# Federated Test-Time Adaptation without Waiting: Asynchronous Collaboration in Dynamic Environments

**Anonymous authors**
Paper under double-blind review

## Abstract

Federated Learning (FL) supports privacy-preserving learning by enabling clients to train models collaboratively without sharing raw data. However, deployed models may encounter unseen and dynamically changing test distributions. Test-Time Adaptation (TTA) mitigates this distribution shift by adapting models online using unlabeled test data. Federated Test-Time Adaptation (FedTTA) extends TTA to FL by allowing clients to share adaptation knowledge. Most existing FedTTA methods rely on synchronous collaboration. They update shared information until all participating clients complete local adaptation, forcing the system to wait for stragglers. To circumvent this limitation, we consider asynchronous FedTTA without synchronization waiting. However, asynchronous updates may cause fast clients to dominate the shared state, while erroneous updates can directly affect subsequent clients. To address these problems, we propose an asynchronous FedTTA method for dynamic environments. Our method uses a dynamic shared memory to reduce the cumulative influence of repeated updates from fast clients and isolates shared knowledge from local TTA to limit error propagation. The local adaptation process remains unchanged, making our method compatible with most existing TTA methods. Across the evaluated datasets and architectures, our method reduces measured wall-clock time by 44.6%–64.2% compared with synchronous FedTTA.

## 1 Introduction

Federated learning (FL) enables multiple clients to jointly train a global model without sharing raw data (Li et al., 2023). Federated training relies on data distributions observed during training. However, client environments may continue to change after deployment. For instance, changes in weather, lighting, device conditions, or user behavior can cause incoming test data to gradually deviate from the training distribution, degrading the performance of an otherwise effective global model (Jiang & Lin, 2023; Tan et al., 2023).

Test-Time Adaptation (TTA) has emerged as an effective approach to mitigating distribution shifts during deployment. TTA adapts a model using unlabeled test data received during inference, allowing it to adjust to unknown target distributions without access to source training data or target labels (Wang et al., 2021). Yet, independent TTA relies solely on local test data. It cannot benefit from knowledge acquired by other clients and is more susceptible to local overfitting and error accumulation (Niu et al., 2023). To address this issue, recent work extends federated collaboration to the test phase, allowing clients to share knowledge acquired during local adaptation (Zhang et al., 2024; Bao et al., 2025). Following Zhang et al. (2026), we refer to methods that combine FL and TTA as Federated Test-Time Adaptation (FedTTA).

Although FedTTA extends collaboration across clients to the test phase, clients in practical deployments complete local adaptation at different speeds. Differences in computational resources, communication conditions, and current workloads lead to substantial variation in the time required for local TTA (Smith et al., 2017). As illustrated in Fig. 1, existing



Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

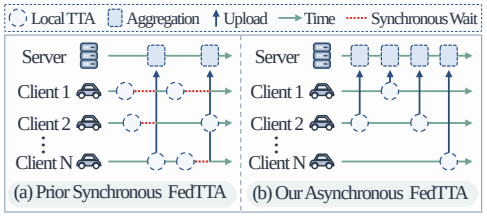


Figure 1: Comparison of synchronous and asynchronous FedTTA. (a) Synchronous methods wait for clients to complete local TTA before aggregating their updates. (b) Our method processes each update as it arrives, without waiting for synchronization.

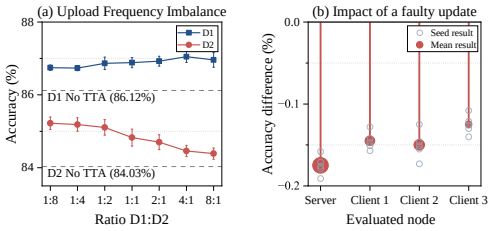


Figure 2: (a) Test accuracy on two target domains, D1 and D2, under different upload frequency ratios. Dashed lines indicate the baseline without TTA. (b) Accuracy differences between a single erroneous update and a normal update, measured at the server and other clients across five random seeds, with the mean also shown.

FedTTA methods typically use synchronous collaboration. The server waits for multiple clients to complete local TTA before updating the shared model. Slow clients can therefore limit responsiveness, making it difficult for the shared model to keep pace with changing target environments. To eliminate this waiting, we consider an asynchronous FedTTA setting in which clients upload updates immediately after completing local TTA. Namely, the server processes each update as it arrives, allowing the shared model to evolve continuously without waiting for other clients.

This asynchronous update scheme avoids delays caused by slow clients but introduces new challenges. Faster clients produce more updates within the same time interval, so their target knowledge is incorporated into the shared model more frequently. When clients encounter different target distributions, this imbalance can gradually bias the global model toward faster clients. As shown in Fig. 2 (a), increasing the upload frequency of clients from one target domain improves the shared model's performance on that domain while reducing its performance on another. This result indicates that client processing speed directly influences the direction of global adaptation.

Beyond the bias caused by unequal update frequencies, asynchronous updates can amplify the unreliability of local TTA. Without target labels, local TTA relies on the model's own predictions to guide adaptation. Incorrect predictions can thus be repeatedly reinforced, producing harmful local updates (Wang et al., 2022). Incorporating these unreliable updates can directly degrade the global model and erode previously acquired knowledge (Wang et al., 2026). In synchronous FedTTA, updates from multiple clients are typically aggregated together, which can partially dilute the influence of an individual harmful update. In the asynchronous setting considered here, each arriving update immediately affects the shared model. A single erroneous local adaptation step can therefore directly alter the model received by subsequent clients. As shown in Fig. 2 (b), a single erroneous update degrades the shared model on the server, and its adverse effects persist in clients that adapt afterward.

To address the above challenges, we propose a without-waiting FedTTA method for dynamic environments. Our method maintains a dynamic shared memory bank and immediately processes the adaptation knowledge uploaded by each client after local adaptation. Instead of directly accumulating client updates, the server writes only the residual information in the current adaptation knowledge that has not yet been absorbed by the shared memory. As similar knowledge arrives repeatedly, its contribution gradually diminishes, thereby mitigating the shared-model bias caused by differences in client update frequencies. Meanwhile, the shared memory is used only for test-time prediction correction and does not participate in the initialization or optimization of local TTA, thereby isolating local adaptation from global knowledge sharing and reducing the influence of erroneous shared knowledge on subsequent local adaptation. Our method does not modify the original local TTA procedure and is therefore compatible with a variety of existing TTA methods. Experimental results demonstrate that our method consistently improves performance across multiple datasets and asynchronous test scenarios.

Our main contributions are summarized as follows:

- We investigate without-waiting asynchronous FedTTA and identify two key issues: fast client dominance caused by heterogeneous client speeds and rapid error propagation from unreliable local adaptation.
- To address fast client dominance, we propose a dynamic shared memory with residual write-in, which reduces the repeated influence of frequent updates and mitigates shared-state bias.
- To address error propagation, we isolate shared knowledge from local TTA and use it only for prediction correction, reducing error propagation while remaining compatible with most local TTA methods.
- Extensive experiments validate the effectiveness and efficiency of our method, demonstrating runtime reductions of 44.6%–64.2% compared with synchronous FedTTA.

## 2 Related Work

### 2.1 Test-Time Adaptation

TTA adapts a pretrained model using unlabeled test data to mitigate distribution shifts during deployment. Early studies mainly focused on efficient adaptation. Tent (Wang et al., 2021) minimizes prediction entropy by updating normalization-layer affine parameters, while T3A (Iwasawa & Matsuo, 2021) adjusts the classifier using test-time features. Later, EATA (Niu et al., 2022), SAR (Niu et al., 2023), and DeYO (Lee et al., 2024) improved adaptation reliability through sample selection, stable optimization, and reliability assessment, while FRET (You et al., 2025) reduces feature redundancy and enhances discriminability. For continuously changing test streams, CoTTA (Wang et al., 2022) and NOTE (Gong et al., 2022) mitigate error accumulation, forgetting, and temporal bias, while GOLD (Lai et al., 2026) balances efficiency and generalization through lightweight low-rank updates. However, these methods are primarily designed for single-client adaptation and do not leverage adaptation knowledge from other clients.

### 2.2 Asynchronous Federated Learning

Asynchronous FL reduces synchronization waiting for slow clients and improves training efficiency in heterogeneous environments. FedBuff (Nguyen et al., 2022) buffers asynchronous updates to balance efficiency and aggregation stability, FedFix (Fraboni et al., 2023) adopts time-triggered aggregation, and FedCompass (Li et al., 2024) adjusts local computation according to client capabilities to alleviate update staleness caused by system heterogeneity. Recent studies have further focused on optimization bias in asynchronous training. FedCure (Chen et al., 2026) mitigates participation bias under Non-IID data through participation-aware scheduling, while AFLTuning (Pang et al., 2026) improves participation imbalance through frequency-aware aggregation. These methods are mainly designed for supervised federated training. In contrast, asynchronous FedTTA lacks ground-truth supervision for assessing local update reliability, making erroneous adaptations more likely to persist and rapidly affect subsequent clients.

