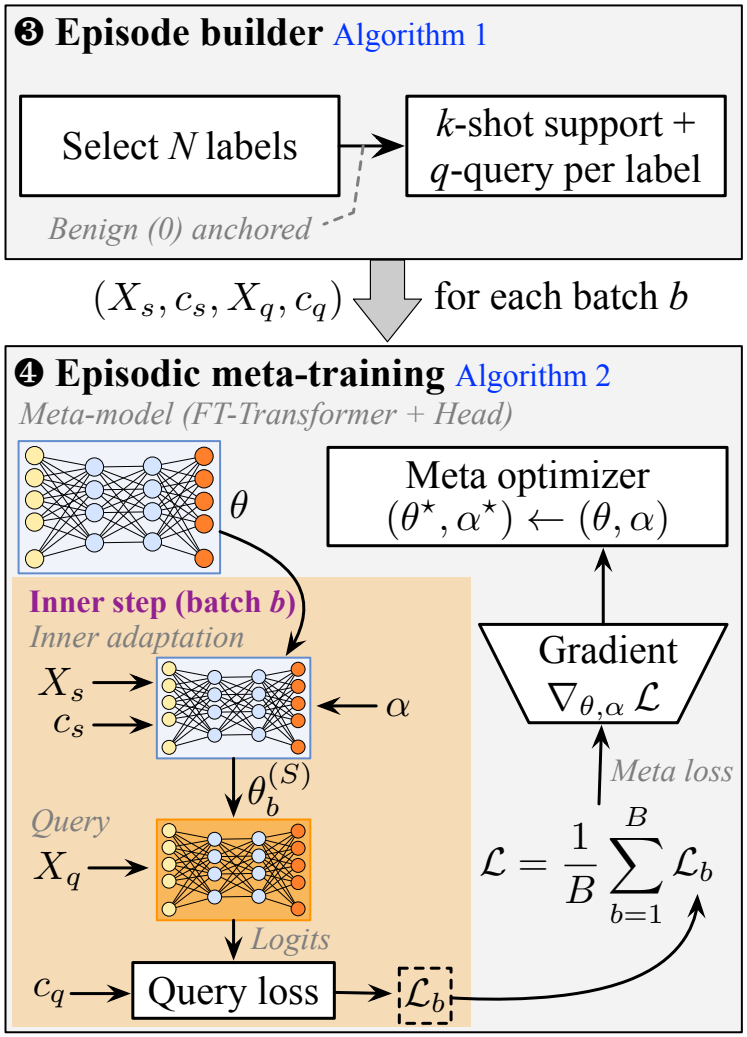
$$\theta^{(t+1)} = \theta^{(t)} - \alpha \odot \nabla_{\theta^{(t)}} \ell_s\big(\theta^{(t)}; \tau\big). \quad (3)$$

where,

- $t = 0, \ldots, S-1$,
- $\theta^{(0)} \equiv \theta$,
- S is the number of inner steps.

Core Component 2: Efficient Meta-Learning—in detail

Episodic meta-training Meta-loss → Inner-loop adaptation



We adapt θ on the support by optimization-based updates and evaluate on the query.

In the MetaSGD formulation, a learnable step-size tensor α of the same structure as θ scales the inner gradients:

$$\theta^{(t+1)} = \theta^{(t)} - \alpha \odot \nabla_{\theta^{(t)}} \ell_s\big(\theta^{(t)}; \tau\big). \quad (3)$$

where,

- $t = 0, \ldots, S-1$,
- $\theta^{(0)} \equiv \theta$,
- S is the number of inner steps.

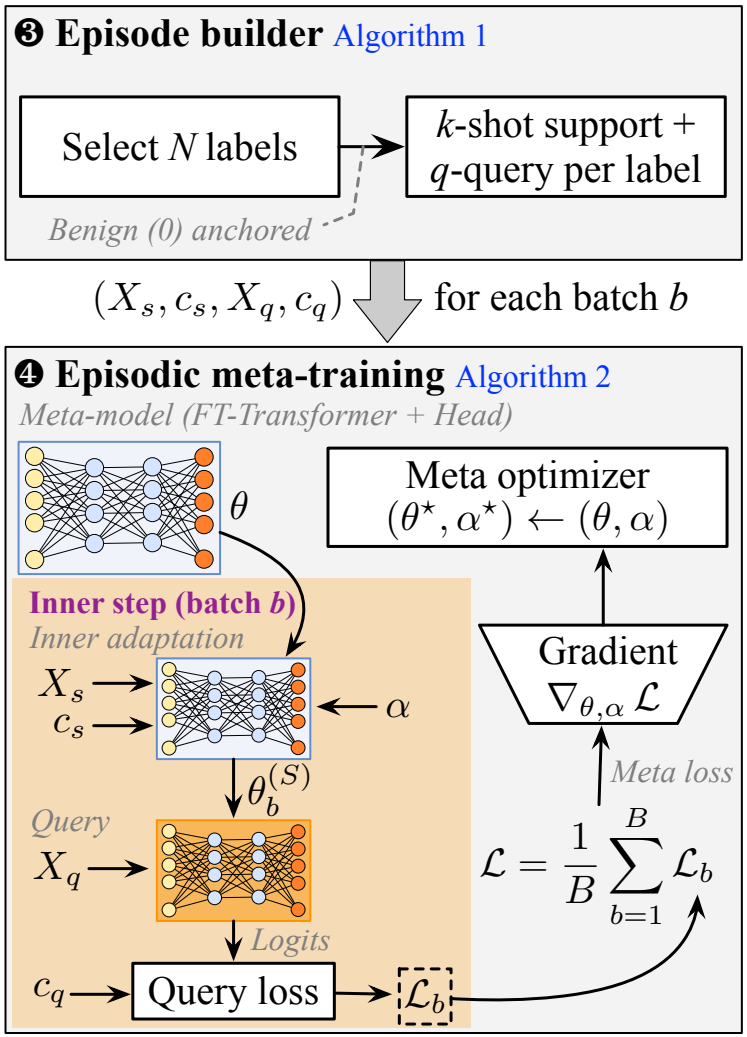
The adapted parameters after S steps are $\theta' = \theta^{(S)}$.

Core Component 2: Efficient Meta-Learning—in detail

Episodic meta-training Meta-loss → Inner-loop adaptation→ Outer-loop meta-objective

Core Component 2: Efficient Meta-Learning—in detail

Episodic meta-training Meta-loss → Inner-loop adaptation→ Outer-loop meta-objective

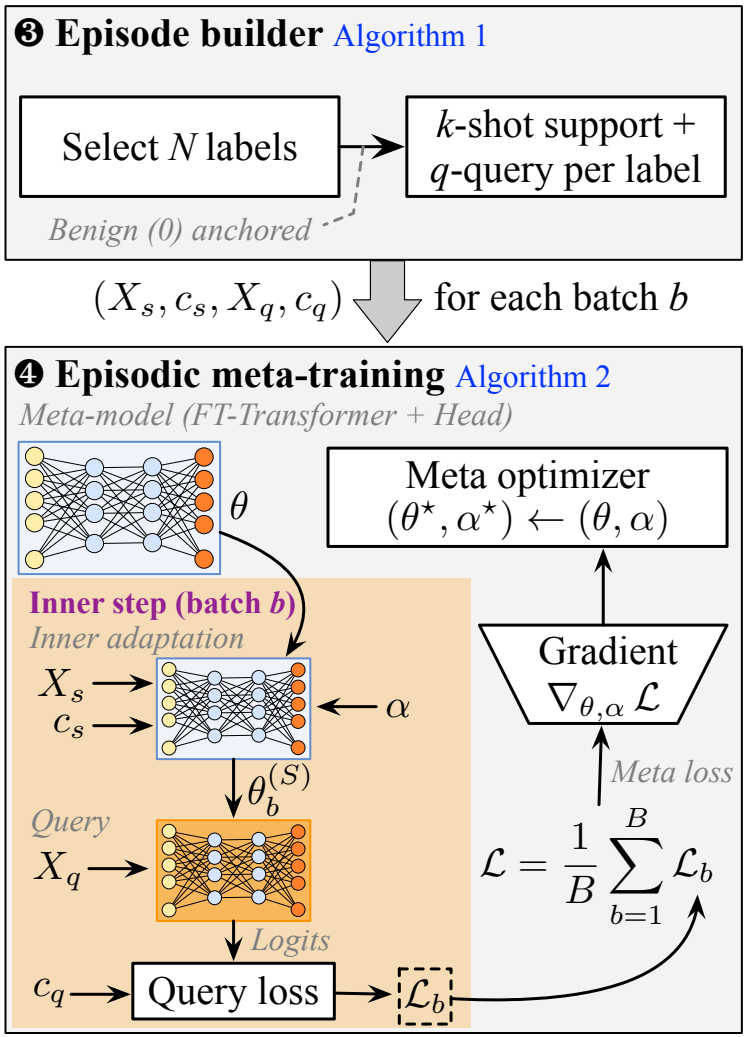


The meta-learner minimizes the expected query loss after adaptation:

$$\min_{\theta,\alpha} \quad \mathbb{E}_{\tau\sim p(\tau)}\left[\, \ell_q\big(\theta'(\theta,\alpha;\tau);\tau\big)\,\right] \tag{4}$$

Core Component 2: Efficient Meta-Learning—in detail

Episodic meta-training Meta-loss → Inner-loop adaptation→ Outer-loop meta-objective



The meta-learner minimizes the expected query loss after adaptation:

