Reviving Stale Updates: Data-Free Knowledge Distillation for Asynchronous Federated Learning

Baris Askin[†]
Carnegie Mellon University

Holger R. Roth
NVIDIA

Zhenyu Sun[†]
Northwestern University

Carlee Joe-Wong
Carnegie Mellon University

Gauri Joshi
Carnegie Mellon University

Ziyue Xu
NVIDIA

Abstract

Federated learning (FL) enables collaborative model training across distributed clients without sharing raw data, yet its scalability is limited by synchronization overhead. Asynchronous federated learning (AFL) alleviates this issue by allowing clients to communicate independently, thereby improving wall-clock efficiency in large-scale, hardware-heterogeneous environments. However, asynchrony introduces updates computed on outdated global models (*staleness*) that can destabilize optimization and hinder convergence. We propose FedRevive, an AFL framework that revives stale updates through data-free knowledge distillation (DFKD). FedRevive integrates parameter-space aggregation with a server-side DFKD process that transfers knowledge from stale client updates to the current global model without access to data. A meta-learned generator synthesizes pseudo-samples for multi-teacher distillation. A hybrid aggregation scheme combining raw with DFKD updates effectively mitigates staleness while retaining AFL scalability. Experiments on vision and text benchmarks show that FedRevive achieves faster training by up to 29.7% and higher final accuracy by up to 11.4 percentage points than baselines.

NVIDIA FLARE Implementation Project Codes

1 Introduction

Modern artificial intelligence models thrive on large amounts of data, which raises serious concerns about privacy and regulatory compliance [54]. Federated learning (FL) can help address this dilemma by enabling many clients (e.g., smartphones, wearables, hospital servers) to collaboratively train a shared model without uploading raw data [36]. In vanilla FL, the server broadcasts the current global model, clients perform local training on private data, and the server aggregates returned updates to form the next global iterate. This paradigm preserves data locality and widens access to otherwise restricted data, while reducing central storage risks. Moreover, relying only on public datasets is increasingly insufficient, as large models saturate public corpora and demand broader coverage across domains. FL supports both cross-silo (e.g., hospitals and enterprises) and cross-device (e.g., millions of phones) settings, offering a practical path to leverage diverse private data at scale [44, 22].

In *synchronous* FL, the server waits for all (or a selected subset of) participating clients to complete their local updates before aggregation [36]. Although this strategy aligns updates to the same global

[†] Work was done during an internship at NVIDIA.

Correspondence to: Baris Askin <baskin@andrew.cmu.edu>, Holger Roth <hroth@nvidia.com>, Ziyue Xu <ziyuex@nvidia.com>.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

state, it suffers from the *straggler problem*: slow or unavailable clients delay the entire round, leading to underutilized resources and poor scalability under heterogeneous availability and compute capacity [50]. *Asynchronous* FL (AFL) alleviates this bottleneck by allowing the server to incorporate each client update upon arrival, without waiting for others [50, 38]. Clients train and communicate independently, and the server continuously updates the global model. This design improves wall-clock efficiency and makes AFL appealing for cross-device deployments such as mobile networks and for cross-silo collaborations where institutional availability fluctuates.

The flexibility of AFL, however, comes with a major drawback: *update staleness*. Since clients may train on older global checkpoints, the server receives delayed updates that no longer match the current optimization state, which can distort training dynamics and degrade performance, especially under not independently and identically distributed (non-IID) data [50, 38, 2, 43]. This issue becomes more pronounced as the system scales, since larger deployments naturally exhibit a wider spread of client runtimes and delays [21]. Existing AFL methods mitigate staleness by buffering or smoothing arriving updates [38] or by explicitly correcting or down-weighting delayed updates [5]. While effective in certain regimes, these strategies can slow server progress or suppress information carried by slower clients. Even a delayed update can encode rich signals about a client's local data. The key challenge is therefore not only to control the harmful drift induced by staleness, but also to retain the useful knowledge in delayed client models. This motivates the following question:

*Can we **revive** stale updates in asynchronous FL to extract useful knowledge for improved scalability?*

Our idea: distill knowledge from stale updates. We answer this question by combining parameter-space aggregation with *data-free knowledge distillation* (DFKD). Knowledge distillation (KD) transfers information from one or more teachers to a student via softened targets or intermediate representations [17]. Data-free variants remove the assumption that a public dataset is available for distillation [31], which aligns with practical FL deployments. At a high level, our method treats received client models as teachers and the current server model as the student, and uses server-side DFKD to extract transferable knowledge from delayed client states. Then, by coupling the regular parameter-space update with a DFKD update, our method mitigates the adverse effects of staleness. Integrating DFKD into AFL is nontrivial because (i) no data are typically available on the server, (ii) stale client models can be heterogeneous and only partially trained, and (iii) the additional server-side processing should remain moderate. FedRevive targets cross-device FL regimes where stale updates due to a large number of clients harm optimization.