### 2.3 Federated Test-Time Adaptation

FedTTA leverages federated collaboration to improve label-free adaptation to unseen target distributions. Early studies mainly exploited global or personalized knowledge learned during federated training to enhance local adaptation. For example, FedTHE (Jiang & Lin, 2023) combines global and local classifiers, while FedSPL (Liang et al., 2025) adopts teacher–student adaptation. Subsequently, FedTSA (Zhang et al., 2024), FedCTTA (Rajib et al., 2025), and Latte (Bao et al., 2025) extend collaboration to deployment by exploiting cross-client correlations or shared memories to exchange test-time knowledge. More recently, FedPAT (Guo et al., 2026) uses stable cross-client prototype relationships to guide adaptation on unseen clients, while COTE (Zhu et al., 2026) extends FedTTA to prompt-space

adaptation for vision-language models. BPFedCTTA (Wang et al., 2026) further considers sequential client arrivals, combining prior-guided local adaptation with uncertainty-gated aggregation to stabilize global model updates. However, existing FedTTA research primarily focuses on knowledge sharing under synchronous collaboration, whereas our work extends collaboration to asynchronous settings with concurrent adaptation and unequal client update rates. Beyond removing synchronization barriers, we account for update-frequency bias and error feedback in the collaboration design to support both adaptation gains and collaboration stability.

# 3 Preliminaries and Problem Formulation

## 3.1 Preliminaries

Consider a federated system comprising $N$ clients, denoted by $\mathcal{C} = \{C_1, C_2, \ldots, C_N\}$. After federated training, the server maintains a shared model parameterized by $\boldsymbol{\theta}_{\mathrm{g}}$, while each client performs local adaptation using unlabeled test data arriving during deployment. For client $C_i$, let $\mathcal{B}_i$ denote the current batch of test samples. The client updates its local model parameters $\boldsymbol{\theta}_i$ using an unsupervised TTA loss:

$$\boldsymbol{\theta}_i^{+} = \boldsymbol{\theta}_i - \eta \nabla_{\boldsymbol{\theta}_i} \mathcal{L}_{\mathrm{TTA}} \left( \boldsymbol{\theta}_i; \mathcal{B}_i \right). \tag{1}$$

Here, $\eta$ is the local adaptation learning rate, and $\boldsymbol{\theta}_i^{+}$ denotes the model parameters after adaptation. The loss $\mathcal{L}_{\mathrm{TTA}}$ is the unsupervised adaptation objective defined by the chosen TTA method, such as entropy minimization or pseudo-label learning.

FedTTA further allows clients to share their adaptation information. Let $\mathcal{U}_i$ denote the collection of adaptation information uploaded by client $C_i$. The server updates the shared model using the information received from participating clients:

$$\boldsymbol{\theta}_{\mathrm{g}}^{+} = \mathcal{G} \left( \boldsymbol{\theta}_{\mathrm{g}}, \{\mathcal{U}_i\}_{C_i \in \mathcal{S}} \right), \tag{2}$$

where $\mathcal{S} \subseteq \mathcal{C}$ is the set of clients participating in the current collaboration, and $\mathcal{G}$ denotes the server-side collaboration operator.

## 3.2 Problem Formulation

Existing FedTTA methods typically wait for participating clients to complete local adaptation before aggregating their adaptation information. When client processing speeds differ, this synchronization introduces delays caused by slower clients. We therefore consider asynchronous FedTTA, where each client uploads its adaptation information $\mathcal{U}_i$ immediately after local adaptation. Upon receiving an update, the server updates the shared model as

$$\boldsymbol{\theta}_{\mathrm{g}}^{+} = \mathcal{G} \left( \boldsymbol{\theta}_{\mathrm{g}}, \mathcal{U}_i \right). \tag{3}$$

The updated model is then returned to client $C_i$ for subsequent test data and local adaptation. This without-waiting collaboration improves responsiveness to dynamic environments, but processing each arriving update immediately introduces two key challenges.

**Challenge 1: Fast clients dominate the shared model.** In practical deployments, clients often operate in different target environments and receive test data from different distributions. Let $P_i$ denote the target distribution of client $C_i$, with $P_i \neq P_j$ in general. In asynchronous FedTTA, each client uploads an update immediately after local adaptation, allowing faster clients to contribute more frequently. Let $\pi_i$ denote the fraction of arriving updates contributed by client $C_i$. The target information reaching the server can be approximated by

$$P_{\mathrm{g}} = \sum_{i=1}^{N} \pi_i P_i. \tag{4}$$

Faster clients have larger $\pi_i$, so their target distributions are sampled more frequently during shared-model adaptation. Thus, differences in client speed translate into implicit weights over target distributions, giving faster clients greater cumulative influence and gradually biasing the shared model toward their adaptation direction.

**Challenge 2: Erroneous updates immediately affect subsequent clients.** Without ground-truth supervision, local TTA can reinforce incorrect predictions and produce erroneous updates. Suppose client $C_i$ uploads harmful adaptation information $\mathcal{U}_i^{\text{bad}}$. In asynchronous FedTTA, the server immediately incorporates it into the shared model:

$$\boldsymbol{\theta}_{\text{g}}^{+} = \mathcal{G}\left(\boldsymbol{\theta}_{\text{g}}, \mathcal{U}_i^{\text{bad}}\right). \tag{5}$$

The affected model $\boldsymbol{\theta}_{\text{g}}^{+}$ is then distributed to other clients, causing subsequent adaptation to start from a degraded state. Unlike single-client TTA, where errors mainly accumulate locally, asynchronous FedTTA allows such errors to propagate rapidly across clients.

# 4 Methodology

To enable stable knowledge sharing across clients in asynchronous FedTTA, the server maintains adaptation memory for different test environments and uses residual updates to reduce the cumulative influence of frequent uploads. Clients use shared knowledge only for prediction to limit error propagation.

Let $\boldsymbol{\theta} = (\phi, h)$ denote a classification model, where $\phi$ is the feature extractor and $h$ is the classifier. Its logit output for a sample $\mathbf{x}$ is $\mathbf{z}(\mathbf{x}) = h(\phi(\mathbf{x}))$. Each client maintains a fixed shared model $\boldsymbol{\theta}_{\text{g}}$, which remains frozen throughout testing, and a local adaptation model $\boldsymbol{\theta}_i$, which is continuously updated along the local test stream. The local model is initialized from the fixed shared model only at the beginning of testing. The server maintains a dynamic adaptation memory bank, $\mathcal{M} = \{(\mathbf{p}_j, \mathbf{M}_j)\}_{j=1}^{J}$, where $\mathbf{p}_j$ is the prototype of the $j$-th test environment and $\mathbf{M}_j \in \mathbb{R}^{K \times D}$ is its adaptation memory matrix. Here, $K$ is the number of classes and $D$ is the sample feature dimension. Each memory matrix maps sample features to logit corrections for the corresponding environment.

## 4.1 Local Adaptation and Shared Knowledge Utilization

For the current test batch $\mathcal{B}_i$, client $i$ performs TTA according to Eq. 1, starting from its previous local model state. This produces the adapted model $\boldsymbol{\theta}_i^{+}$, whose output $\mathbf{z}_i^{+}(\mathbf{x})$ serves as the basis for the current prediction. The client caches only the most recently received environment prototype $\widehat{\mathbf{p}}_i$ and adaptation memory matrix $\widehat{\mathbf{M}}_i$. It uses this shared knowledge to complement local adaptation without storing the entire adaptation memory bank $\mathcal{M}$.

Because the test environment may change, the client checks whether the cached entry remains applicable before using it. Specifically, it extracts intermediate-layer features for the current batch using the fixed shared model and computes their mean $\boldsymbol{\mu}_i$ and standard deviation $\boldsymbol{\sigma}_i$. These statistics are concatenated and $\ell_2$-normalized to form the environment descriptor $\mathbf{s}_i$. We use intermediate-layer features because they are more sensitive to environment-induced distribution shifts than shallow features dominated by local textures or deep features dominated by class semantics.

If a cached entry exists and the similarity between its prototype and the current environment descriptor is greater than the environment matching threshold $\delta$, the client considers it applicable to the current environment. The client then uses the cached adaptation memory matrix to correct the fixed shared model's output and combines it with the local TTA output:

$$\mathbf{z}_{\text{pred}}(\mathbf{x}) = (1-\lambda)\mathbf{z}_i^{+}(\mathbf{x}) + \lambda\left[\mathbf{z}_{\text{g}}(\mathbf{x}) + \widehat{\mathbf{M}}_i\mathbf{k}(\mathbf{x})\right]. \tag{6}$$

Here, $\mathbf{k}(\mathbf{x}) = \phi_{\text{g}}(\mathbf{x})/\|\phi_{\text{g}}(\mathbf{x})\|_2 \in \mathbb{R}^D$ is the normalized final feature representation from the fixed shared model. The term $\widehat{\mathbf{M}}_i\mathbf{k}(\mathbf{x})$ provides a logit correction based on the sample features, and $\lambda \in [0,1]$ controls the contribution of shared knowledge. If no cached entry exists or the environment does not match, the client uses only the local TTA output: $\mathbf{z}_{\text{pred}}(\mathbf{x}) = \mathbf{z}_i^{+}(\mathbf{x})$. The predicted class is the one with the largest logit in $\mathbf{z}_{\text{pred}}(\mathbf{x})$.