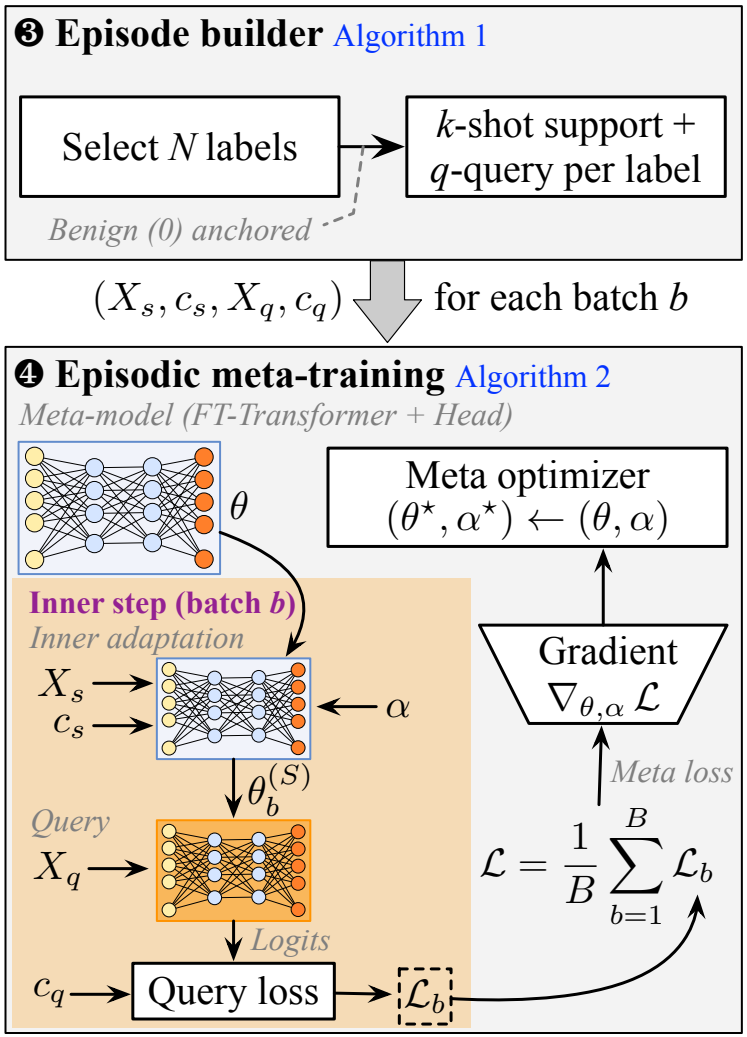
$$\min_{\theta,\alpha} \; \mathbb{E}_{\tau\sim p(\tau)}\left[\, \ell_q\big(\theta'(\theta,\alpha;\tau);\tau\big)\,\right] \tag{4}$$

where,

- $p(\tau)$ is the episode distribution induced by the data processor.

Core Component 2: Efficient Meta-Learning—in detail

Episodic meta-training Meta-loss → Inner-loop adaptation→ Outer-loop meta-objective



The meta-learner minimizes the expected query loss after adaptation:

$$\min_{\theta,\alpha} \; \mathbb{E}_{\tau \sim p(\tau)} \left[\ell_q \left(\theta'(\theta, \alpha; \tau); \tau \right) \right] \tag{4}$$

where,

- $p(\tau)$ is the episode distribution induced by the data processor.

In practice, we use a meta-batch of B episodes $\{\tau_b\}_{b=1}^{B}$ per outer iteration and optimize the averaged query loss with Adam.

Core Component 2: Efficient Meta-Learning—in detail

Episodic meta-training Meta-loss → Inner-loop adaptation→ Outer-loop meta-objective

Core Component 2: Efficient Meta-Learning—in detail

Episodic meta-training Meta-loss → Inner-loop adaptation→ Outer-loop meta-objective

Require: Scaled dataset $\mathcal{D} = \{(x_i, c_i)\}_i$; episode builder $\textsc{BuildEpisode}$; classes per episode N; shots k; queries q; inner steps S; meta-batch size B; outer iters (epochs) E; model f_θ; step sizes α (fixed or learnable).

Ensure: Meta-initialization $(\theta^\star, \alpha^\star)$; held-out metrics.

1: Initialize θ (and α if learnable); choose outer optimizer Opt

Core Component 2: Efficient Meta-Learning—in detail

Episodic meta-training Meta-loss → Inner-loop adaptation→ Outer-loop meta-objective

Require: Scaled dataset $\mathcal{D} = \{(x_i, c_i)\}_i$; episode builder $\textsc{BuildEpisode}$; classes per episode N; shots k; queries q; inner steps S; meta-batch size B; outer iters (epochs) E; model f_θ; step sizes α (fixed or learnable).

Ensure: Meta-initialization $(\theta^\star, \alpha^\star)$; held-out metrics.

1: Initialize θ (and α if learnable); choose outer optimizer Opt
2: **for** $e = 1$ **to** E **do** ▷ outer loop
3: $\quad \mathcal{L} \leftarrow 0$
4: $\quad$ **for** $b = 1$ **to** B **do** ▷ sample a meta-batch of episodes

Core Component 2: Efficient Meta-Learning—in detail

Episodic meta-training Meta-loss → Inner-loop adaptation→ Outer-loop meta-objective

Require: Scaled dataset $\mathcal{D} = \{(x_i, c_i)\}_i$; episode builder BuildEpisode; classes per episode N; shots k; queries q; inner steps S; meta-batch size B; outer iters (epochs) E; model f_θ; step sizes α (fixed or learnable).

Ensure: Meta-initialization $(\theta^\star, \alpha^\star)$; held-out metrics.

Initialize θ (and α if learnable); choose outer optimizer Opt
for $e = 1$ **to** E **do** ▷ outer loop
    $\mathcal{L} \leftarrow 0$
    for $b = 1$ **to** B **do** ▷ sample a meta-batch of episodes
        $\tau_b \leftarrow \text{BuildEpisode}(\mathcal{D}, N, k, q)$
        $\theta_b^{(0)} \leftarrow \theta$

Core Component 2: Efficient Meta-Learning—in detail

Episodic meta-training Meta-loss → Inner-loop adaptation→ Outer-loop meta-objective

Require: Scaled dataset $\mathcal{D} = \{(x_i, c_i)\}_i$; episode builder BuildEpisode; classes per episode N; shots k; queries q; inner steps S; meta-batch size B; outer iters (epochs) E; model f_θ; step sizes α (fixed or learnable).

Ensure: Meta-initialization $(\theta^\star, \alpha^\star)$; held-out metrics.

Initialize θ (and α if learnable); choose outer optimizer Opt
for $e = 1$ **to** E **do** ▷ outer loop
  $\mathcal{L} \leftarrow 0$
  for $b = 1$ **to** B **do** ▷ sample a meta-batch of episodes
    $\tau_b \leftarrow \text{BuildEpisode}(\mathcal{D}, N, k, q)$
    $\theta_b^{(0)} \leftarrow \theta$
    for $s = 0$ **to** $S-1$ **do** ▷ inner adaptation
      $g \leftarrow \nabla_{\theta_b^{(s)}} \ell_s(\theta_b^{(s)}; \tau_b)$
      $\theta_b^{(s+1)} \leftarrow \theta_b^{(s)} - \alpha \odot g$
    end for

Core Component 2: Efficient Meta-Learning—in detail

Episodic meta-training Meta-loss → Inner-loop adaptation→ Outer-loop meta-objective

Require: Scaled dataset $\mathcal{D} = \{(x_i, c_i)\}_i$; episode builder BuildEpisode; classes per episode N; shots k; queries q; inner steps S; meta-batch size B; outer iters (epochs) E; model f_θ; step sizes α (fixed or learnable).

Ensure: Meta-initialization $(\theta^\star, \alpha^\star)$; held-out metrics.

Initialize θ (and α if learnable); choose outer optimizer Opt
for $e = 1$ **to** E **do** ▷ outer loop
  $\mathcal{L} \leftarrow 0$
  for $b = 1$ **to** B **do** ▷ sample a meta-batch of episodes
    $\tau_b \leftarrow \text{BuildEpisode}(\mathcal{D}, N, k, q)$
    $\theta_b^{(0)} \leftarrow \theta$
    for $s = 0$ **to** $S-1$ **do** ▷ inner adaptation
      $g \leftarrow \nabla_{\theta_b^{(s)}} \ell_s(\theta_b^{(s)}; \tau_b)$
      $\theta_b^{(s+1)} \leftarrow \theta_b^{(s)} - \alpha \odot g$
    end for
    $\mathcal{L} \leftarrow \mathcal{L} + \ell_q(\theta_b^{(S)}; \tau_b)$ ▷ query loss
  end for

Core Component 2: Efficient Meta-Learning—in detail

Episodic meta-training Meta-loss → Inner-loop adaptation→ Outer-loop meta-objective

Require: Scaled dataset $\mathcal{D} = \{(x_i, c_i)\}_i$; episode builder BuildEpisode; classes per episode N; shots k; queries q; inner steps S; meta-batch size B; outer iters (epochs) E; model f_θ; step sizes α (fixed or learnable).

Ensure: Meta-initialization $(\theta^\star, \alpha^\star)$; held-out metrics.

Initialize θ (and α if learnable); choose outer optimizer Opt
for $e = 1$ **to** E **do** ▷ outer loop
  $\mathcal{L} \leftarrow 0$
  for $b = 1$ **to** B **do** ▷ sample a meta-batch of episodes
    $\tau_b \leftarrow \text{BuildEpisode}(\mathcal{D}, N, k, q)$
    $\theta_b^{(0)} \leftarrow \theta$
    for $s = 0$ **to** $S-1$ **do** ▷ inner adaptation
      $g \leftarrow \nabla_{\theta_b^{(s)}} \ell_s(\theta_b^{(s)}; \tau_b)$
      $\theta_b^{(s+1)} \leftarrow \theta_b^{(s)} - \alpha \odot g$
    end for
    $\mathcal{L} \leftarrow \mathcal{L} + \ell_q(\theta_b^{(S)}; \tau_b)$ ▷ query loss
  end for
  $\mathcal{L} \leftarrow \mathcal{L}/B$
  Update $(\theta, \alpha) \leftarrow \text{Opt.step}\big(\nabla_{\theta,\alpha} \mathcal{L}\big)$

Core Component 2: Efficient Meta-Learning—in detail

Episodic meta-training Meta-loss → Inner-loop adaptation→ Outer-loop meta-objective

Require: Scaled dataset $\mathcal{D} = \{(x_i, c_i)\}_i$; episode builder BuildEpisode; classes per episode N; shots k; queries q; inner steps S; meta-batch size B; outer iters (epochs) E; model f_θ; step sizes α (fixed or learnable).