Related work. AFL builds on earlier work on asynchronous distributed stochastic gradient descent [9, 29]. In federated settings, asynchronous aggregation has been explored as a practical remedy for stragglers by integrating updates upon arrival rather than at synchronized rounds [50, 38, 51, 7, 43]. To mitigate staleness, prior work explores buffering or delay-aware reweighting strategies that control instability but can also attenuate information from slower clients [38, 6, 46, 33, 60, 10, 8]. In parallel, knowledge distillation (KD) is a tool for transferring predictive structure from teachers to students [17], and has been adapted to FL to mitigate heterogeneity, support model-architecture mismatch, and improve privacy via logits or ensemble-based supervision [26, 30, 41, 11]. These FL-KD methods typically assume access to a shared dataset for distillation, either on the server or on clients [24, 19, 28]. DFKD removes this dependency by synthesizing pseudo-samples for distillation without real data [31, 35, 12], and several recent works adopt DFKD within synchronous FL to avoid public data while improving robustness under non-IID data [13, 58, 62, 48]. However, the use of KD in asynchronous FL remains largely unexplored. Some efforts rely on public data assumptions often unrealistic in FL deployments [32]. Unlike synchronous FL, where distillation leverages relatively stable teacher models, AFL distills from client models misaligned with the current server state. Although FedRevive builds on established DFKD techniques, it introduces an AFL-specific update that distills predictive knowledge from stale client models into the current server model. Adapting distillation to continually changing, partially trained, and heterogeneous teachers poses nontrivial design challenges, which we carefully address through a persistent meta-learned generator, class-aware multi-teacher distillation, and staleness-dependent hybrid aggregation. Appendix A presents an extended related work discussion.

Contributions. The main contributions of the paper are summarized as follows:

- We propose FedRevive, an AFL framework that *revives* stale client updates by combining parameter aggregation with server-side, multi-teacher DFKD.

• We carefully design a DFKD pipeline tailored to AFL without requiring any extra client metadata beyond regular FL updates. We leverage multi-teacher distillation with computation-light process to operationalize our idea.
• We demonstrate improved speed and accuracy over the baseline methods under realistic simulations with image and text-domain experiments, achieving faster time-to-target accuracy and better final performance, and support the proposed method with ablation studies on hyperparameters, algorithmic components and overhead analysis.

Our results indicate that stale client updates still carry rich, transferable knowledge. By extracting that knowledge in a data-free manner and fusing it with parameter-space aggregation, AFL can retain its scalability advantages without sacrificing model quality.

2 FL Problem & Proposed Method

Problem formulation. We consider an FL setup with N clients collaboratively training a global model $\mathbf{x} \in \mathbb{R}^d$. Each client i has $f_i : \mathbb{R}^d \to \mathbb{R}$, a local objective function defined on private data. Local objectives are different across clients due to heterogeneity in data distribution. The goal of FL is to find $\mathbf{x}$ minimizing the global objective, formulated as the average of local losses, $\frac{1}{N}\sum_{i\in[N]} f_i(\mathbf{x})$.

Algorithm 1 Generic AFL Framework

Server Process:
1: **Inputs:** Initial model $\mathbf{x}^{(0)}$, server and client learning rates η_s and η_c, #local iterations K, #rounds T, #selected active clients N_a.
2: Send $\mathbf{x}^{(0)}$ to N_a randomly selected active clients.
3: **for** $t = 0$ **to** $T-1$ **do**
4: **Wait** until the server receives a τ_i^t-stale updated model $\widetilde{\mathbf{x}}_i^{(t-\tau_i^t)}$ from client i.
5: Client update $\mathbf{\Delta}_i^{(t-\tau_i^t)} \leftarrow \widetilde{\mathbf{x}}_i^{(t-\tau_i^t)} - \mathbf{x}^{(t-\tau_i^t)}$.
6: **Process update with Eq. (1), (2), or (3).**
7: Assign local training to a randomly selected active client with the updated model $\mathbf{x}^{(t+1)}$.
8: **end for**

Client i's Local Training (Async. and Parallel):
9: **Input:** Received server model $\mathbf{x}$.
10: Initialize $\widetilde{\mathbf{x}} \leftarrow \mathbf{x}$.
11: **for** $k = 1$ **to** K **do**
12: $\widetilde{\mathbf{x}} \leftarrow \widetilde{\mathbf{x}} - \eta_c \widetilde{\nabla} f_i(\widetilde{\mathbf{x}})$ {A stoch. grad. update}
13: **end for**
14: **Return** $\widetilde{\mathbf{x}}$ to the server.