This fusion is used only for the current prediction. It does not enter the local TTA loss, pseudo-label generation, or model updates. The local model therefore continues adapting

from its own state, preventing errors in shared predictions from being learned through local adaptation.

After prediction, the client extracts and uploads its local adaptation information. To provide a common reference across clients, we define a correction vector relative to the fixed shared model:

$$\mathbf{v}(\mathbf{x}) = \left(\mathbf{I}_K - \frac{1}{K}\mathbf{1}\mathbf{1}^\top\right)\left[\mathbf{z}_i^+(\mathbf{x}) - \mathbf{z}_{\mathrm{g}}(\mathbf{x})\right]. \tag{7}$$

Here, $\mathbf{I}_K$ is the identity matrix and $\mathbf{1}$ is the $K$-dimensional all-ones vector. This centering operation removes the logit offset shared across classes while preserving the relative changes among class logits induced by local TTA. Shared knowledge affects only the final prediction, whereas the uploaded correction vector is always computed from the difference between the local adaptation model and the fixed shared model. Errors in shared predictions are therefore not fed back into subsequent uploads, limiting their continued influence on other clients.

The client uploads the environment descriptor together with the sample representations and correction vectors for the current batch: $\mathcal{U}_i = \left(\mathbf{s}_i, \left\{\left(\mathbf{k}(\mathbf{x}), \mathbf{v}(\mathbf{x})\right)\right\}_{\mathbf{x}\in\mathcal{B}_i}\right)$. Adaptation information is uploaded asynchronously after each batch. To reduce additional communication, the client requests a new cached entry only when its cache is missing or no longer matches the current environment.

### 4.2 Asynchronous Updates of Adaptation Memory

The server processes each upload $\mathcal{U}_i$ immediately upon arrival, without waiting for other clients. It first uses the environment descriptor $\mathbf{s}_i$ to select an adaptation memory matrix, then updates it using the uploaded sample representations and correction vectors.

If the adaptation memory bank is empty, or the maximum similarity between $\mathbf{s}_i$ and the existing environment prototypes is below $\delta$, the server treats the upload as coming from a new test environment and initializes a new entry:

$$\mathbf{p}_{J+1} = \mathbf{s}_i, \mathbf{M}_{J+1} = \frac{1}{|\mathcal{B}_i|}\sum_{\mathbf{x}\in\mathcal{B}_i}\mathbf{v}(\mathbf{x})\mathbf{k}(\mathbf{x})^\top. \tag{8}$$

The server then adds $(\mathbf{p}_{J+1}, \mathbf{M}_{J+1})$ to the adaptation memory bank, storing adaptation information for the new environment separately from existing environments.

Otherwise, the server assigns the upload to the most similar existing environment, indexed by $j^\star$. Within the same environment, frequent uploads may contain similar adaptation information. To reduce their cumulative influence, we use a delta-rule update that writes only the residual correction not yet represented by the current adaptation memory matrix:

$$\mathbf{M}_{j^\star}^+ = \mathbf{M}_{j^\star} + \frac{1}{|\mathcal{B}_i|}\sum_{\mathbf{x}\in\mathcal{B}_i}\left[\mathbf{v}(\mathbf{x}) - \mathbf{M}_{j^\star}\mathbf{k}(\mathbf{x})\right]\mathbf{k}(\mathbf{x})^\top. \tag{9}$$

Here, $\mathbf{M}_{j^\star}\mathbf{k}(\mathbf{x})$ is the correction produced by the current adaptation memory matrix, and $\mathbf{v}(\mathbf{x}) - \mathbf{M}_{j^\star}\mathbf{k}(\mathbf{x})$ is its residual relative to the uploaded correction. When the adaptation memory matrix already represents the uploaded information, the residual is small, so repeated uploads produce smaller updates. The update magnitude therefore depends on information not yet represented, reducing the cumulative influence of repeated uploads from fast clients.

Alongside the adaptation memory matrix, the server updates the corresponding environment prototype using an exponential moving average followed by normalization:

$$\mathbf{p}_{j^\star}^+ = \frac{(1-\beta)\mathbf{p}_{j^\star} + \beta\mathbf{s}_i}{\left\|(1-\beta)\mathbf{p}_{j^\star} + \beta\mathbf{s}_i\right\|_2}. \tag{10}$$

Here, $\beta \in [0, 1]$ controls the prototype update rate, allowing subsequent environment matching to track distribution changes within the same environment. After processing the upload, the server returns the updated environment prototype and adaptation memory matrix only if the client has requested a new cached entry.

# 5 Experiments

## 5.1 Experimental Setup

**Datasets and Models:** We conduct image classification experiments on CIFAR10-C, CIFAR100-C, and ImageNet-C, using all 15 corruption types at severity level 5. We adopt ResNet-18 for CIFAR10-C and CIFAR100-C, and ResNet-50 for ImageNet-C. We further evaluate ViT-B/16 on ImageNet-C to examine the applicability of our method across different network architectures. All models are initialized from publicly available pretrained models on the corresponding clean datasets, with further details provided in the Appendix.

**Asynchronous Communication and Dynamic Heterogeneous Test Streams:** We construct a dynamic test stream for each client. Each client sequentially experiences all 15 corruption types at severity level 5, with each corruption lasting for 120 test samples before switching to the next environment. Different clients follow different corruption orders, such that they may encounter different environments at the same time. To further simulate heterogeneous client computation speeds, each client $i$ is assigned a fixed local adaptation time $t_i = \max(0.1, 1 + \alpha\epsilon_i)$, where $\epsilon_i \sim \mathcal{N}(0, 1)$ and $\alpha$ controls the degree of client speed heterogeneity. Unless otherwise specified, we set $\alpha = 0.4$. Clients independently progress through their test streams according to their own speeds and upload an update immediately after completing local adaptation. A smaller $t_i$ therefore corresponds to a faster client and a higher update frequency. The server processes updates immediately according to their actual arrival order.

**Baselines and Implementation Details:** We adopt Tent (Wang et al., 2021), SAR (Niu et al., 2023), and DeYO (Lee et al., 2024) as three representative local TTA methods and construct four controlled baselines under a unified experimental framework: Local TTA, Synchronous FedTTA (SFTTA), Asynchronous FedTTA (AFTTA), and FedAsync-based TTA (FATTA). Local TTA performs adaptation independently on each client without inter-client collaboration; SFTTA waits for client updates and aggregates them synchronously; AFTTA immediately aggregates each update upon arrival; and FATTA adapts the staleness-aware aggregation strategy of FedAsync (Xie et al., 2019) to FedTTA by weighting each arriving local TTA update according to its staleness. We use 60 clients in all experiments, with a batch size of 16 and one local adaptation step per batch. The optimizer, learning rate, and method-specific hyperparameters of Tent, SAR, and DeYO follow their original implementations, with details provided in the Appendix. For our method, the environment matching threshold, prediction fusion weight, and prototype update rate are set to $\delta = 0.7$, $\lambda = 0.5$, and $\beta = 0.1$, respectively, unless otherwise specified. Unless otherwise specified, all methods use the same experimental settings. Models continuously perform prediction and adaptation over the full test stream, and we report the average online accuracy. All experiments are independently repeated with five random seeds, and the results are reported as the mean and standard deviation.

## 5.2 Main Results

To evaluate the effectiveness of our method across different local TTA methods and model architectures, we conduct experiments with Tent, SAR, and DeYO, and compare against Local TTA, SFTTA, AFTTA, and FATTA. As shown in Tab. 1, our method consistently performs well across different datasets, local TTA methods, and model architectures, indicating that the shared adaptation memory can effectively integrate test-time knowledge acquired from different clients. In contrast, direct asynchronous aggregation generally performs worse than synchronous collaboration, suggesting that simply updating the shared state according to the arrival order can make knowledge sharing more sensitive to differences in client update frequencies and arrival patterns. When some clients upload similar adaptation information more frequently, their knowledge may be repeatedly accumulated in the shared state, reducing the stability of cross-client collaboration. Our method addresses this issue through residual write-in, which absorbs only the adaptation information that has not yet been represented by the current shared memory, thereby reducing the cumulative influence of redundant updates. Meanwhile, the shared memory is used only for prediction correction and does not participate in the initialization or optimization of local

Table 1: Performance comparison of different methods across datasets and their corresponding models. Best result in bold.