Ensure: Meta-initialization $(\theta^\star, \alpha^\star)$; held-out metrics.

Initialize θ (and α if learnable); choose outer optimizer Opt
for $e = 1$ **to** E **do** ▷ outer loop
  $\mathcal{L} \leftarrow 0$
  for $b = 1$ **to** B **do** ▷ sample a meta-batch of episodes
    $\tau_b \leftarrow \text{BuildEpisode}(\mathcal{D}, N, k, q)$
    $\theta_b^{(0)} \leftarrow \theta$
    for $s = 0$ **to** $S-1$ **do** ▷ inner adaptation
      $g \leftarrow \nabla_{\theta_b^{(s)}} \ell_s(\theta_b^{(s)}; \tau_b)$
      $\theta_b^{(s+1)} \leftarrow \theta_b^{(s)} - \alpha \odot g$
    end for
    $\mathcal{L} \leftarrow \mathcal{L} + \ell_q(\theta_b^{(S)}; \tau_b)$ ▷ query loss
  end for
  $\mathcal{L} \leftarrow \mathcal{L}/B$
  Update $(\theta, \alpha) \leftarrow \text{Opt.step}\big(\nabla_{\theta,\alpha} \mathcal{L}\big)$
end for
$(\theta^\star, \alpha^\star) \leftarrow (\theta, \alpha)$

Core Component 2: Efficient Meta-Learning—in detail

Episodic meta-training Meta-loss → Light-weight Inner-loop adaptation→ Outer-loop meta-objective

Core Component 2: Efficient Meta-Learning—in detail

Episodic meta-training Meta-loss $\rightarrow$ Light-weight Inner-loop adaptation$\rightarrow$ Outer-loop meta-objective

- **Problem.** The inner-loop adaptation of the entire model θ still incurs a significant computational cost, as gradients for all backbone parameters must be computed across S steps.

Core Component 2: Efficient Meta-Learning—in detail

Episodic meta-training Meta-loss → Light-weight Inner-loop adaptation→ Outer-loop meta-objective

- **Problem.** The inner-loop adaptation of the entire model θ still incurs a significant computational cost, as gradients for all backbone parameters must be computed across S steps.
- We partition the model parameters $\theta = (\psi, \omega)$ into
  1. heavyweight backbone parameters ψ (*e.g.*, Attention and FFN layers)
  2. lightweight adaptation parameters ω (*i.e.*, the classification head and all LayerNorm affine parameters).

Core Component 2: Efficient Meta-Learning—in detail

Episodic meta-training Meta-loss → Light-weight Inner-loop adaptation→ Outer-loop meta-objective

- **Problem.** The inner-loop adaptation of the entire model θ still incurs a significant computational cost, as gradients for all backbone parameters must be computed across S steps.
- We partition the model parameters $\theta = (\psi, \omega)$ into
  1. heavyweight backbone parameters ψ (*e.g.*, Attention and FFN layers)
  2. lightweight adaptation parameters ω (*i.e.*, the classification head and all LayerNorm affine parameters).

We freeze ψ in the inner loop and adapt only ω:

$$\omega^{(t+1)} = \omega^{(t)} - \alpha_\omega \odot \nabla_{\omega^{(t)}} \ell_s\big((\psi, \omega^{(t)}); \tau\big), \tag{5}$$

$$\psi^{(t+1)} = \psi^{(t)} \equiv \psi. \tag{6}$$

Core Component 2: Efficient Meta-Learning—in detail

Episodic meta-training Meta-loss → Light-weight Inner-loop adaptation→ Outer-loop meta-objective

- **Problem.** The inner-loop adaptation of the entire model θ still incurs a significant computational cost, as gradients for all backbone parameters must be computed across S steps.
- We partition the model parameters $\theta = (\psi, \omega)$ into
  1. heavyweight backbone parameters ψ (*e.g.*, Attention and FFN layers)
  2. lightweight adaptation parameters ω (*i.e.*, the classification head and all LayerNorm affine parameters).

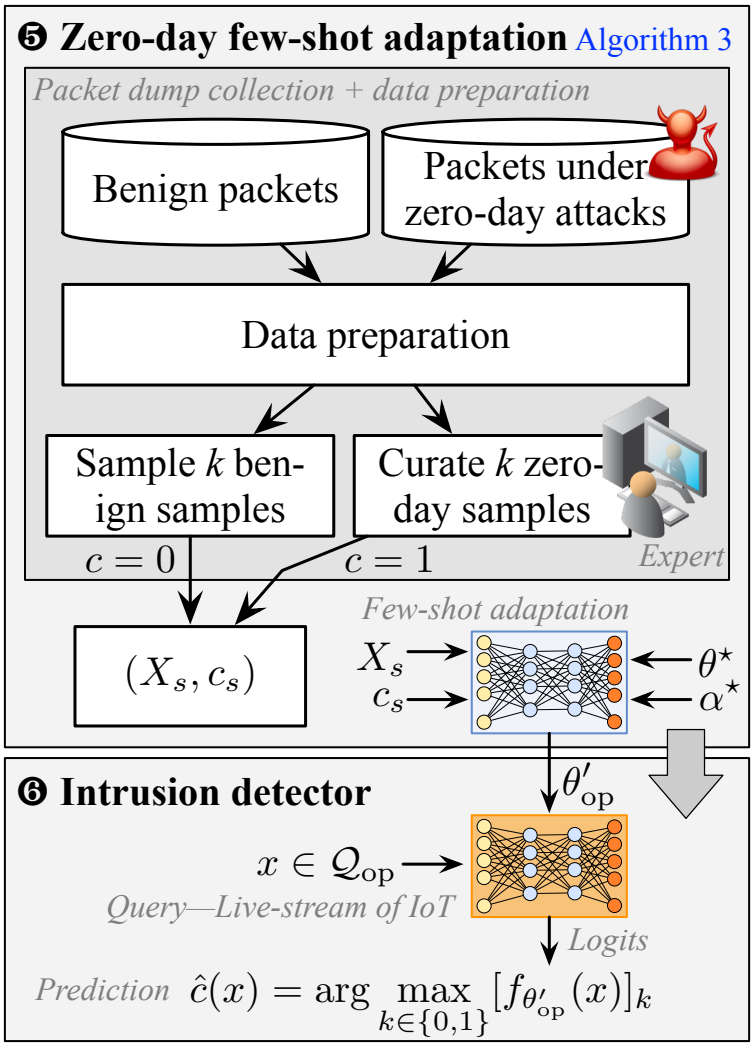
We freeze ψ in the inner loop and adapt only ω:

$$\omega^{(t+1)} = \omega^{(t)} - \alpha_\omega \odot \nabla_{\omega^{(t)}} \ell_s\big((\psi, \omega^{(t)}); \tau\big), \tag{5}$$

$$\psi^{(t+1)} = \psi^{(t)} \equiv \psi. \tag{6}$$

Note that the outer gradient still updates both ψ and ω (and α_ω) using the query loss evaluated at the adapted parameters.

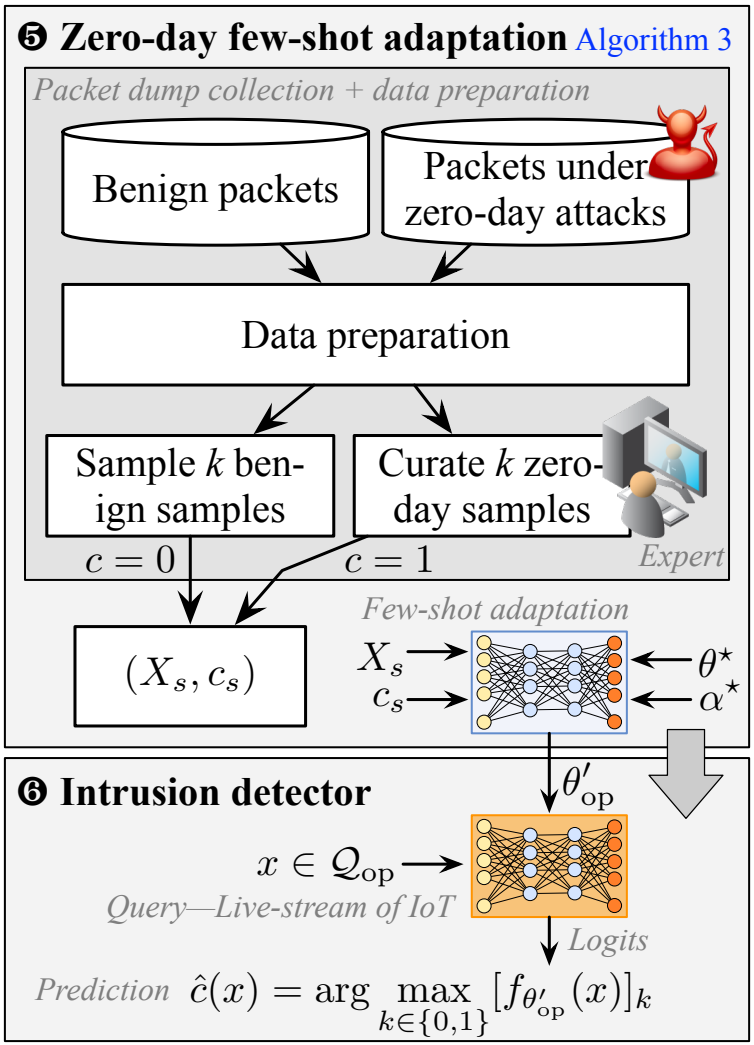
Operational episode



We target binary intrusion detection in deployment; hence we set $N = 2$.