Asynchronous framework. We first describe a generic AFL framework that encompasses both existing asynchronous schemes [50, 38] and our proposed method, FedRevive. The overall process is summarized in Algorithm 1. After model initialization (line 2), the server sends $\mathbf{x}^{(0)}$ to N_a randomly selected active clients for local training. Then, for each server step t, the server waits for the next returned model $\widetilde{\mathbf{x}}_i^{(t-\tau_i^t)}$ from some client i (line 4), where τ_i^t denotes the staleness, namely, how many server updates occurred since the client was asked to perform local training. The server processes the client update via an aggregation rule (line 6), and dispatches the updated model $\mathbf{x}^{(t+1)}$ to a randomly selected active client (line 7) to sustain parallelism. Different AFL variants share this control flow and differ only in the aggregation step, that is, how the server maps $(\mathbf{x}^{(t)}, \mathbf{\Delta}_i^{(t-\tau_i^t)})$ to $\mathbf{x}^{(t+1)}$.

Vanilla and buffered AFL. Vanilla AFL algorithm [50] updates the global model immediately when a client update arrives. Accordingly, the aggregation step in Algorithm 1 (line 6) becomes:

$$\mathbf{x}^{(t+1)} \leftarrow \mathbf{x}^{(t)} + \eta_s \, \mathbf{\Delta}_i^{(t-\tau_i^t)} \tag{1}$$

While this rule is simple and maximally asynchronous, it makes staleness accumulate fast. Every time the server updates $\mathbf{x}$, all clients that are still training are effectively one step further behind. When staleness, τ_i^t, is large, the returned update can yield a biased step and lead to oscillations or divergence. FedBuff [38] mitigates this by buffering updates before applying them. The server maintains a buffer $\mathcal{B}$ of size B, stores incoming client updates, and aggregates only when B updates have been collected. The aggregation step in Algorithm 1 (line 6) becomes:

$$\mathcal{B} \leftarrow \mathcal{B} \cup \{\mathbf{\Delta}_i^{(t-\tau_i^t)}\}$$
$$\textbf{if } |\mathcal{B}| = B: \ \mathbf{x}^{(t+1)} \leftarrow \mathbf{x}^{(t)} + \frac{\eta_s}{B}\sum_{\mathbf{\Delta}\in\mathcal{B}}\mathbf{\Delta} \ \text{ and } \ \mathcal{B} \leftarrow \emptyset, \quad \textbf{else: } \mathbf{x}^{(t+1)} \leftarrow \mathbf{x}^{(t)} \tag{2}$$

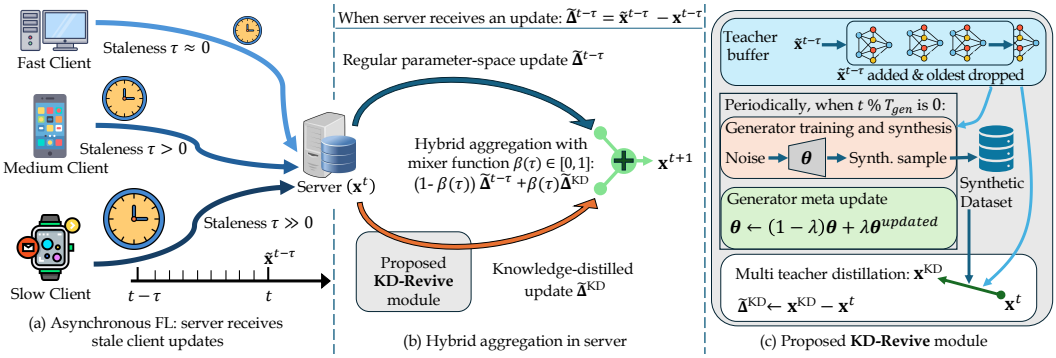


Figure 1: **Proposed FedRevive framework.**

The buffered strategy smooths the staleness effect by reducing the update frequency. Yet, as the buffer size B increases, the infrequent server updates slow down the training. Next, we describe our method, FedRevive, mitigating the staleness without reducing the server model update frequency.

2.1 Overview of FedRevive

FedRevive addresses the staleness problem by leveraging knowledge-distillation. Although directly aggregating the parameters of stale updates can result in training instabilities, the received update still contains informative signals from client data. We augment the standard parameter-space update with a data-free knowledge distillation (DFKD) signal extracted from the updated client model.