| Methods | Dataset & Model | Local TTA | SFTTA | AFTTA | FATTA | Ours |
|---|---|---|---|---|---|---|
| Tent | CIFAR10-C+ResNet-18 | 73.10 ± 0.29 | 76.23 ± 0.32 | 74.97 ± 0.31 | 75.60 ± 0.27 | **77.68 ± 0.19** |
| | CIFAR100-C+ResNet-18 | 48.17 ± 0.21 | 51.68 ± 0.22 | 50.38 ± 0.17 | 50.56 ± 0.26 | **53.42 ± 0.32** |
| | ImageNet-C+ResNet-50 | 23.33 ± 0.34 | 25.56 ± 0.46 | 24.39 ± 0.41 | 25.25 ± 0.39 | **28.38 ± 0.05** |
| | ImageNet-C+ViT-B/16 | 37.54 ± 0.24 | 39.90 ± 0.27 | 38.65 ± 0.23 | 39.55 ± 0.20 | **40.28 ± 0.24** |
| SAR | CIFAR10-C+ResNet-18 | 73.09 ± 0.32 | 75.31 ± 0.32 | 74.15 ± 0.32 | 75.31 ± 0.31 | **77.86 ± 0.25** |
| | CIFAR100-C+ResNet-18 | 48.02 ± 0.14 | 50.40 ± 0.24 | 49.14 ± 0.25 | 50.13 ± 0.23 | **53.18 ± 0.33** |
| | ImageNet-C+ResNet-50 | 23.07 ± 0.40 | 25.22 ± 0.46 | 23.89 ± 0.44 | 25.10 ± 0.41 | **28.11 ± 0.20** |
| | ImageNet-C+ViT-B/16 | 37.35 ± 0.26 | 39.66 ± 0.33 | 38.46 ± 0.11 | 39.44 ± 0.20 | **40.02 ± 0.27** |
| DeYO | CIFAR10-C+ResNet-18 | 73.30 ± 0.34 | 75.86 ± 0.26 | 74.58 ± 0.23 | 75.48 ± 0.26 | **78.05 ± 0.27** |
| | CIFAR100-C+ResNet-18 | 48.96 ± 0.10 | 51.80 ± 0.20 | 50.78 ± 0.21 | 50.55 ± 0.24 | **53.88 ± 0.29** |
| | ImageNet-C+ResNet-50 | 22.90 ± 0.08 | 26.73 ± 0.22 | 25.38 ± 0.31 | 25.66 ± 0.58 | **28.47 ± 0.23** |
| | ImageNet-C+ViT-B/16 | 36.41 ± 0.34 | 40.64 ± 0.40 | 39.43 ± 0.55 | 39.86 ± 0.16 | **41.02 ± 0.20** |

TTA, keeping local adaptation isolated from global knowledge sharing. Overall, our method maintains stable and consistent performance across different TTA methods, datasets, and model architectures without requiring synchronization waiting.

### 5.3 Analysis of Fast Client Dominance

To further analyze fast client dominance, we consider one slow client and one fast client with a processing speed ratio of 1 : 8, while keeping all other settings identical. As shown in Fig. 3, AFTTA and FATTA give the fast client a much larger accuracy gain because its more frequent updates accumulate greater influence on the shared state. In contrast, our method only absorbs information not yet represented in the shared memory, so repeated updates from the fast client contribute progressively less. Meanwhile, the slow client can still benefit from cross client knowledge stored in the shared memory. As a result, our method reduces the advantage caused by frequent updates and alleviates fast client dominance.

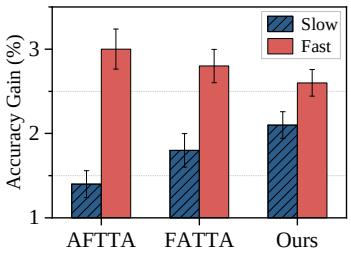


Figure 3: Accuracy gains of slow and fast clients.

### 5.4 Effect of erroneous updates

To evaluate the impact of erroneous updates, we inject a single faulty update at a fixed arrival event and compare the accuracy trajectories with and without fault injection. The dashed line marks its arrival at the server. In AFTTA, the trajectories diverge rapidly, showing that one faulty update can perturb the shared state and influence subsequent adaptation. FATTA reduces this deviation through staleness aware weighting, but the faulty information is still incorporated into the shared model. In contrast, the trajectories of our method remain nearly overlapping. The residual update mechanism limits the influence of each update on the shared memory, while the isolation mechanism prevents affected shared knowledge from entering subsequent local TTA, thereby limiting the propagation of erroneous updates.

### 5.5 Runtime Overhead

To evaluate runtime efficiency, we measure the wall-clock time required by our method and SFTTA to complete the same test workload. Both methods process identical test streams under the same hardware configuration and client-speed heterogeneity settings. SFTTA imposes a synchronization barrier before each subsequent collaboration round, whereas our method allows clients to continue independently and processes their uploads as they arrive. As shown in Fig. 5, our method reduces the measured wall-clock time by 55.0%–64.2% on CIFAR10-C with ResNet-18, 55.9%–64.0% on CIFAR100-C with ResNet-18, 44.6%–48.1%

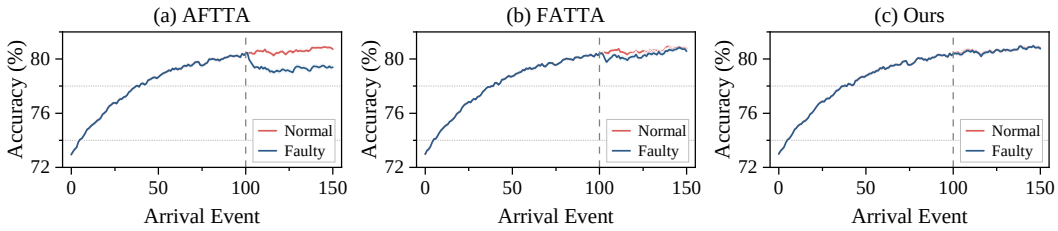


Figure 4: Faulty update effect on CIFAR10-C, with the dashed line indicating its arrival.

on ImageNet-C with ResNet-50, and 57.1%–60.0% on ImageNet-C with ViT-B/16, with each range covering the three local TTA methods. These results demonstrate lower end-to-end runtime for our asynchronous implementation under the evaluated experimental settings.

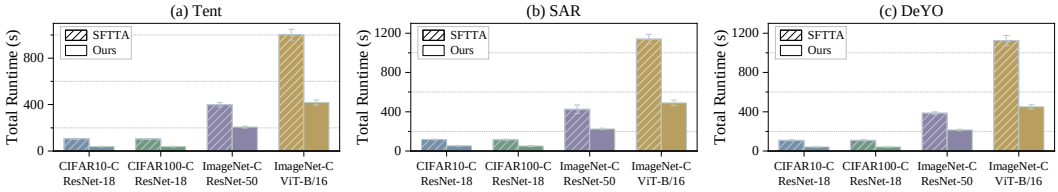


Figure 5: Wall-clock time comparison between our method and SFTTA across different datasets, models, and local TTA methods.

### 5.6 Hyperparameter Sensitivity

We further examine the sensitivity of our method to hyperparameter choices. Fig. 6 shows the distributions of accuracy gains over the corresponding Local TTA baselines across 20 hyperparameter combinations, measured in percentage points. Within each dataset–model setting, the gains for each TTA method remain relatively concentrated across the tested configurations. This suggests that our method is not overly sensitive to hyperparameter choices within the evaluated ranges.

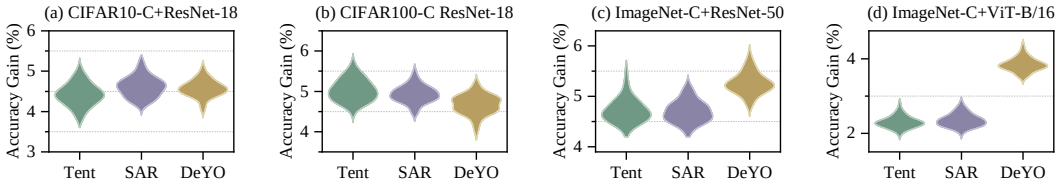


Figure 6: Distribution of accuracy gains over 20 hyperparameter combinations across different datasets, models, and local TTA methods.

## 6 Conclusion

This work extends FedTTA to asynchronous collaboration, allowing clients with heterogeneous processing speeds to share adaptation knowledge without synchronization waiting. The proposed method addresses fast client dominance and cross-client error propagation through residual updates to a dynamic shared memory and the separation of shared knowledge from local adaptation. Residual updates reduce the cumulative influence of redundant uploads, while shared knowledge is used only for prediction correction to prevent error feedback into local adaptation. Extensive experiments demonstrate the effectiveness, robustness, and efficiency of our method. Future work will explore how shared knowledge can safely guide local adaptation without reintroducing error feedback.

## AI Use Statement

During this study, we used generative AI tools to assist with the retrieval and discovery of related work, as well as with the writing of portions of the experimental code. All relevant references were verified by the authors against the original publications, and all AI-assisted code was manually inspected and validated through actual execution. All experimental results reported in this paper were obtained from experiments conducted by the authors. The authors reviewed all AI-assisted content and take full responsibility for the final content and results presented in this work.

## References

Wenxuan Bao, Ruxi Deng, Ruizhong Qiu, Tianxin Wei, Hanghang Tong, and Jingrui He. Latte: Collaborative test-time adaptation of vision-language models in federated learning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 133–143, 2025. doi: 10.1109/ICCV51701.2025.00020.

Yue Chen, Jianfeng Lu, Shuqin Cao, Wei Wang, Gang Li, and Guanghui Wen. FedCure: Mitigating participation bias in semi-asynchronous federated learning with Non-IID data. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 20418–20426, 2026. doi: 10.1609/aaai.v40i25.39176.