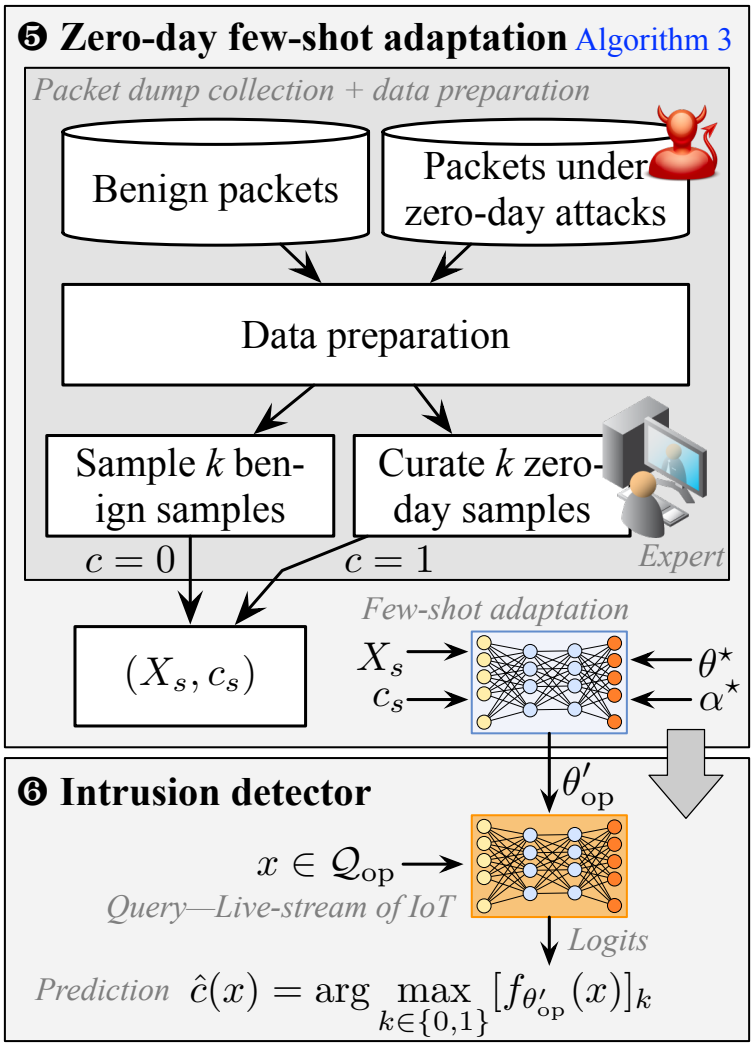
Core Component 3: Intrusion detector

Operational episode



We target binary intrusion detection in deployment; hence we set $N = 2$. With episodic labels $\{0, 1\}$ corresponding to *benign* (anchor, 0) and *attack* (1).

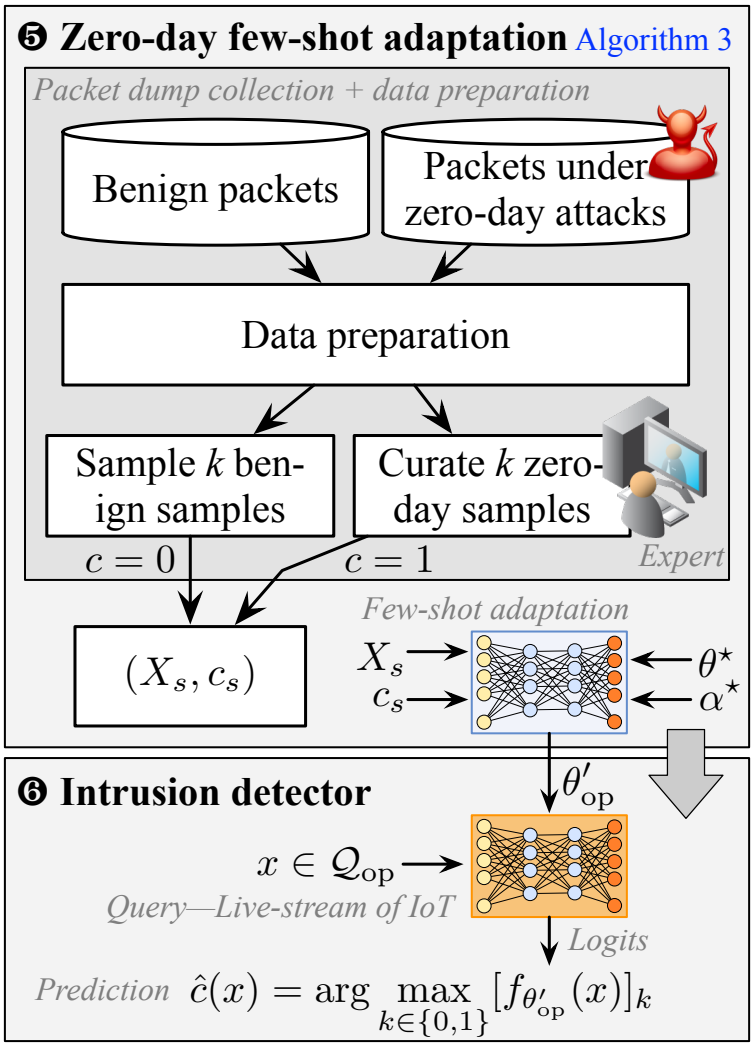
Operational episode



We target binary intrusion detection in deployment; hence we set $N = 2$. With episodic labels $\{0, 1\}$ corresponding to *benign* (anchor, 0) and *attack* (1). Let $\tau_{\text{op}} = (X_s, c_s, \mathcal{Q}_{\text{op}})$ denote an operational episode, where,

- X_s and c_s are on-site labeled support pairs
- $\mathcal{Q}_{\text{op}}$ is a batch or live stream of unlabeled packets to be classified.

Operational episode



Require: Meta-initialization $(\theta^\star, \alpha^\star)$; operational episode $\tau_{\text{op}} = (X_s, c_s, \mathcal{Q}_{\text{op}})$ with labels $\{0$: benign, 1: attack$\}$; inner steps S.

Ensure: Predictions $\hat{c}(x) \in \{0, 1\}$ for $x \in \mathcal{Q}_{\text{op}}$ (and scores if desired).

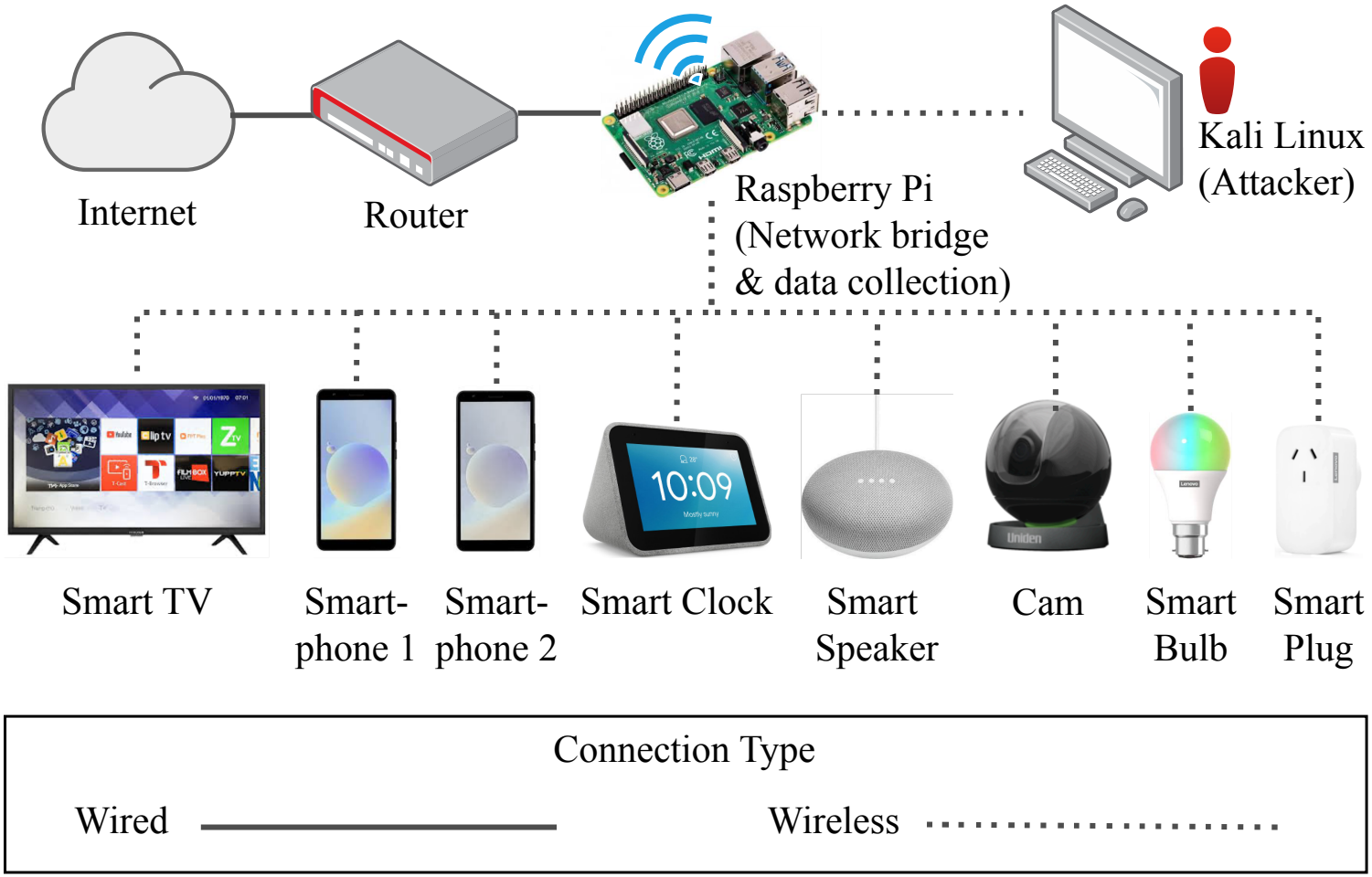
1: $\theta'_{\text{op}} \leftarrow$ apply S inner steps to $(\theta^\star, \alpha^\star)$ on (X_s, c_s)
2: **for** each $x \in \mathcal{Q}_{\text{op}}$ **do**
3: $\quad \hat{c}(x) = \arg\max_{k\in\{0,1\}} [f_{\theta'_{\text{op}}}(x)]_k$
4: **end for**

Experimental Setup

- **Dataset:** UQ-IoT-IDS-2021

Experimental Setup

- **Dataset:** UQ-IoT-IDS-2021



Experimental Setup

- **Dataset:** UQ-IoT-IDS-2021
  - 41,404,983 packets in total.
  - 17,422,556 (≈42%) benign packets.
  - 23,982,427 (≈58%) attack packets.