FedRevive aggregation. In FedRevive, the server adaptively fuses the conventional parameter-space update with a knowledge-distilled update to alleviate the adverse effect of staleness. When the server receives a τ_i^t-stale client update $\boldsymbol{\Delta}_i^{(t-\tau_i^t)}$, it first obtains a knowledge-distilled update through a DFKD process. The server then combines it with original parameter update through an adaptive weighting function $\beta(\tau_i^t) \in [0,1]$ controlling the reliance on the knowledge-distilled signal according to the degree of staleness. The aggregation step in Algorithm 1 (line 6) becomes:

$$\begin{aligned} &\boldsymbol{\Delta}_i^{\mathrm{KD}} \leftarrow \text{KD-Revive}(\tilde{\mathbf{x}}_i^{(t-\tau_i^t)},\, \mathbf{x}^{(t)})\,, \\ &\mathbf{x}^{(t+1)} \leftarrow \mathbf{x}^{(t)} + \eta_s \left[(1-\beta(\tau_i^t))\boldsymbol{\Delta}_i^{(t-\tau_i^t)} + \beta(\tau_i^t)\boldsymbol{\Delta}_i^{\mathrm{KD}}\right] \end{aligned} \tag{3}$$

where KD-Revive$(\cdot)$ encapsulates our proposed DFKD mechanism. The function $\beta : \mathbb{N}_0 \to [0,1]$ is a non-decreasing mapping of the staleness τ_i^t, such that the server increasingly relies on the KD-based update as the client update becomes more outdated. Intuitively, when $\tau_i^t \approx 0$, the client model is nearly up-to-date, and the raw parameter update remains reliable, so $\beta(\tau_i^t) \approx 0$; conversely, when τ_i^t is large, $\beta(\tau_i^t) \approx 1$, distilled knowledge extracted from the stale client model is weighted more. The choice of $\beta(\tau_i^t)$ is detailed in Section 3.

Challenges of integrating KD in AFL. Integrating KD into AFL is complicated by the absence of shared data at the server, continually changing teacher models, and client heterogeneity. We first provide an overview of the main issues and how FedRevive addresses them.

(a) Lack of distillation data. Standard KD practice requires data to mediate the knowledge transfer between the teacher and the student models. In FL, a public or server-accessible dataset is rarely available due to privacy and communication constraints. Our KD-Revive module performs data-free KD and synthesizes pseudo-inputs guided jointly by the incoming client model and the current server model, enabling distillation without access to real data. Note that FedRevive only requires standard federated updates as in FedAvg [36] with no additional information from client. Generated pseudo-samples do not provide extra information and are simply intermediate computations for extracting knowledge from already-received updates. Client updates arise after multiple local optimization steps and are mixed with asynchronous delay. Empirically, this does not provide enough signal to synthesize realistic client images, even though it carries intermediate predictive structure useful for knowledge transfer. We provide a discussion and representative synthesized samples in Appendix I.

(b) Dynamic teachers and need for lightweight operations. Teacher models in KD or DFKD setups are typically frozen and fully trained oracles. In FL, however, client models continually evolve and are only partially trained. Transferring knowledge from such transient teachers is non-trivial. Moreover, the overhead due to the distillation process must remain computationally light to avoid a new bottleneck. We adopt a lightweight meta-learning strategy inspired by [12], maintaining a single generator that evolves throughout training. This generator efficiently adapts every T_{gen} rounds to the incoming client models and synthesizes pseudo-samples to extend the synthetic KD dataset ($\mathcal{D}_{\text{KD}}$), initialized empty in the beginning of the training.

(c) Data heterogeneity across clients. Due to limited data size and non-IID distributions, each client's update tends to overfit to its local data, which leads to biased knowledge transfer. We mitigate this via *multi-teacher distillation*. The server maintains a buffer $\mathcal{C}$ of the c most recently received client models and distills from all buffered teachers jointly. To better exploit client-specific information, the server derives proxy class proportions of clients from the received updates *without additional information*. Our results do not assume access to extra information beyond regular updates.

Per-client class-proportion proxies from regular updates. To weight buffered teachers in a class-aware manner, we associate each client i with a *proxy* class-proportion vector $\hat{\boldsymbol{\pi}}_i \in \mathcal{S}^C$ where C denotes the number of classes, $\mathcal{S}^C$ denotes C-simplex, and $\hat{\pi}_i^j = [\hat{\boldsymbol{\pi}}_i]_j$ denotes the proxy proportion of class j on client i. The server computes $\hat{\boldsymbol{\pi}}_i$ using only the regular updates and server-side forward passes, requiring *no additional client computation or information*.

Let $\mathbf{y}_i^{(u)}$ denote the u-th model upload received from client i (ordered by server reception), and consider the first T_{est} uploads $\{\mathbf{y}_i^{(u)}\}_{u=1}^{T_{\text{est}}}$. For each upload, the server probes $\mathbf{y}_i^{(u)}$ with a batch of random inputs $\mathbf{r}$ (Gaussian noise for vision models, random token sequences for language models), and computes a per-upload proxy:

$$\widetilde{\boldsymbol{\pi}}_i^{(u)} = (1/|\mathbf{r}|)\textstyle\sum_{r\in\mathbf{r}} \sigma(\mathbf{y}_i^{(u)}(r))\,, \quad u = 1, \ldots, T_{\text{est}},$$