Yann Fraboni, Richard Vidal, Laetitia Kameni, and Marco Lorenzi. A general theory for federated optimization with asynchronous and heterogeneous clients updates. *Journal of Machine Learning Research*, 24(110):1–43, 2023. URL https://jmlr.org/papers/v24/22-0689.html.

Taesik Gong, Jongheon Jeong, Taewon Kim, Yewon Kim, Jinwoo Shin, and Sung-Ju Lee. NOTE: Robust continual test-time adaptation against temporal correlation. In *Advances in Neural Information Processing Systems*, volume 35, pp. 27253–27266, 2022. doi: 10.52202/068431-1976.

Shunxin Guo, Jiaqi Lv, Zhiqiang Kou, Shuxia Lin, and Xin Geng. FedPAT: Federated test-time adaptation via prototype affinity topology. In *Proceedings of the 43rd International Conference on Machine Learning*, volume 306 of *Proceedings of Machine Learning Research*. PMLR, 2026. URL https://icml.cc/virtual/2026/poster/65320.

Yusuke Iwasawa and Yutaka Matsuo. Test-time classifier adjustment module for model-agnostic domain generalization. In *Advances in Neural Information Processing Systems*, volume 34, pp. 2427–2440, 2021. URL https://proceedings.neurips.cc/paper/2021/hash/1415fe9fea0fa1e45dddcff5682239a0-Abstract.html.

Liangze Jiang and Tao Lin. Test-time robust personalization for federated learning. In *International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=3aBuJEza5sq.

Guannan Lai, Da-Wei Zhou, Zhenguo Li, and Han-Jia Ye. The golden subspace: Where efficiency meets generalization in continual test-time adaptation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 3866–3875, 2026. URL https://openaccess.thecvf.com/content/CVPR2026/html/Lai_The_Golden_Subspace_Where_Efficiency_Meets_Generalization_in_Continual_Test-Time_CVPR_2026_paper.html.

Jonghyun Lee, Dahuin Jung, Saehyung Lee, Junsung Park, Juhyeon Shin, Uiwon Hwang, and Sungroh Yoon. Entropy is not enough for test-time adaptation: From the perspective of disentangled factors. In *International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/cd0665986c1e9c15f6569ac944bcf88a-Abstract-Conference.html.

Qinbin Li, Zeyi Wen, Zhaomin Wu, Sixu Hu, Naibo Wang, Yuan Li, Xu Liu, and Bingsheng He. A survey on federated learning systems: Vision, hype and reality for data privacy

and protection. *IEEE Transactions on Knowledge and Data Engineering*, 35(4):3347–3366, 2023. doi: 10.1109/TKDE.2021.3124599.

Zilinghan Li, Pranshu Chaturvedi, Shilan He, Han Chen, Gagandeep Singh, Volodymyr Kindratenko, Eliu A. Huerta, Kibaek Kim, and Ravi Madduri. FedCompass: Efficient cross-silo federated learning on heterogeneous client devices using a computing power-aware scheduler. In *International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/a9f3457fa97f106f1756885237787789-Abstract-Conference.html.

Haoyuan Liang, Xinyu Zhang, Shilei Cao, Guowen Li, and Juepeng Zheng. TTA-FedDG: Leveraging test-time adaptation to address federated domain generalization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 18658–18666, 2025. doi: 10.1609/aaai.v39i18.34053.

John Nguyen, Kshitiz Malik, Hongyuan Zhan, Ashkan Yousefpour, Mike Rabbat, Mani Malek, and Dzmitry Huba. Federated learning with buffered asynchronous aggregation. In *Proceedings of the 25th International Conference on Artificial Intelligence and Statistics*, volume 151 of *Proceedings of Machine Learning Research*, pp. 3581–3607. PMLR, 2022. URL https://proceedings.mlr.press/v151/nguyen22b.html.

Shuaicheng Niu, Jiaxiang Wu, Yifan Zhang, Yaofo Chen, Shijian Zheng, Peilin Zhao, and Mingkui Tan. Efficient test-time model adaptation without forgetting. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 16888–16905. PMLR, 2022. URL https://proceedings.mlr.press/v162/niu22a.html.

Shuaicheng Niu, Jiaxiang Wu, Yifan Zhang, Zhiquan Wen, Yaofo Chen, Peilin Zhao, and Mingkui Tan. Towards stable test-time adaptation in dynamic wild world. In *International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=g2YraF75Tj.

Xiaoyi Pang, Yajie Zhou, Zhibo Wang, Xuanyi Hao, Tao Han, and Song Guo. Unlocking the potential of asynchronous federated learning in Non-IID edge environments. *IEEE Transactions on Mobile Computing*, 25(10):15897–15908, 2026. doi: 10.1109/TMC.2026.3683093.

Rakibul Hasan Rajib, Md. Akil Raihan Iftee, Mir Sazzat Hossain, A. K. M. Mahbubur Rahman, Sajib Mistry, M. Ashraful Amin, and Amin Ahsan Ali. FedCTTA: A collaborative approach to continual test-time adaptation in federated learning. In *2025 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, 2025. doi: 10.1109/IJCNN64981.2025.11228519.

Virginia Smith, Chao-Kai Chiang, Maziar Sanjabi, and Ameet S. Talwalkar. Federated multi-task learning. In *Advances in Neural Information Processing Systems*, volume 30, pp. 4424–4434, 2017. URL https://proceedings.neurips.cc/paper/2017/hash/6211080fa89981f66b1a0c9d55c61d0f-Abstract.html.

Yue Tan, Chen Chen, Weiming Zhuang, Xin Dong, Lingjuan Lyu, and Guodong Long. Is heterogeneity notorious? taming heterogeneity to handle test-time shift in federated learning. In *Advances in Neural Information Processing Systems*, volume 36, pp. 27167–27180, 2023. doi: 10.52202/075280-1183.

Dequan Wang, Evan Shelhamer, Shaoteng Liu, Bruno Olshausen, and Trevor Darrell. Tent: Fully test-time adaptation by entropy minimization. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=uXl3bZLkr3c.

Liwen Wang, Xingbo Dong, Iman Yi Liao, and Zhe Jin. Towards stable federated continual test-time adaptation in wild world. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 29451–29461, 2026. URL https://openaccess.thecvf.com/content/CVPR2026/html/Wang_Towards_Stable_

Federated_Continual_Test-Time_Adaptation_in_Wild_World_CVPR_2026_paper.html.

Qin Wang, Olga Fink, Luc Van Gool, and Dengxin Dai. Continual test-time domain adaptation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 7201–7211, 2022. doi: 10.1109/CVPR52688.2022.00706.

Cong Xie, Sanmi Koyejo, and Indranil Gupta. Asynchronous federated optimization. *arXiv preprint arXiv:1903.03934*, 2019. URL https://arxiv.org/abs/1903.03934.

Linjing You, Jiabao Lu, Xiayuan Huang, and Xiangli Nie. FRET: Feature redundancy elimination for test time adaptation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 2120–2130, 2025. URL https://openaccess.thecvf.com/content/ICCV2025/html/You_FRET_Feature_Redundancy_Elimination_for_Test_Time_Adaptation_ICCV_2025_paper.html.

Chen Zhang, Ge Su, Huaxiao Zhou, Lu Hao, Fenglin Zhu, Jintai Chen, Yuxiang Cai, Jianwei Yin, and Hongxia Xu. Federated learning meets test-time adaptation: Methods, challenges, and future directions. *International Journal of Computer Vision*, 134(8):361, 2026. doi: 10.1007/s11263-026-02937-6.

Jiayuan Zhang, Xuefeng Liu, Yukang Zhang, Guogang Zhu, Jianwei Niu, and Shaojie Tang. Enabling collaborative test-time adaptation in dynamic environment via federated learning. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 4191–4202, 2024. doi: 10.1145/3637528.3671908.

Rui Zhu, Liang Bai, Yanming Guo, Yirun Ruan, Tianyuan Yu, and Zhihe Lu. Controllable federated prompt learning at test time. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 39455–39465, 2026. URL https://openaccess.thecvf.com/content/CVPR2026/html/Zhu_Controllable_Federated_Prompt_Learning_at_Test_Time_CVPR_2026_paper.html.

# Appendix

## A.1 Experimental Details

We conduct all experiments on a high-performance deep learning server running Ubuntu 20.04.4 LTS. The server is equipped with two Intel(R) Xeon(R) Silver 4316 CPUs at 2.30 GHz and four NVIDIA GeForce RTX 4090 GPUs, each with approximately 24 GB of memory. Our implementation is based on Python 3.10 and PyTorch 2.4.1, covering model construction, local adaptation, server aggregation, and communication.

**Datasets.** We conduct image classification experiments on three widely used corruption robustness benchmarks: CIFAR10-C, CIFAR100-C, and ImageNet-C. These datasets are constructed from the clean test sets of CIFAR10, CIFAR100, and ImageNet, respectively. Each benchmark contains 15 standard corruption types, and each corruption has five severity levels. A higher severity level indicates a stronger corruption. Representative examples of different corruptions at severity level 5 are shown in Fig. 7. All experiments in this work use severity level 5. For CIFAR10-C and CIFAR100-C, each corruption contains 10,000 test images at each severity level. For ImageNet-C, each corruption contains 50,000 test images at each severity level.