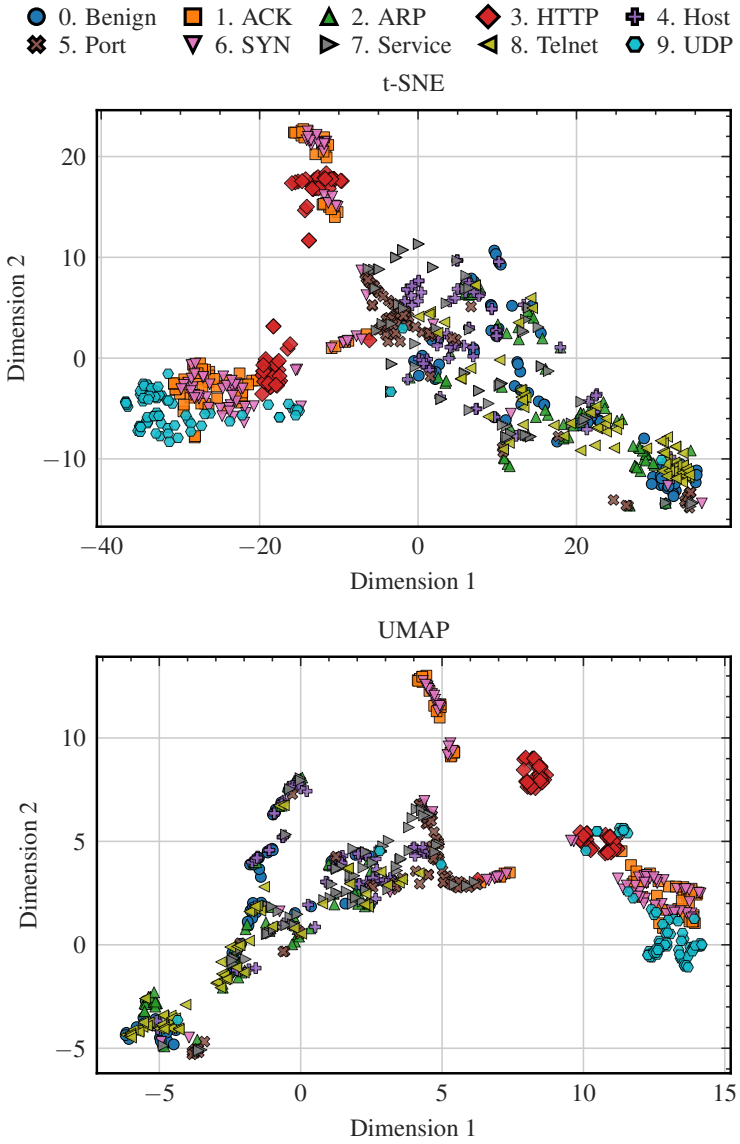
Table 1
Overview of the UQ-IoT-IDS-2021 dataset. The total of 41,404,983 packets are available for 10 classes.

Class (Attack type)	#packet dumps	#packets	Duration	#packets/min	#uniq. MAC addr.	#uniq. IP addr.	#uniq. flows
0. Benign	1	17,422,556	7 days, 3:00:01	1,698.10	43	12,134	1,170,975
1. ACK Flooding	8	8,910,000	0:19:02	70,632.41	27	490	520,482
2. ARP Spoofing	8	15,302	0:17:24	538.32	27	622	2,149
3. HTTP Flooding	1	683,408	0:02:07	323,385.98	13	31	14,143
4. Host Discovery	2	27,644	0:17:31	0.55	29	488	2,464
5. Port Scanning	8	41,624	0:04:39	202.26	22	323	17,066
6. SYN Flooding	8	5,521,169	0:16:43	43,333.23	27	449	940,632
7. Service Detection	8	92,644	0:26:23	430.13	25	662	21,803
8. Telnet brute force	2	129,072	0:28:46	1.07	17	501	1,119
9. UDP Flooding	8	8,561,564	0:19:36	494.88	22	488	524,161

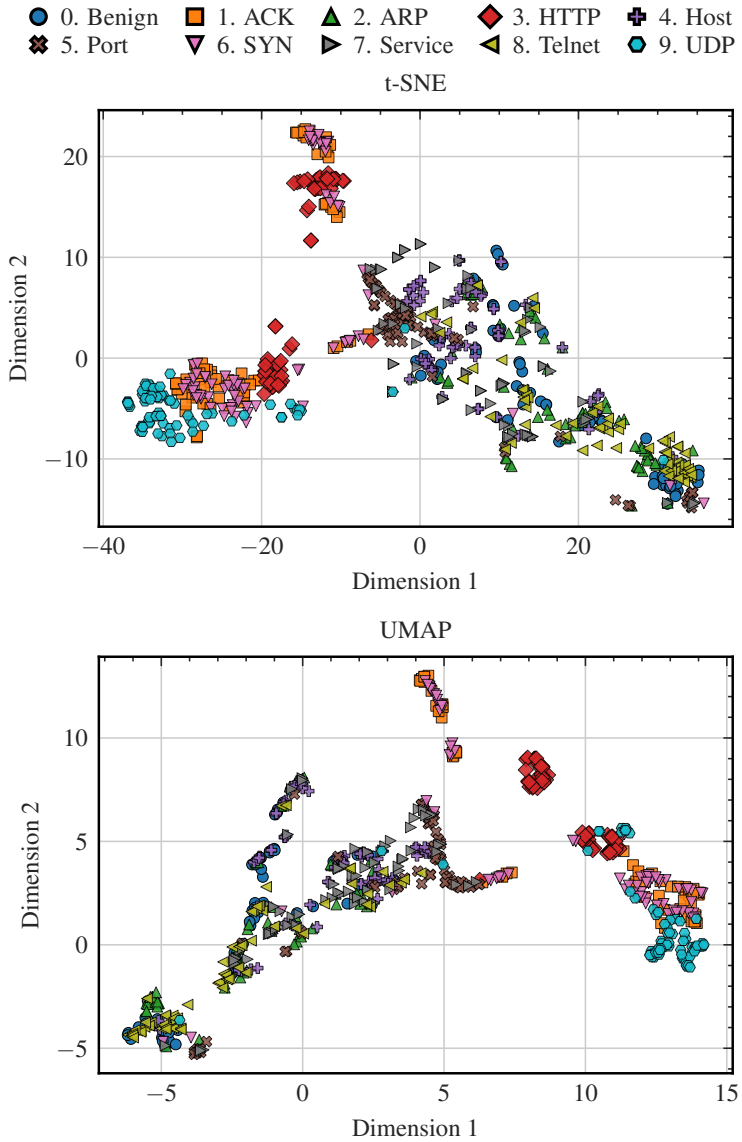
Experimental Setup

- **Dataset:** UQ-IoT-IDS-2021
  - A realistic, large-scale dataset collected from heterogeneous IoT devices (smartphones, cameras, smart speakers, etc.).
  - 10 classes total (9 attack families + 1 benign).
- **Features:** Kitsune AfterImage
  - Extracts a 100-dimensional feature vector per packet, capturing time-decayed statistics.
- **Zero-Day Protocol:** Leave-One-Attack-Out (LOAO)
  - We train the meta-learner on 8 attack families.
  - We test its few-shot adaptation ability on the 9th, unseen attack family.
  - This protocol is repeated for all 9 attacks.
- **Task:** k-shot binary classification (Benign vs. Zero-Day Attack). We focus on $k = 5$ shots.

Experimental Setup—Dataset overview

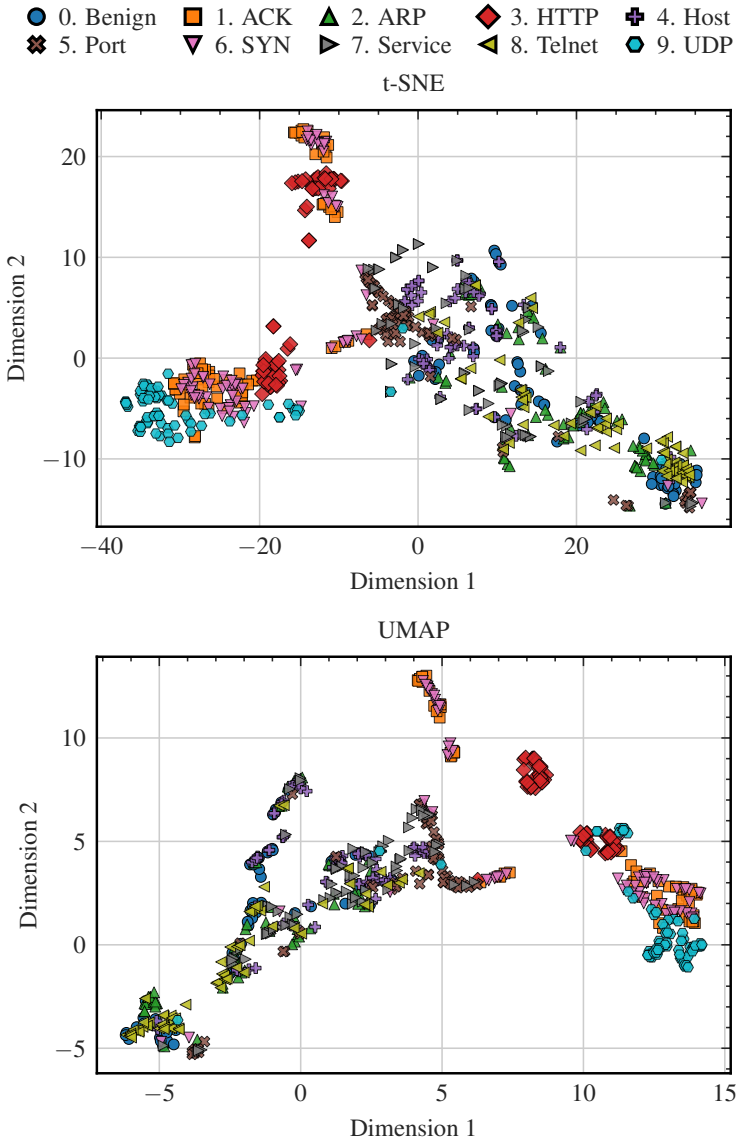


Experimental Setup—Dataset overview



- Host Discovery (Class 4) and Telnet Brute-force (Class 8) overlap heavily with Benign (Class 0).