where σ denotes the softmax function. The server then averages these vectors and keeps this proxy fixed for the remainder of training, $\hat{\boldsymbol{\pi}}_i \leftarrow \frac{1}{T_{\text{est}}}\sum_{u=1}^{T_{\text{est}}} \widetilde{\boldsymbol{\pi}}_i^{(u)}$, and uses running-average for $u < T_{\text{est}}$. This adds negligible server overhead since it only requires forward passes and T_{est} is small (2 in our experiments), while imposing no additional client-side computation. Although this proxy is noisy, FedRevive is empirically robust to its errors (Appendix B, Fig. 5). Importantly, it is used only as a coarse dominant-class cue to guide synthesis and teacher weighting, rather than to recover exact client histograms. In Appendix B, we report KL divergence of proxy proportions, top-1 and top-2 dominant-class recovery, a controlled study across label-skew levels, and ablation experiments comparing estimated and true class proportions.

2.2 Detailed Steps of KD-Revive

When the server receives an updated model $\widetilde{\mathbf{x}}_i^{(t-\tau_i^t)}$ from client i (Algorithm 1, line 4), it executes KD-Revive to compute a data-free knowledge-distilled update (Eq. (3)). This procedure consists of four steps: (i) updating the model buffer, (ii) periodic generator training and synthesis, (iii) meta-update of the generator, and (iv) multi-teacher distillation. FedRevive framework is depicted in Fig. 1.

(1) Update the model buffer. The server maintains a KD buffer $\mathcal{C}$ storing c most recently received client models to use as teachers. Receiving $\widetilde{\mathbf{x}}_i^{(t-\tau_i^t)}$, the buffer is updated as:

$$\mathcal{C} \leftarrow \mathcal{C} \cup \{\widetilde{\mathbf{x}}_i^{(t-\tau_i^t)}\} \backslash \{\text{oldest model in } \mathcal{C}\}.$$

The rolling buffer ensures that the distillation process utilizes recent and diverse client knowledge. Let $\mathcal{I}_t$ denote the set of client indices whose models are currently buffered. For class-aware weighting, we normalize the proxy client proportions across buffered teachers, for each class j and client $i \in \mathcal{I}_t$: $\hat{\omega}_i^j \triangleq \hat{\pi}_i^j / \sum_{k\in\mathcal{I}_t} \hat{\pi}_k^j$ so that $\sum_{i\in\mathcal{I}_t} \hat{\omega}_i^j = 1$ for every class j. We use $\hat{\omega}_i^j$ to weight teacher contributions in subsequent steps.

(2) Periodic generator training and synthesis. A generator $G_{\boldsymbol{\theta}}$ with parameters $\boldsymbol{\theta}$ is initialized once at the start of training. To keep server overhead small, the server runs generator training and

synthesis only once every T_{gen} server updates; in intermediate rounds, it uses the latest $\mathcal{D}_{\text{KD}}$ and proceeds directly to Step (4). In a synthesis iteration, we sample latent vectors $\mathbf{z} = \{\mathbf{z}_b\}_{b=1}^{B}$ and target labels $\mathbf{y}_{\mathbf{z}} = \{y_b\}_{b=1}^{B}$ (uniformly at random over classes), and train $(\mathbf{z}, \boldsymbol{\theta}')$ for K_{synth} iterations, where $\boldsymbol{\theta}'$ is initialized by $\boldsymbol{\theta}' \leftarrow \boldsymbol{\theta}$:

$$(\mathbf{z}, \boldsymbol{\theta}') \leftarrow (\mathbf{z}, \boldsymbol{\theta}') - \eta_{\text{synth}} \nabla_{(\mathbf{z}, \boldsymbol{\theta}')} \mathcal{L}_{\text{synth}}\big(G_{\boldsymbol{\theta}'}(\mathbf{z})\big).$$

Let $\mathbf{r}_b = G_{\boldsymbol{\theta}'}(\mathbf{z}_b)$ be a synthetic input, and denote $\tilde{\mathbf{p}}_{k,b} = \sigma(\widetilde{\mathbf{x}}_k(\mathbf{r}_b))$ and $\mathbf{p}_b = \sigma(\mathbf{x}^{(t)}(\mathbf{r}_b))$ as the predictive distribution of teacher k in $\mathcal{C}$ and student (current server model), respectively. The synthesis loss is defined as:

$$\mathcal{L}_{\text{synth}} = \frac{1}{B} \sum_{b=1}^{B} \sum_{k \in \mathcal{I}_t} \hat{\boldsymbol{\omega}}_k^{y_b} \Big[\underbrace{\alpha_{\text{target}}\, \text{Cross-Entropy}\big(\tilde{\mathbf{p}}_{k,b},\, y_b\big)}_{\mathcal{L}_{\text{target}}\text{: target loss}} + \underbrace{\alpha_{\text{feature}}\, \|\phi(\widetilde{\mathbf{x}}_k, \mathbf{r}_b) - \bar{\phi}(\widetilde{\mathbf{x}}_k)\|^2}_{\mathcal{L}_{\text{feature}}\text{: feature loss}} - \underbrace{\alpha_{\text{adv}}\, \mathbf{1}\{\hat{c}_{k,b} = \hat{c}_b\}\, \text{KL}\big(\tilde{\mathbf{p}}_{k,b} \,\|\, \mathbf{p}_b\big)}_{\mathcal{L}_{\text{adv}}\text{: adversarial loss}} \Big], \tag{4}$$