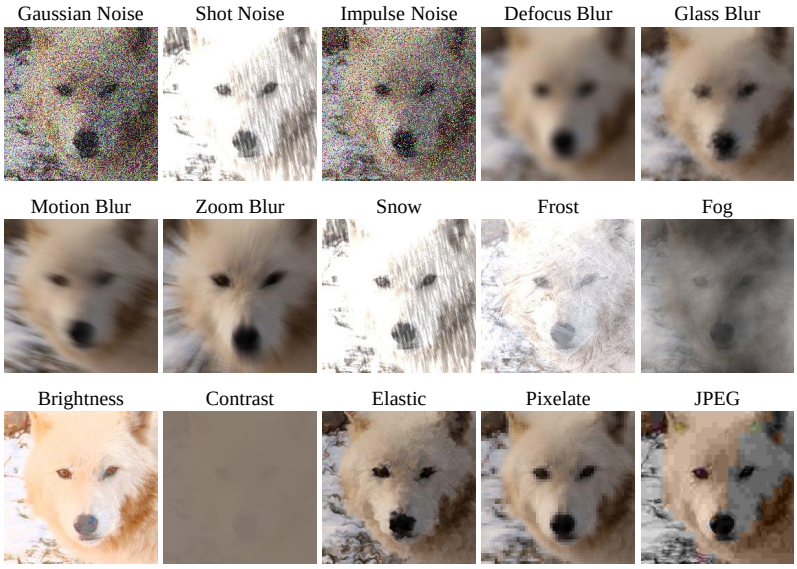


Figure 7: Examples of the 15 corruption types in ImageNet-C at severity level 5. The same ImageNet image is corrupted by different synthetic perturbations, including noise, blur, weather effects, and digital distortions.

**Models.** For reproducibility, we use publicly available pretrained models as the source models for all datasets. For CIFAR10-C, we use a ResNet-18[1] pretrained on the CIFAR10 training set. For CIFAR100-C, we use a ResNet-18[2] pretrained on the CIFAR100 training set. For ImageNet-C, we use a ResNet-50[3] pretrained on the ImageNet-1K training set as the default source model. We further evaluate a ViT-B/16[4] pretrained on ImageNet-1K to examine the applicability of our method across different network architectures.

**Local TTA Configuration.** Our method does not rely on a specific local TTA strategy and can be integrated with different local test-time adaptation methods. We select Tent,

[1] https://drive.google.com/file/d/1YsyHY3rFCaWWDTOVh-RuI1I2bJ5i9Yey/view?usp=sharing

[2] https://drive.google.com/file/d/10JVqrgnTlG9Mbj7ogNww2MZZHHlFyzB8/view?usp=sharing

[3] https://download.pytorch.org/models/resnet50-0676ba61.pth

[4] https://download.pytorch.org/models/vit_b_16-c867db91.pth

SAR, and DeYO as three representative local TTA methods for evaluation. The optimizer, learning rate, and method-specific hyperparameters of each method are configured following their original papers and official implementations. The detailed settings are provided in Tab. 2.

Table 2: Configurations of the local TTA methods. $K$ denotes the number of classes.

| Method | Hyperparameter | ResNet-18 | ResNet-50 | ViT-B/16 |
|---|---|---|---|---|
| **Tent** | optimizer | Adam | SGD | SGD |
| | learning rate | $1 \times 10^{-3}$ | $2.5 \times 10^{-4}$ | $1 \times 10^{-3}$ |
| | momentum | – | 0.9 | 0.9 |
| **SAR** | optimizer | SAM (SGD) | SAM (SGD) | SAM (SGD) |
| | learning rate | $2.5 \times 10^{-4}$ | $2.5 \times 10^{-4}$ | $1 \times 10^{-3}$ |
| | momentum | 0.9 | 0.9 | 0.9 |
| | radius $\rho$ | 0.05 | 0.05 | 0.05 |
| | entropy threshold $E_0$ | $0.4 \ln K$ | $0.4 \ln K$ | $0.4 \ln K$ |
| | moving average factor for $e_m$ | 0.9 | 0.9 | 0.9 |
| | reset threshold $e_0$ | 0.2 | 0.2 | 0.2 |
| **DeYO** | optimizer | SGD | SGD | SGD |
| | learning rate | $2.5 \times 10^{-4}$ | $2.5 \times 10^{-4}$ | $1 \times 10^{-3}$ |
| | momentum | 0.9 | 0.9 | 0.9 |
| | entropy threshold $\tau_{\text{Ent}}$ | $0.5 \ln K$ | $0.5 \ln K$ | $0.5 \ln K$ |
| | PLPD threshold $\tau_{\text{PLPD}}$ | 0.2 | 0.2 | 0.2 |
| | normalization factor $\text{Ent}_0$ | $0.4 \ln K$ | $0.4 \ln K$ | $0.4 \ln K$ |
| | number of patches | $4 \times 4$ | $4 \times 4$ | $4 \times 4$ |

**Feature Layer Selection.** We adopt ResNet-18, ResNet-50, and ViT-B/16. ResNet-18 and ResNet-50 each contain four residual stages, using BasicBlock and Bottleneck, respectively, while ViT-B/16 contains 12 Transformer encoder blocks. We denote the ResNet stages as Layer 1–4 and the ViT blocks as Layer 1–12. Since the environment descriptor $\mathbf{s}_i$ and sample representation $\mathbf{k}(\mathbf{x})$ serve different purposes, they are extracted from different layers. $\mathbf{s}_i$ captures environment variations such as texture, background, and corruption, and is therefore built from intermediate features. In contrast, $\mathbf{k}(\mathbf{x})$ requires stronger semantic information and is extracted from the final feature before the classification head. Specifically, we use Layer 2, Layer 2, and Layer 6 for ResNet-18, ResNet-50, and ViT-B/16, with dimensions of 128, 512, and 768, respectively. The corresponding final feature dimensions are 512, 2048, and 768. As shown in Fig. 8, intermediate layers focus more on environment related regions, while higher layers focus more on discriminative object regions.

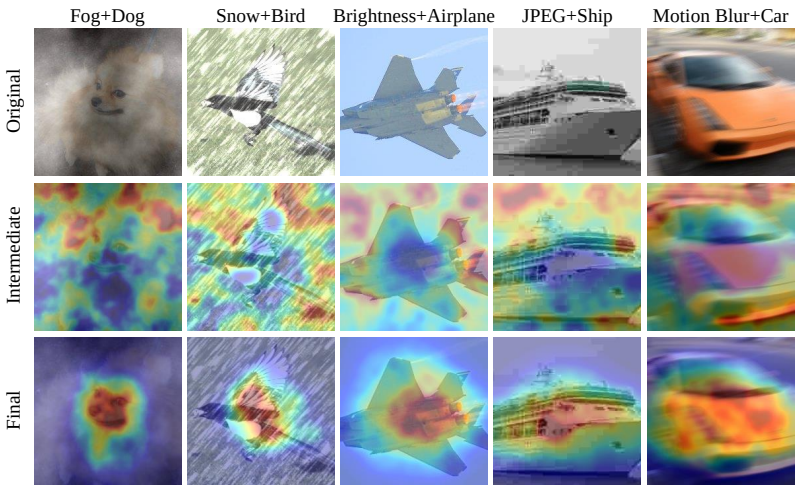


Figure 8: Grad-CAM visualizations of ViT-B/16. The first row shows input samples, and the second and third rows show Layer 6 and Layer 12, respectively. Layer 6 focuses more on environment related regions, while Layer 12 focuses more on discriminative object regions.

**Visualization of Client Speed Heterogeneity.** We use $\alpha$ to control the degree of heterogeneity in client computation speeds. A larger $\alpha$ increases the variation in client processing times, thereby leading to a more imbalanced update arrival frequency. Fig. 9 visualizes the asynchronous update arrivals of 60 clients under different levels of speed heterogeneity.

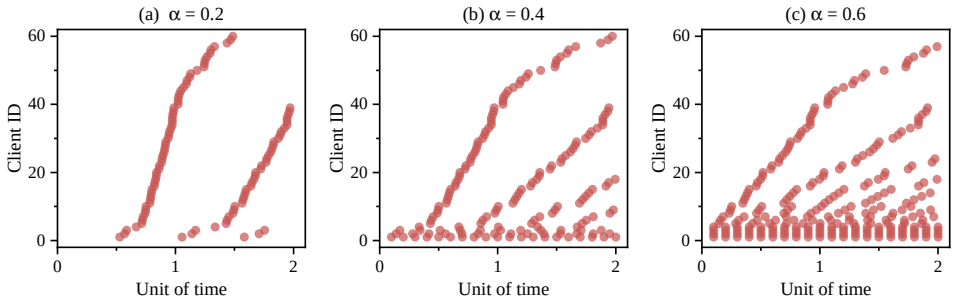


Figure 9: Asynchronous update arrivals under different client speed heterogeneity levels.