Experimental Setup—Dataset overview



- Host Discovery (Class 4) and Telnet Brute-force (Class 8) overlap heavily with Benign (Class 0).
- Their proximity signals high distributional similarity, likely making these attacks harder to separate with few labels.

Results: Performance vs. Baselines (Avg. F1-Score)

We compared MetaFTT ($k = 5$) against 7 baselinesp—see Appendix 2.

Table: Average F1-Score across all 9 zero-day scenarios.

Method	**Average F1-Score**
Kitsune (Unsupervised)	0.6360
FT-Transformer (Fine-tuning)	0.8735
FC-Net (2020)	0.8631
FS-IDS (2022)	0.8705
MASiNet (2024)	0.8380
PTN-IDS (2024)	0.8385
MACML (2025)	0.8019
MetaFTT (Ours)	**0.9240**

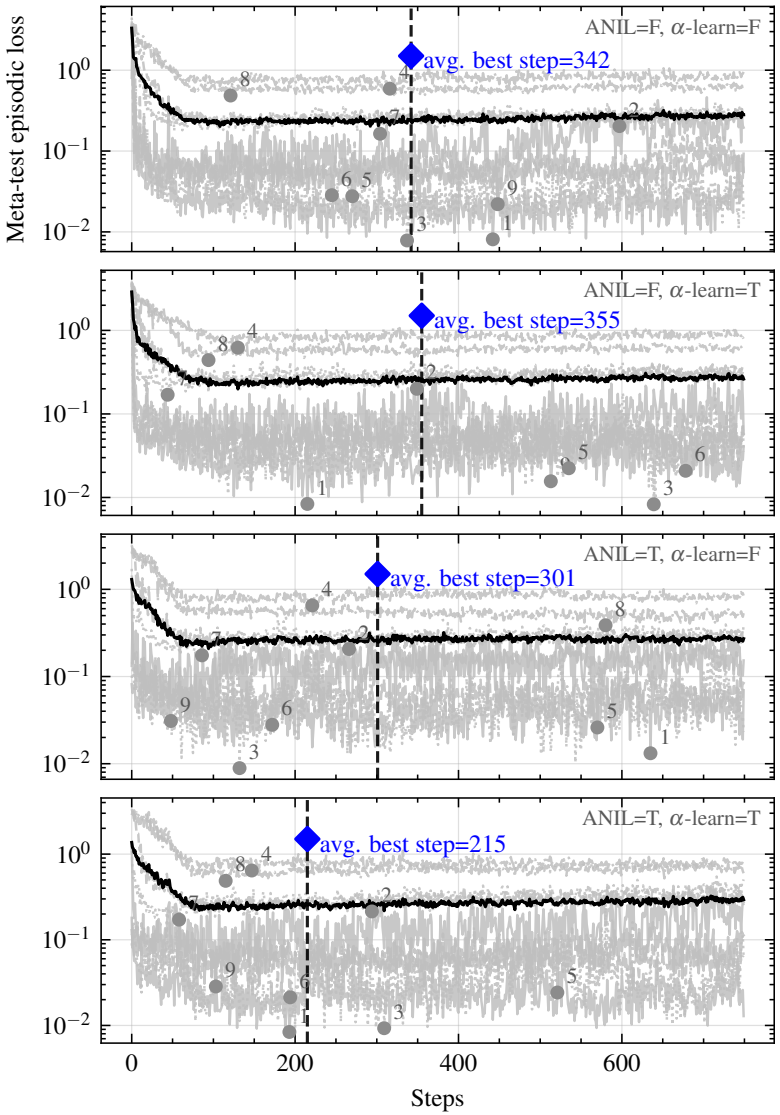
Key Takeaway: MetaFTT achieves the highest average F1-score, significantly outperforming standard fine-tuning and all five state-of-the-art few-shot NIDS baselines.

Results: Ablation Study

Question: Does our lightweight adaptation strategy work?

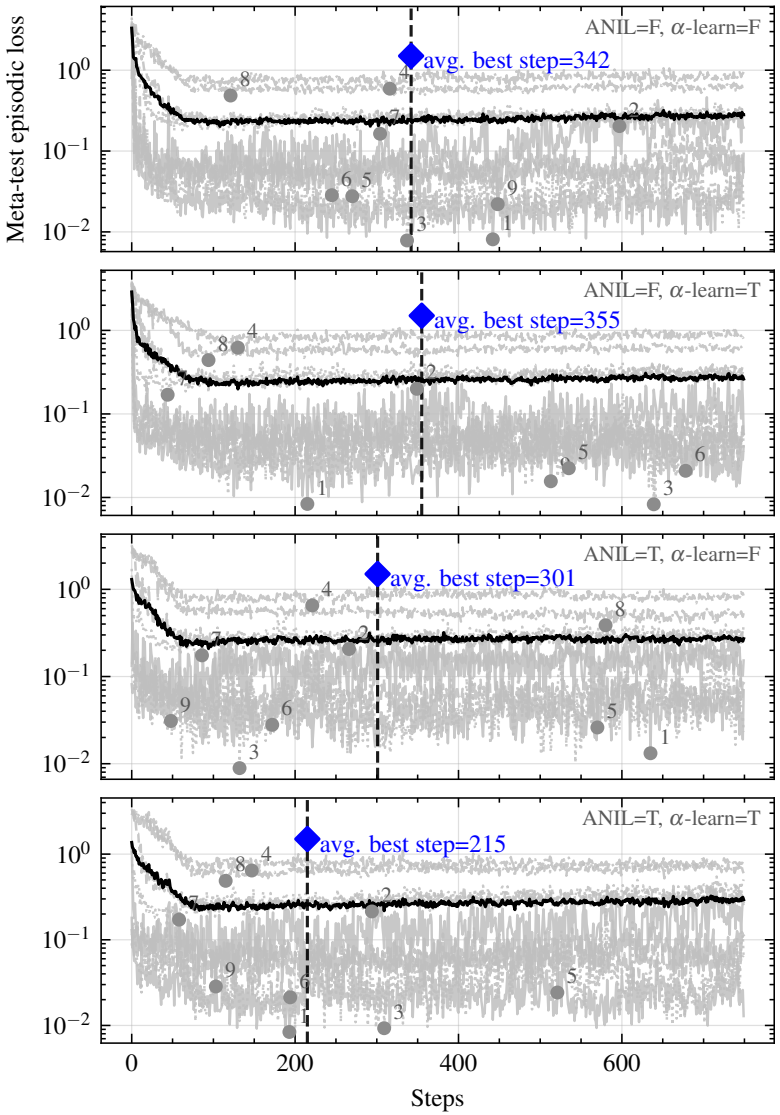
Results: Ablation Study

Question: Does our lightweight adaptation strategy work?



Results: Ablation Study

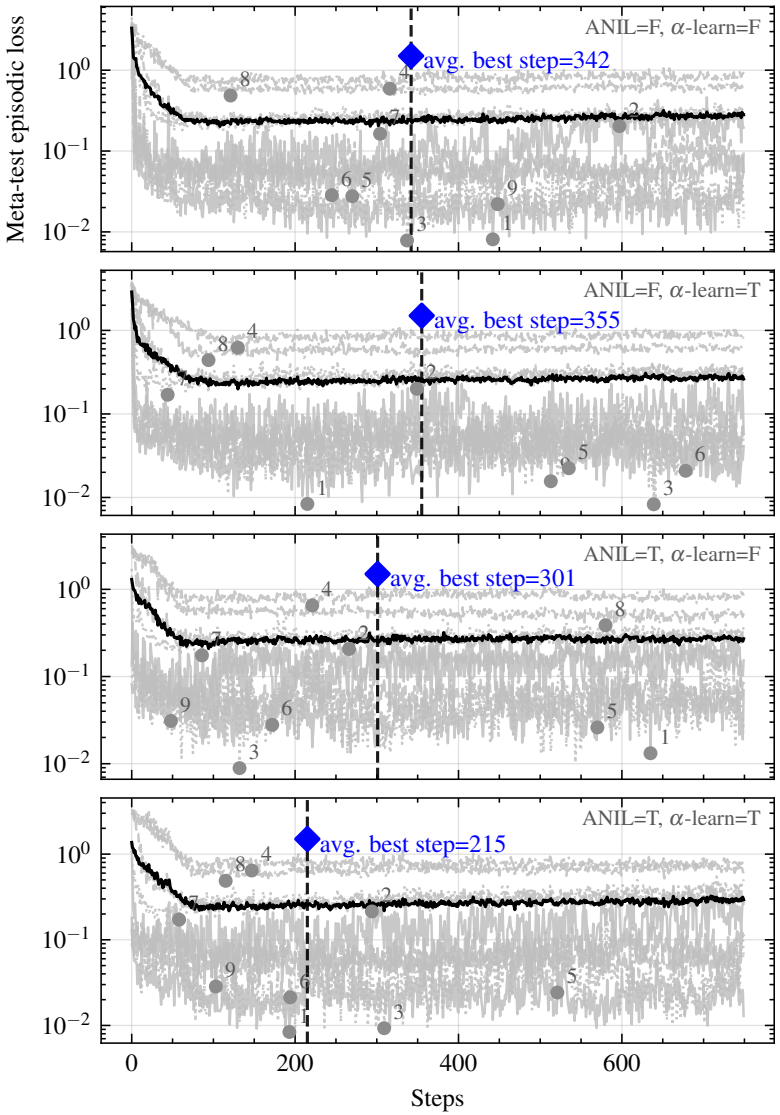
Question: Does our lightweight adaptation strategy work?