where $\phi(\widetilde{\mathbf{x}}_k, \mathbf{r}_b)$ extracts intermediate layer statistics from teacher k on input $\mathbf{r}_b$, and $\bar{\phi}(\widetilde{\mathbf{x}}_k)$ denotes the corresponding running statistics stored in $\widetilde{\mathbf{x}}_k$. The choice of ϕ depends on the modality and model architecture. Defining $\hat{c}_{k,b} = \arg\max_j (\tilde{\mathbf{p}}_{k,b})_j$ and $\hat{c}_b = \arg\max_j (\mathbf{p}_b)_j$, $\mathcal{L}_{\text{adv}}$ discourages trivial synthetic data where the teacher and student classification match, by amplifying discrepancies. Lowest-$\mathcal{L}_{\text{synth}}$ batch in K_{synth} iterations is added to $\mathcal{D}_{\text{KD}}$ and oldest samples are discarded if it is full:

$$\mathcal{D}_{\text{KD}} \leftarrow \mathcal{D}_{\text{KD}} \cup \{(\mathbf{r}_b, y_b)\}_{b=1}^{B}.$$

(3) Meta-update of the generator. The meta-update is done only in synthesis rounds (every T_{gen} server rounds), after Step (2). Reptile-style update is performed on the meta-parameters [12]:

$$\boldsymbol{\theta} \leftarrow (1 - \lambda)\boldsymbol{\theta} + \lambda \boldsymbol{\theta}',$$

where $\lambda \in (0, 1)$ is the step size. Other optimizers (e.g., momentum or Adam) can likewise be used to update $\boldsymbol{\theta}$ using the same pseudo-gradient direction $(\boldsymbol{\theta}' - \boldsymbol{\theta})$. This update lets $G_{\boldsymbol{\theta}}$ adapt to changing teacher models without catastrophic drift, preserving generalizable knowledge during federated training. Meta-learning stabilizes synthetic-data generation under asynchronous updates.

(4) Multi-teacher distillation. We distill from buffered teachers into a student $\mathbf{x}_s$, initialized as $\mathbf{x}_s \leftarrow \mathbf{x}^{(t)}$. At each distillation step, we iterate over buffered teachers $k \in \mathcal{I}_t$ and draw a teacher-aware minibatch $\mathcal{B}_k$ from $\mathcal{D}_{\text{KD}}$ using class-weighted sampling based on $\hat{\boldsymbol{\pi}}_k$. Concretely, each element $(\mathbf{r}, y) \in \mathcal{B}_k$ is obtained by first sampling a label $y \sim \hat{\boldsymbol{\pi}}_k$ and then sampling $\mathbf{r}$ uniformly from $\{(\mathbf{r}', y') \in \mathcal{D}_{\text{KD}} : y' = y\}$. The student is trained to minimize KL divergence to match the teachers:

$$\mathcal{L}_{\text{KD}} \triangleq \frac{1}{c} \sum_{k \in \mathcal{I}_t} \frac{1}{|\mathcal{B}_k|} \sum_{(\mathbf{r}, y) \in \mathcal{B}_k} \text{KL}\Big(\sigma\big(\widetilde{\mathbf{x}}_k(\mathbf{r})\big) \Big\| \sigma\big(\mathbf{x}_s(\mathbf{r})\big)\Big),$$

and the student is updated with $\mathbf{x}_s \leftarrow \mathbf{x}_s - \eta_{\text{KD}} \nabla_{\mathbf{x}_s} \mathcal{L}_{\text{KD}}$ for K_{KD} steps. Finally, we define the distilled update as $\boldsymbol{\Delta}^{\text{KD}} = \mathbf{x}_s - \mathbf{x}^{(t)}$, used in the aggregation rule Eq. (3).