**Details of Motivating Experiments.** We further provide the detailed settings of the two motivating experiments in Fig. 2. For Fig. 2(a), we construct two target domain clients on CIFAR10-C, where client 0 corresponds to fog (D1) and client 1 corresponds to spatter (D2). Both clients use the same Tent configuration, and the server immediately aggregates updates in their arrival order. By adjusting the client processing speeds, we consider seven D1:D2 upload frequency ratios: 1:8, 1:4, 1:2, 1:1, 2:1, 4:1, and 8:1, and report the mean and standard deviation over five random seeds. Without TTA, the accuracies on D1 and D2 are 86.12% and 84.03%, respectively, corresponding to the horizontal dashed lines in the figure. As the relative upload frequency of client 0 increases, the accuracy on D1 generally increases while that on D2 decreases. To further quantify this effect, we use the 1:1 setting as the reference and measure the relative model shift by the difference between the accuracy changes of the two domains. As shown in Fig. 10, a higher upload frequency of client 0 leads to a larger shift toward D1. Each doubling of its relative upload frequency results in an average shift of 0.203 percentage points toward D1, with a 95% confidence interval of $[0.140, 0.266]$. The Pearson correlation coefficient is 0.918, indicating a strong positive correlation between upload frequency and model shift.

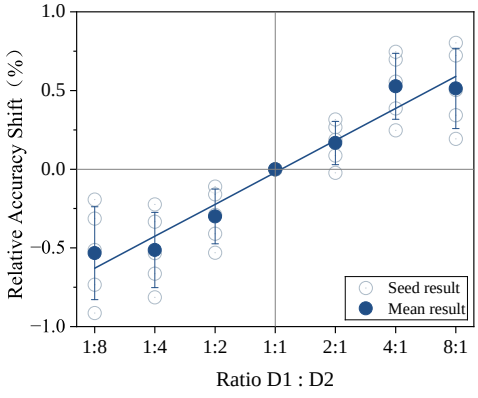


Figure 10: Relative accuracy shift under different upload frequencies. D1 and D2 denote the fog and spatter domains, respectively. The x axis shows the upload frequency ratio, and the y axis shows the accuracy shift relative to the 1:1 setting. Positive values indicate a shift toward D1, while negative values indicate a shift toward D2. Hollow circles show five independent runs, and solid circles show the mean.

For Fig. 2(b), we construct a single faulty update experiment on Gaussian Noise in CIFAR10-C. The experiment includes four clients, each performing Tent on a disjoint subset of samples.

After 100 normal asynchronous updates, one Tent update from client 0 is replaced with an artificially constructed faulty update. Specifically, we cyclically shift the labels of the current batch and perform one cross entropy update using the incorrect labels. The update norm is then scaled to the median norm of normal Tent updates, so that the faulty update has a similar magnitude but a different direction. The ground truth labels are used only to construct this single faulty update, while normal TTA remains fully unlabeled. The normal and faulty trajectories are identical except for this update. We compare model performance after server aggregation and after subsequent clients complete local Tent, and report the accuracy difference between the two trajectories.

## A.2 Supplementary Experiments

**Effect of Different Pretrained Source Models.** To investigate the effect of pretrained source models on test-time adaptation performance, we further evaluate two publicly available CIFAR-10 ResNet-18 models on CIFAR-10-C, in addition to the source model used in the main experiments. We denote the three models as Model 1, Model 2[5], and Model 3[6], respectively, where Model 1 is the source model used in the main experiments. For a fair comparison, we vary only the initial source model while keeping all other experimental settings unchanged. As shown in Tab. 3, different pretrained source models lead to noticeable differences in absolute adaptation performance, while the relative performance trends across methods remain largely consistent. Federated methods consistently improve over Local TTA, with comparable gains across different source models. In particular, our method achieves the best or near-best performance across all three models. These results indicate that although the choice of source model affects the overall performance level, it has only a limited impact on the relative gains of our method, demonstrating stable performance across different pretrained source models.

Table 3: Performance comparison across different pretrained ResNet-18 checkpoints on CIFAR10-C.

| Methods | pretrained Model | Local TTA | SFTTA | AFTTA | FATTA | Ours |
|---|---|---|---|---|---|---|
| Tent | Model 1 | $73.10 \pm 0.29$ | $76.23 \pm 0.32$ | $74.97 \pm 0.31$ | $75.60 \pm 0.27$ | $\mathbf{77.52 \pm 0.19}$ |
| | Model 2 | $79.72 \pm 0.08$ | $80.02 \pm 0.09$ | $79.44 \pm 0.12$ | $79.66 \pm 0.07$ | $\mathbf{80.31 \pm 0.08}$ |
| | Model 3 | $71.20 \pm 0.16$ | $76.48 \pm 0.14$ | $74.91 \pm 0.16$ | $78.53 \pm 0.08$ | $\mathbf{79.10 \pm 0.10}$ |
| SAR | Model 1 | $73.09 \pm 0.32$ | $75.31 \pm 0.32$ | $74.15 \pm 0.32$ | $75.31 \pm 0.31$ | $\mathbf{77.70 \pm 0.25}$ |
| | Model 2 | $80.13 \pm 0.10$ | $80.61 \pm 0.10$ | $79.88 \pm 0.11$ | $80.54 \pm 0.09$ | $\mathbf{81.04 \pm 0.08}$ |
| | Model 3 | $73.42 \pm 0.19$ | $77.64 \pm 0.15$ | $75.73 \pm 0.18$ | $79.02 \pm 0.10$ | $\mathbf{79.68 \pm 0.11}$ |
| DeYO | Model 1 | $73.30 \pm 0.34$ | $75.86 \pm 0.26$ | $74.58 \pm 0.23$ | $75.48 \pm 0.26$ | $\mathbf{77.87 \pm 0.27}$ |
| | Model 2 | $81.00 \pm 0.09$ | $81.44 \pm 0.07$ | $80.72 \pm 0.10$ | $81.37 \pm 0.07$ | $\mathbf{81.89 \pm 0.08}$ |
| | Model 3 | $74.80 \pm 0.20$ | $78.23 \pm 0.16$ | $76.39 \pm 0.19$ | $79.45 \pm 0.12$ | $\mathbf{80.12 \pm 0.13}$ |

**Scalability with the Number of Clients.** To evaluate the scalability of our method with a larger client population, we further increase the number of clients to 120 and 180. As shown in Tab. 4, the performance of all methods gradually decreases as the number of clients increases. With more clients sharing the test stream, local observations become more fragmented, while asynchronous interactions become more frequent and diverse, making cross-client adaptation increasingly challenging. Nevertheless, our method consistently achieves the best performance across all evaluated datasets, model architectures, and local TTA methods. Moreover, its performance degrades relatively mildly as the client population grows, indicating better robustness to increasing client scale. This behavior can be attributed to the shared adaptation memory, which continuously preserves useful cross-client knowledge, together with residual write-in that limits the cumulative influence of frequent and redundant updates. Overall, the results demonstrate that our method maintains effective collaborative adaptation under larger-scale asynchronous deployment.

[5] https://github.com/kuangliu/pytorch-cifar

[6] https://drive.usercontent.google.com/download?id=17fmN8eQdLpq2jIMQ_X0IXDPXfI9oVWgq&export=download&confirm=t

Table 4: Performance comparison across datasets and models with different numbers of clients.