- We compared 4 configurations (Full vs. Lightweight, Fixed vs. Learnable inner LR).

Results: Ablation Study

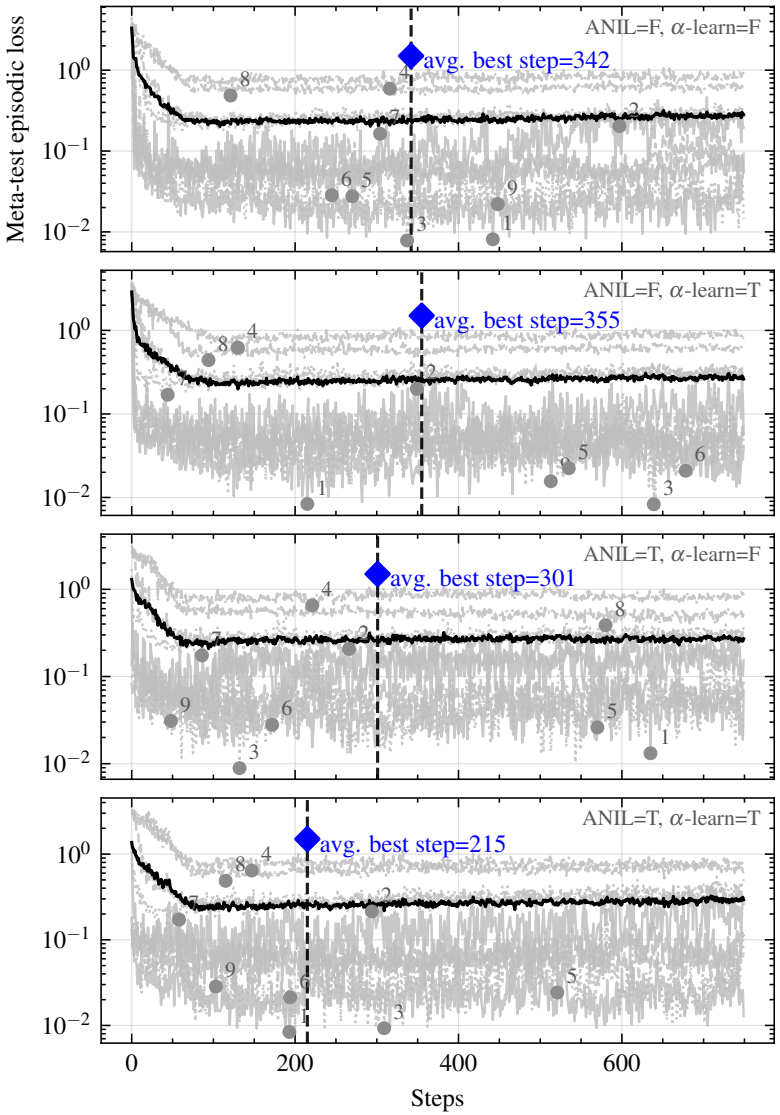
Question: Does our lightweight adaptation strategy work?



- We compared 4 configurations (Full vs. Lightweight, Fixed vs. Learnable inner LR).
- **Performance:** All configurations achieved comparable F1-scores (0.924–0.930)—see Table 2.

Results: Ablation Study

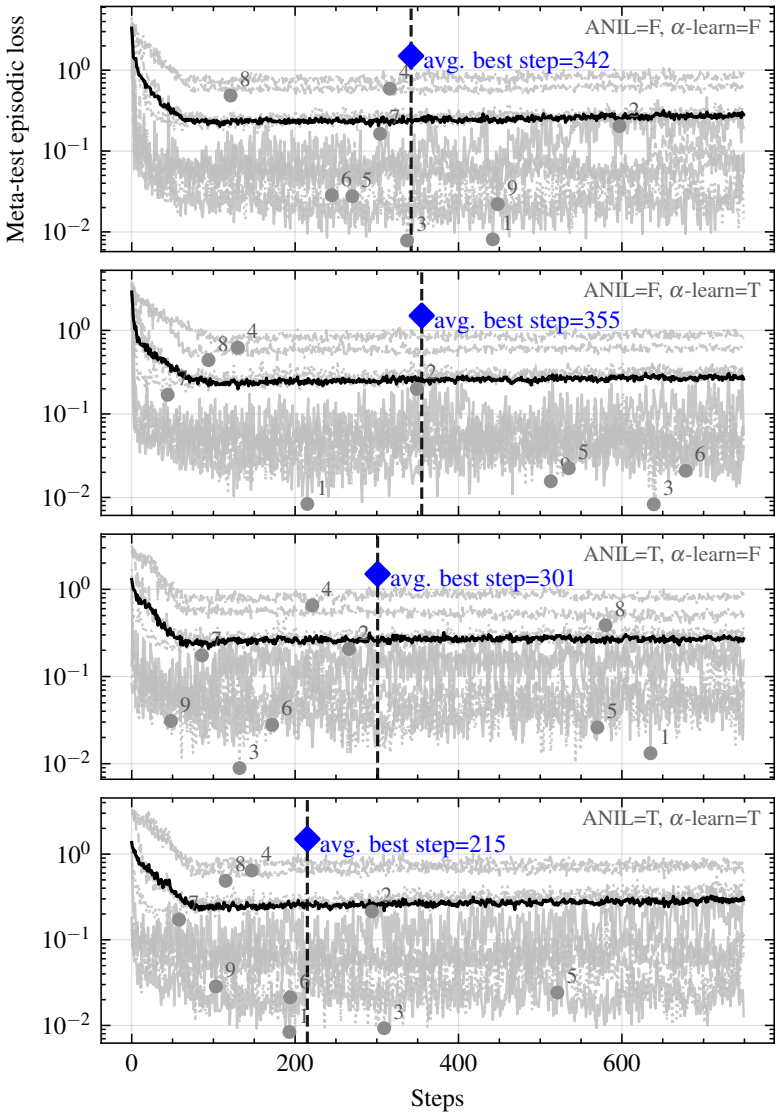
Question: Does our lightweight adaptation strategy work?



- We compared 4 configurations (Full vs. Lightweight, Fixed vs. Learnable inner LR).
- **Performance:** All configurations achieved comparable F1-scores (0.924–0.930)—see Table 2.
- **Findings:** Our proposed method (bottom panel) converges **significantly faster** (avg. best step 215) and to a lower loss.

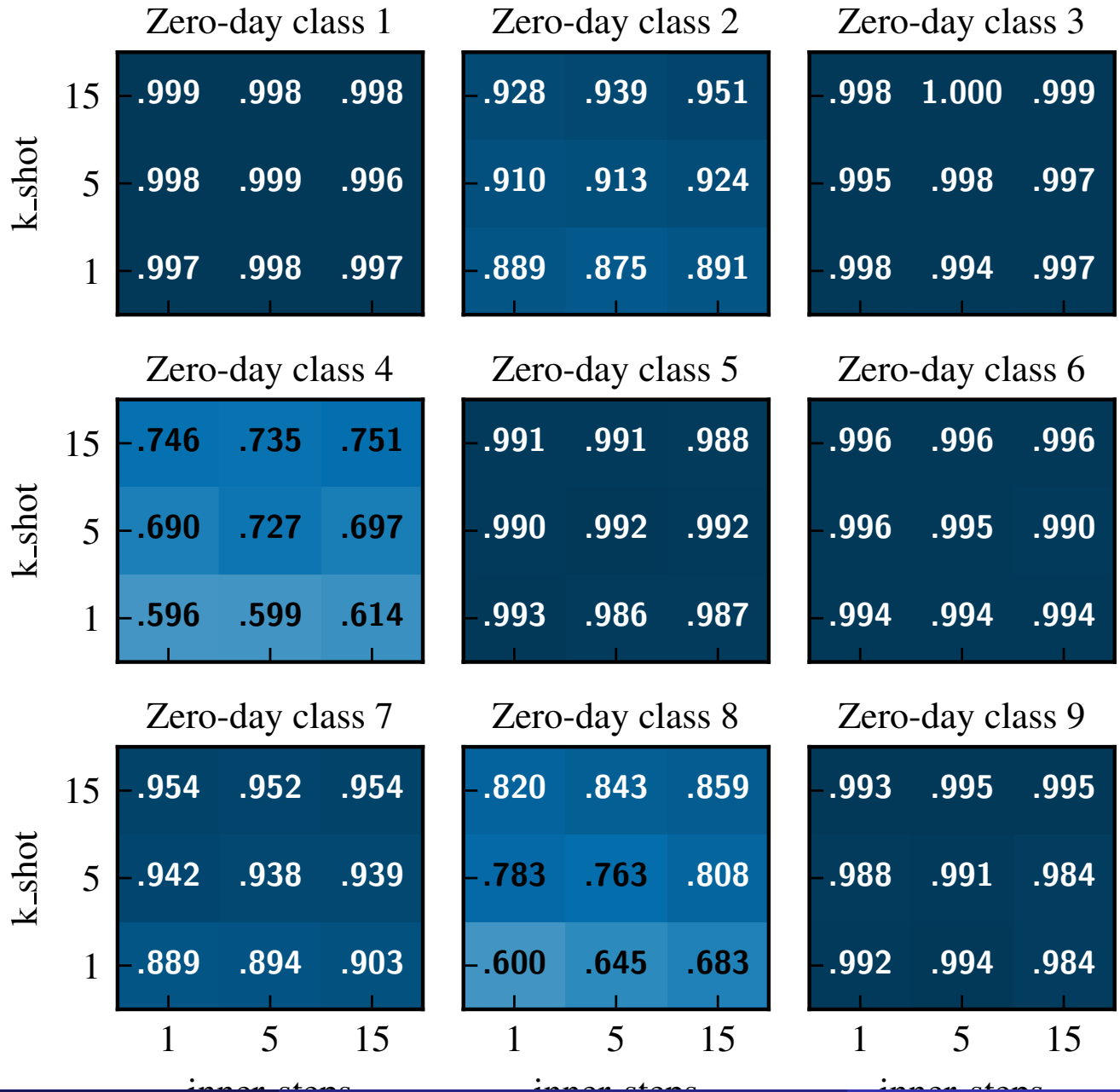
Results: Ablation Study

Question: Does our lightweight adaptation strategy work?