2.3 Overhead Analysis

FedRevive adds server-side computation from periodic generator training and multi-teacher distillation. Compared with FedAvg averaging, this overhead is non-trivial, but it is server-only and leaves communication and client computation unchanged. In AFL regimes where client computation and communication dominate wall-clock time [22, 61], the extra server work is typically modest in practice. Periodic synthesis and a small number of distillation steps further limit the cost. For a real deployment setting, server computation can also be pipelined with client progress: upon receiving an update, the server may dispatch the next local-training request using the current global state while executing synthesis, distillation, and aggregation asynchronously, which is discussed in Appendix C. This trades blocking time for additional staleness, but does not remove the server's throughput limit. If arrivals exceed available capacity, practical options include reducing K_{synth} or K_{KD}, increasing

T_{gen}, or skipping KD for selected arrivals by setting $\beta = 0$ for those updates in Eq. (3). The choices of K_{synth}, K_{KD}, and T_{gen} are discussed and evaluated in Section 3 and Appendix C, D.1. Appendix C reports FLOPs/runtime breakdown. Fig. 3 presents a sensitivity experiment with different KD budgets. Since the effective server blocking time depends on the server accelerator, implementation parallelism, and communication regime, we evaluate total wall-clock time under a range of realistic server/client speed ratios. Section 3 reports the client-computation/communication dominated AFL setting where server KD is overlappable, while Appendix D.1 evaluates wall-clock performance after adding measured blocking server overhead under several hardware configurations. When the server is relatively weak, the wall-clock gains may be smaller, although the accuracy improvements remain. In many FL deployments, however, the server is substantially more capable than the participating clients, so FedRevive is particularly well suited to such settings. FedRevive matches FedAvg communication cost and adds only bounded memory for the teacher buffer and generator, independent of clients and rounds (Appendix H).

3 Experiments and Results

We evaluate our proposed method and baselines on both image and text classification tasks under diverse system conditions. All experiments assume a large-scale FL environment with 1000 clients, where only 10% are active at any time, selected uniformly at random, reflecting the intermittent availability of edge devices. To induce data heterogeneity, we follow Dirichlet partitioning [1], unless the dataset is naturally partitioned. High data heterogeneity, with concentration parameter $\alpha = 0.5$, is used in the main experiments, and Appendix E presents results with milder heterogeneity ($\alpha = 1$).

Since our focus is on AFL, all results are presented in simulated wall-clock time, accounting for realistic computation and communication delays. To simulate system heterogeneity, we categorize clients into *slow*, *medium*, and *fast* groups by computation and communication speeds. Each client is assigned random delay parameters based on its group. When a client participates in training, its local computation and communication durations are sampled from its random generator, ensuring stochastic time variations across rounds. This system heterogeneity across clients and rounds reflects real environments. Additional details are in Appendix D. Main results assume non-blocking server computation; Appendix D.1 confirms the gains persist with measured overhead on practical strong-server setups. Results are averaged over three random seeds with shaded regions showing standard error. Hyperparameters are tuned on validation results. As asynchronous methods often fluctuate, we plot running-average performance until the peak for clarity. Appendix F provides raw curves.

3.1 Datasets, Models, and Baselines

We benchmark on image and text classification tasks. **CIFAR-10** and **CIFAR-100** are 10- and 100-class RGB image classification datasets, respectively, and use ResNet-18 with 11.2M parameters [16]. **FEMNIST** is a 62-class grayscale handwritten character classification dataset with a natural user-level partition across approximately 3500 writers [4]; we use a multi-layer CNN with 421k parameters. **20NewsGroups** is a 20-class news topic classification dataset, for which we use a pretrained T5-small transformer with 35.6M parameters [42]. Except for FEMNIST, we simulate 1000 clients by repeatedly sampling the dataset with 350 samples per client. For FEMNIST, we randomly select 1000 users from the natural partition as clients. Hyperparameter choices are detailed in Appendix J.

We compare FedRevive with representative baselines and ablation variants as summarized below:

- **FedAvg:** The vanilla synchronous FL algorithm [36]. The server waits for all active clients before aggregation, which eliminates staleness, but the round time can be dominated by stragglers [50].
- **FedBuff (Eq. (2)):** A buffered asynchronous variant of FedAvg [38]. The server accumulates incoming client updates in a buffer and aggregates once the buffer is full. When the buffer size is 1, FedBuff reduces to standard asynchronous FedAvg [50] in Eq. (1). Since we tune the buffer size including 1, we report stronger FedBuff as the representative asynchronous baseline.
- **FedFa:** An AFL baseline [52], stabilizing stale updates with a sliding window over recent updates.
- **FedAC:** An AFL baseline [57], with buffered prospective momentum aggregation. For fair comparison of all methods, the client-side operations are kept the same as in other methods.
- **FADAS:** An AFL baseline [49], proposing adaptive asynchronous federated optimization incorporating delayed client updates into an Adam-like server update.