| Methods | Dataset & Model | Local TTA | SFTTA | AFTTA | FATTA | Ours |
|---|---|---|---|---|---|---|
| | | | Client Number = 120 | | | |
| Tent | CIFAR10-C+ResNet-18 | 72.65 ± 0.31 | 75.68 ± 0.34 | 74.27 ± 0.34 | 75.00 ± 0.30 | **77.33 ± 0.21** |
| | CIFAR100-C+ResNet-18 | 47.62 ± 0.23 | 51.03 ± 0.25 | 49.63 ± 0.20 | 49.86 ± 0.29 | **53.02 ± 0.34** |
| | ImageNet-C+ResNet-50 | 22.68 ± 0.38 | 24.81 ± 0.50 | 23.49 ± 0.45 | 24.45 ± 0.43 | **27.93 ± 0.23** |
| | ImageNet-C+ViT-B/16 | 36.99 ± 0.28 | 39.25 ± 0.31 | 37.85 ± 0.27 | 38.85 ± 0.24 | **39.88 ± 0.28** |
| SAR | CIFAR10-C+ResNet-18 | 72.64 ± 0.34 | 74.76 ± 0.35 | 73.45 ± 0.35 | 74.71 ± 0.34 | **77.51 ± 0.27** |
| | CIFAR100-C+ResNet-18 | 47.47 ± 0.17 | 49.75 ± 0.27 | 48.39 ± 0.28 | 49.43 ± 0.26 | **52.78 ± 0.35** |
| | ImageNet-C+ResNet-50 | 22.42 ± 0.44 | 24.47 ± 0.50 | 22.99 ± 0.48 | 24.30 ± 0.45 | **27.66 ± 0.24** |
| | ImageNet-C+ViT-B/16 | 36.80 ± 0.30 | 39.01 ± 0.37 | 37.66 ± 0.16 | 38.74 ± 0.24 | **39.62 ± 0.31** |
| DeYO | CIFAR10-C+ResNet-18 | 72.85 ± 0.36 | 75.31 ± 0.29 | 73.88 ± 0.26 | 74.88 ± 0.29 | **77.70 ± 0.29** |
| | CIFAR100-C+ResNet-18 | 48.41 ± 0.13 | 51.15 ± 0.23 | 50.03 ± 0.24 | 49.85 ± 0.27 | **53.48 ± 0.31** |
| | ImageNet-C+ResNet-50 | 22.25 ± 0.12 | 25.98 ± 0.27 | 24.48 ± 0.35 | 24.86 ± 0.62 | **28.02 ± 0.27** |
| | ImageNet-C+ViT-B/16 | 35.86 ± 0.39 | 39.99 ± 0.44 | 38.63 ± 0.59 | 39.16 ± 0.20 | **40.62 ± 0.24** |
| | | | Client Number = 180 | | | |
| Tent | CIFAR10-C+ResNet-18 | 72.25 ± 0.34 | 75.18 ± 0.37 | 73.67 ± 0.37 | 74.45 ± 0.33 | **77.03 ± 0.23** |
| | CIFAR100-C+ResNet-18 | 47.17 ± 0.26 | 50.48 ± 0.28 | 48.98 ± 0.23 | 49.26 ± 0.32 | **52.67 ± 0.36** |
| | ImageNet-C+ResNet-50 | 22.13 ± 0.42 | 24.21 ± 0.54 | 22.84 ± 0.49 | 23.80 ± 0.47 | **27.53 ± 0.27** |
| | ImageNet-C+ViT-B/16 | 36.54 ± 0.32 | 38.70 ± 0.35 | 37.20 ± 0.31 | 38.25 ± 0.28 | **39.53 ± 0.31** |
| SAR | CIFAR10-C+ResNet-18 | 72.24 ± 0.37 | 74.26 ± 0.38 | 72.85 ± 0.38 | 74.16 ± 0.37 | **77.21 ± 0.30** |
| | CIFAR100-C+ResNet-18 | 47.02 ± 0.20 | 49.20 ± 0.30 | 47.74 ± 0.31 | 48.83 ± 0.29 | **52.43 ± 0.38** |
| | ImageNet-C+ResNet-50 | 21.87 ± 0.48 | 23.87 ± 0.54 | 22.34 ± 0.52 | 23.65 ± 0.49 | **27.26 ± 0.28** |
| | ImageNet-C+ViT-B/16 | 36.35 ± 0.34 | 38.46 ± 0.41 | 37.01 ± 0.20 | 38.14 ± 0.28 | **39.27 ± 0.35** |
| DeYO | CIFAR10-C+ResNet-18 | 72.45 ± 0.39 | 74.81 ± 0.32 | 73.28 ± 0.29 | 74.33 ± 0.32 | **77.40 ± 0.32** |
| | CIFAR100-C+ResNet-18 | 47.96 ± 0.16 | 50.60 ± 0.26 | 49.38 ± 0.27 | 49.25 ± 0.30 | **53.13 ± 0.34** |
| | ImageNet-C+ResNet-50 | 21.70 ± 0.16 | 25.38 ± 0.31 | 23.83 ± 0.39 | 24.21 ± 0.66 | **27.62 ± 0.31** |
| | ImageNet-C+ViT-B/16 | 35.41 ± 0.43 | 39.44 ± 0.48 | 37.98 ± 0.63 | 38.56 ± 0.24 | **40.27 ± 0.28** |

**Effect of Client Participation Ratio.** We evaluate client participation ratios of 30%, 60%, and 100% on CIFAR10-C using Tent, SAR, and DeYO as local TTA methods. The results show that our method maintains stable performance across different participation ratios and achieves the highest mean accuracy in all settings. This stability stems from the shared memory's ability to retain adaptation knowledge uploaded by clients, while residual updates suppress the cumulative influence of redundant information. With fewer participating clients, knowledge is accumulated and retained across successive uploads. As more clients participate, additional information can be incorporated without disproportionately amplifying redundant knowledge. This mechanism therefore supports stable cross-client collaboration across different client participation ratios.

Table 5: Performance comparison on CIFAR10-C under different client participation ratios.

| Methods | Participation | Local TTA | SFTTA | AFTTA | FATTA | Ours |
|---|---|---|---|---|---|---|
| Tent | 30 | 75.03 ± 0.87 | 75.88 ± 0.79 | 75.71 ± 0.78 | 75.29 ± 0.92 | **76.65 ± 0.80** |
| | 60 | 75.20 ± 0.55 | 76.10 ± 0.58 | 75.99 ± 0.60 | 75.43 ± 0.56 | **77.25 ± 0.50** |
| | 100 | 73.10 ± 0.29 | 76.23 ± 0.32 | 74.97 ± 0.31 | 75.60 ± 0.27 | **77.52 ± 0.19** |
| SAR | 30 | 75.07 ± 0.84 | 75.05 ± 0.86 | 75.05 ± 0.85 | 74.66 ± 0.74 | **76.80 ± 0.78** |
| | 60 | 75.16 ± 0.55 | 75.13 ± 0.58 | 75.13 ± 0.58 | 74.91 ± 0.56 | **77.42 ± 0.48** |
| | 100 | 73.09 ± 0.32 | 75.31 ± 0.32 | 74.15 ± 0.32 | 75.31 ± 0.31 | **77.70 ± 0.25** |
| DeYO | 30 | 75.35 ± 0.92 | 75.61 ± 0.90 | 75.53 ± 0.93 | 75.10 ± 0.76 | **76.95 ± 0.82** |
| | 60 | 75.49 ± 0.58 | 75.72 ± 0.60 | 75.65 ± 0.60 | 75.30 ± 0.57 | **77.58 ± 0.48** |
| | 100 | 73.30 ± 0.34 | 75.86 ± 0.26 | 74.58 ± 0.23 | 75.48 ± 0.26 | **77.87 ± 0.27** |

## A.3 Algorithm

We provide the pseudocode of our method, which consists of local adaptation, shared knowledge utilization, and asynchronous server-side memory updates. For each test batch $\mathcal{B}_i$, the client performs TTA, extracts the environment descriptor $\mathbf{s}_i$, and uses the cached memory $\widehat{\mathbf{M}}_i$ for prediction when the current environment is matched; otherwise, only the local TTA output is used. The client then constructs $\mathbf{k}(\mathbf{x})$ and $\mathbf{v}(\mathbf{x})$ and uploads them asynchronously with $\mathbf{s}_i$. The server either creates a new memory entry or updates the most similar one via residual write-in and prototype refinement. Algorithm 1 summarizes the overall procedure.

**Algorithm 1:**

**Input:** Fixed shared model $\boldsymbol{\theta}_{\mathrm{g}}$; local models $\{\boldsymbol{\theta}_i\}$; memory bank $\mathcal{M} = \emptyset$; matching threshold $\delta$; fusion weight $\lambda$; prototype update rate $\beta$.
**Output:** Online predictions and updated memory bank $\mathcal{M}$.

**for** *each client* $C_i$ *in parallel* **do**
  **for** *each test batch* $\mathcal{B}_i$ **do**
    Perform local TTA on $\mathcal{B}_i$ to obtain $\boldsymbol{\theta}_i^+$ // Eq. 1
    Extract environment descriptor $\mathbf{s}_i$
    **if** *cached entry exists and environment similarity* $\geq \delta$ **then**
      Predict using local TTA with cached memory correction // Eq. 6
    **else**
      Use local TTA prediction
    **end**
    Extract sample representation $\mathbf{k}(\mathbf{x})$
    Compute correction vector $\mathbf{v}(\mathbf{x})$ // Eq. 7
    Construct adaptation information $\mathcal{U}_i$ and upload it asynchronously
  **end**
**end**
**for** *each arriving upload* $\mathcal{U}_i$ **do**
  Compute similarities between $\mathbf{s}_i$ and existing environment prototypes
  **if** *memory bank is empty or environment similarity* $< \delta$ **then**
    Initialize a new environment memory // Eq. 8
    Add the new environment entry to $\mathcal{M}$
  **else**
    Select the most similar environment $j^\star$
    Update $\mathbf{M}_{j^\star}$ by residual write-in // Eq. 9
    Update environment prototype $\mathbf{p}_{j^\star}$ // Eq. 10
  **end**
  **if** $C_i$ *requests shared knowledge* **then**
    Return the corresponding environment memory to $C_i$
  **end**
**end**

## A.4 Limitations

Although our method can mitigate fast client dominance and error propagation in asynchronous collaboration, it still has certain limitations. To reduce the impact of erroneous shared knowledge on subsequent clients, our method uses shared knowledge only for prediction and does not involve it in local TTA optimization. This isolation mechanism limits error propagation, but also restricts the direct benefit of shared knowledge to local adaptation. Therefore, when local TTA continuously produces unreliable updates under severe distribution shifts, the shared memory can only improve the current prediction and cannot further correct the local model' s adaptation process. Future work will investigate how to safely use shared knowledge to guide local TTA while maintaining reliability.