- We compared 4 configurations (Full vs. Lightweight, Fixed vs. Learnable inner LR).
- **Performance:** All configurations achieved comparable F1-scores (0.924–0.930)—see Table 2.
- **Findings:** Our proposed method (bottom panel) converges **significantly faster** (avg. best step 215) and to a lower loss.
- **Conclusion:** Our lightweight strategy (ANIL=T, α-learn=T) maintains high detection performance.

Results: Sensitivity to k (Shots) and S (Steps)



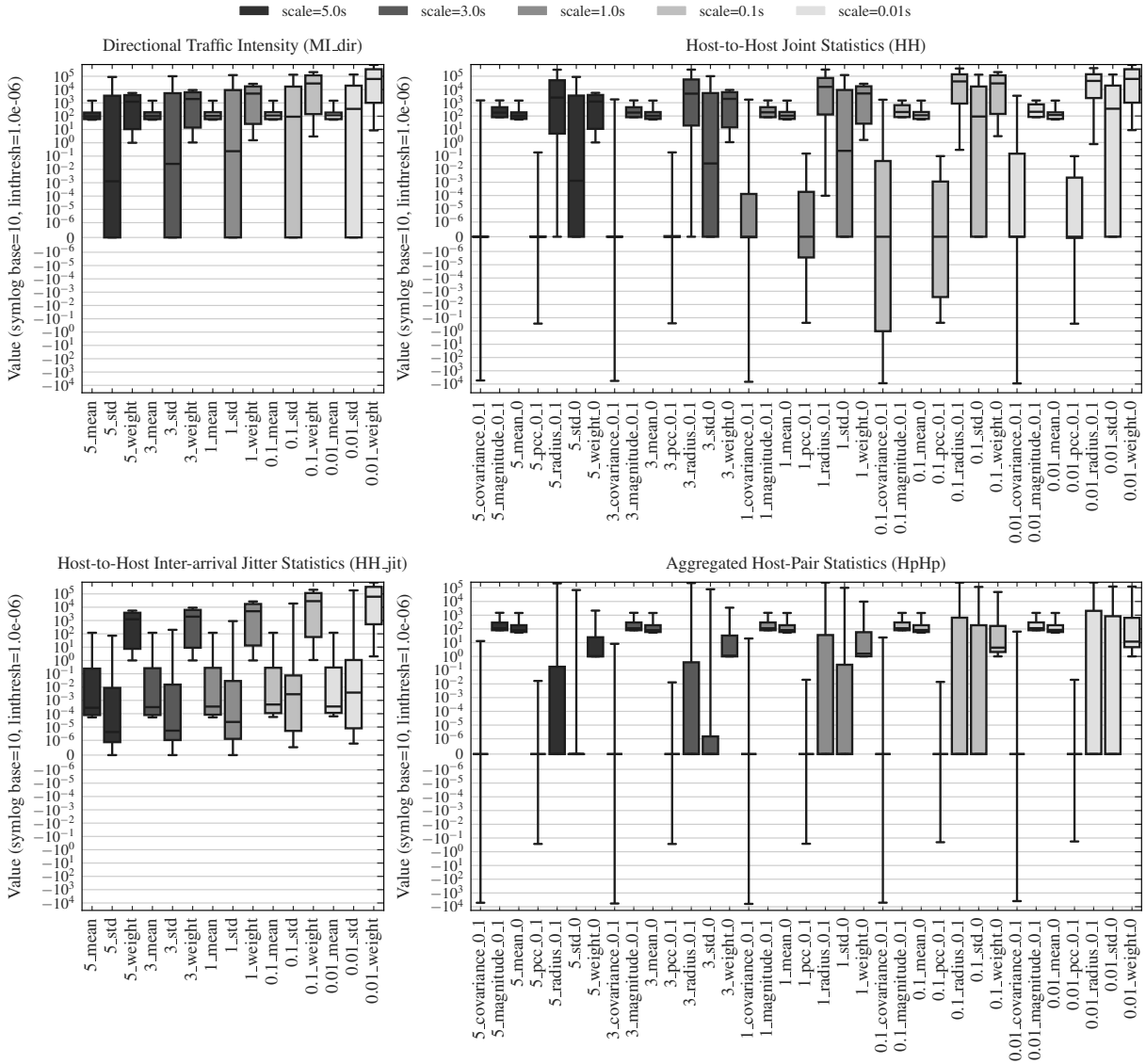
Conclusion

- **We proposed MetaFTT**, a novel framework that integrates an FT-Transformer with optimization-based meta-learning.
- **Key Contribution 1:** We use an architecture (FT-Transformer) that is natively suited for tabular network traffic, avoiding the CNN "image" mismatch.
- **Key Contribution 2:** We introduced a **computationally efficient** meta-learning strategy (first-order approx. + lightweight inner adaptation) that makes this high-capacity model practical.
- **Result:** MetaFTT achieves state-of-the-art performance for few-shot, zero-day attack detection on the realistic UQ-IoT-IDS-2021 dataset.
- **Impact:** This provides a viable, scalable, and effective solution for real-world IoT security.

Thank You

Questions?

Appendix 1. Grouped boxplots of the 100 Kitsune AfterImage features by semantic family—Figure 4.



Appendix 2.—Performance comparison between existing methods using the same dataset. The bold numbers represent the best F1-score per zero-day clas

Zero-day	Fine-tuning of FT-Transformer				Kitsune (2018)				FC-Net (2020)				FS-IDS (2022)			
class	Acc.	Prec.	Recall	F1	Acc.	Prec.	Recall	F1	Acc.	Prec.	Recall	F1	Acc.	Prec.	Recall	F1
1	0.9963	1.0000	0.9927	0.9963	0.9838	0.9980	0.9826	0.9903	0.9850	0.9727	0.9980	0.9852	0.9917	0.9867	0.9968	0.9917
2	0.8310	0.9733	0.6807	0.8011	0.9850	0.2656	0.4114	0.3228	0.7994	0.8536	0.7227	0.7827	0.7041	0.7155	0.6774	0.6960
3	0.9830	0.9925	0.9733	0.9828	0.9889	0.9748	0.9862	0.9805	0.9382	0.8900	1.0000	0.9418	0.9520	0.9126	0.9996	0.9541
4	0.6637	0.7163	0.5420	0.6171	0.9777	0.1806	0.1488	0.1632	0.6432	0.7585	0.4202	0.5408	0.6668	0.7512	0.4989	0.5996
5	0.9547	0.9942	0.9147	0.9528	0.9802	0.5775	0.5717	0.5746	0.9825	0.9859	0.9790	0.9825	0.9698	0.9750	0.9643	0.9696
6	0.9953	0.9993	0.9913	0.9953	0.9818	0.9968	0.9792	0.9879	0.9815	0.9752	0.9882	0.9817	0.9904	0.9826	0.9984	0.9905
7	0.8580	0.9727	0.7367	0.8384	0.9687	0.7511	0.5671	0.6463	0.8736	0.9692	0.7717	0.8592	0.8957	0.9877	0.8014	0.8848
8	0.7293	0.7991	0.6127	0.6936	0.9246	0.2211	0.0384	0.0654	0.8038	0.8428	0.7469	0.7919	0.8182	0.8611	0.7589	**0.8067**
9	0.9840	0.9946	0.9733	0.9838	0.9880	0.9979	0.9875	**0.9927**	0.8967	0.8589	0.9494	0.9019	0.9410	0.9333	0.9499	0.9415
Average	0.8884	0.9380	0.8242	0.8735	0.9754	0.6626	0.6303	0.6360	0.8782	0.9008	0.8418	0.8631	0.8811	0.9006	0.8495	0.8705
Zero-day	MASiNet (2024)				PTN-IDS (2024)				MACML (2025)				**Proposed MetaFTT** ($k = 5$)			
class	Acc.	Prec.	Recall	F1	Acc.	Prec.	Recall	F1	Acc.	Prec.	Recall	F1	Acc.	Prec.	Recall	F1
1	0.9826	0.9920	0.9730	0.9824	0.9818	0.9727	0.9914	0.9820	0.9887	0.9804	0.9974	0.9888	0.9990	0.9995	0.9984	**0.9990**
2	0.8911	0.9335	0.8423	0.8855	0.7508	0.7187	0.8240	0.7678	0.8748	0.8451	0.5990	0.7011	0.9146	0.9261	0.9010	**0.9134**
3	0.9910	0.9911	0.9908	0.9910	0.9558	0.9300	0.9858	0.9571	0.9940	0.9891	0.9990	0.9940	0.9982	0.9990	0.9974	**0.9982**
4	0.6220	0.7254	0.3927	0.5095	0.5132	0.5169	0.4045	0.4538	0.8189	0.8445	0.2531	0.3894	0.7346	0.7490	0.7057	**0.7267**
5	0.9278	0.9819	0.8717	0.9236	0.9293	0.9491	0.9073	0.9278	0.9831	0.9854	0.9807	0.9830	0.9924	0.9881	0.9969	**0.9925**
6	0.9935	0.9948	0.9922	0.9935	0.9805	0.9948	0.9661	0.9802	0.9910	0.9833	0.9989	0.9910	0.9945	0.9995	0.9896	**0.9945**
7	0.8203	0.9614	0.6673	0.7878	0.8935	0.8647	0.9329	0.8975	0.9442	0.9390	0.9502	**0.9446**	0.9380	0.9362	0.9401	0.9381
8	0.5493	0.5671	0.4169	0.4805	0.6676	0.7217	0.5456	0.6214	0.8907	0.4532	0.4421	0.4476	0.7549	0.7391	0.7880	0.7628
9	0.9882	0.9858	0.9907	0.9882	0.9601	0.9966	0.9233	0.9586	0.8130	0.9572	0.6553	0.7780	0.9911	0.9995	0.9828	0.9911
Average	0.8629	0.9037	0.7931	0.8380	0.8481	0.8517	0.8312	0.8385	0.9220	0.8864	0.7640	0.8019	0.9241	0.9262	0.9222	**0.9240**