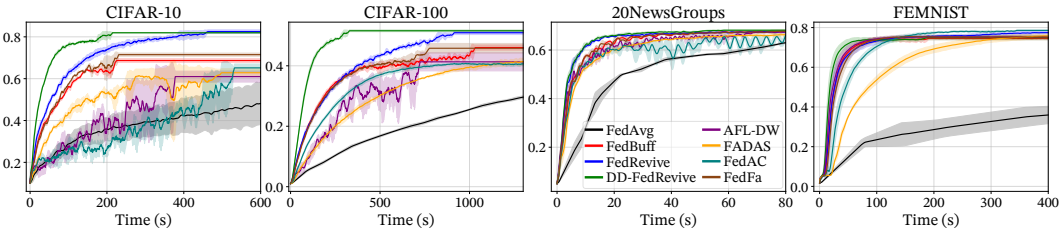


Figure 2: **Test accuracy versus simulated time** on CIFAR-10, CIFAR-100, 20NewsGroups, and FEMNIST. On CIFAR-10, FedRevive converges faster and reaches higher final accuracy than the baselines. On the harder CIFAR-100 task, its advantage becomes more pronounced later in training, when client models provide stronger teachers. On 20NewsGroups, FedRevive converges faster than the baselines. On FEMNIST, asynchronous methods show similar overall convergence due to the relative simplicity of the task, but FedRevive attains among the highest final accuracy.

• **FedRevive (Eq. (3)):** Our proposed method. $K_{\text{synth}} = 2$, $K_{\text{KD}} = T_{\text{gen}} = 10$, and $c = 8$ are used in all experiments unless otherwise stated.
• **DD-FedRevive:** A fully equipped ablation variant, Data-Driven-FedRevive, where the server has access to a disjoint auxiliary dataset of size 16.7% of the total training samples across clients, sampled from the overall client distribution. KD is performed directly on this data, i.e., FedRevive without generator-based synthesis. DD-FedRevive assumes client label proportions are known here rather than estimated. This setting is included only for ablation due to its impractical assumptions.
• **AFL-DW:** Prior asynchronous FL work [6, 46, 33] down-weights stale updates to mitigate their adverse effect. To test whether FedRevive improves performance due to the KD update rather than stale-update down-weighting alone, we include AFL-Down Weighting (AFL-DW) as an ablation without synthesis or distillation. Concretely, we modify Eq. (3) to $\mathbf{x}^{(t+1)} \leftarrow \mathbf{x}^{(t)} + \eta_s \big(1 - \beta(\tau_i^t)\big) \boldsymbol{\Delta}_i$, while keeping all other hyperparameters and $\beta(\cdot)$ identical to FedRevive.

3.2 Main Results and Ablation Experiments

Fig. 2 and Table 1 show training curves, test accuracy and time-to-target results of main experiments.

Image task results. For CIFAR-10, CIFAR-100, and FEMNIST, the synthesis module in FedRevive uses the same lightweight generator architecture as [12], which maps latent noise to an image via a linear projection followed by a CNN block. In all experiments, synchronous baseline FedAvg incurs long wall-clock time because the server waits for all selected clients, which limits practical convergence speed despite avoiding staleness.

Table 1: Final test accuracy and time-to-target, reported as accuracy% (time), where time is the mean simulated time to reach 85% of FedBuff peak accuracy. Per dataset, the best two values are bold and the third best is underlined, separately for accuracy and time. FedRevive provides the strongest accuracy and speed on most datasets.

Method	CIFAR-10	CIFAR-100	20NewsGroups	FEMNIST
FedAvg	63.18 (631.1)	<u>45.73</u> (>1000)	**68.46** (51.2)	63.62 (>1000)
FedBuff	66.23 (96.7)	44.46 (320.7)	<u>67.93</u> (13.5)	74.43 (<u>38.0</u>)
AFL-DW	53.62 (—)	37.89 (525.1)	67.40 (14.3)	75.07 (**35.3**)
FADAS	59.89 (214.7)	31.09 (—)	67.20 (18.1)	<u>76.90</u> (146.0)
FedAC	59.71 (458.9)	43.39 (540.0)	67.14 (16.6)	**78.34** (70.5)
FedFa	<u>70.08</u> (<u>90.3</u>)	45.18 (<u>318.3</u>)	**68.00** (<u>11.1</u>)	75.43 (44.8)
DD-FedRevive	**79.79** (**30.2**)	**50.01** (**99.6**)	<u>67.93</u> (**7.4**)	74.56 (**25.6**)
FedRevive	**81.46** (**69.9**)	**50.09** (**301.1**)	67.46 (**7.8**)	**77.42** (39.9)

On CIFAR-10, FedRevive reaches higher final accuracy than AFL baselines while converging faster. This gain comes from continually transferring knowledge from buffered stale client models through DFKD, so each received update contributes more effectively than parameter aggregation alone. DD-FedRevive, the ablation variant with access to a small public dataset for distillation, exhibits the fastest convergence due to the immediate availability of real data. However, FedRevive surpasses DD-FedRevive in final performance, as the fixed public dataset in DD-FedRevive may lead to partial overfitting, while the synthetic data in FedRevive adaptively evolves throughout the training, yielding richer and more diverse knowledge transfer. Finally, the performance of AFL-DW remains considerably lower than that of FedRevive, confirming that the improvement of our method arises not primarily from down-weighting stale updates but from the proposed DFKD mechanism itself.

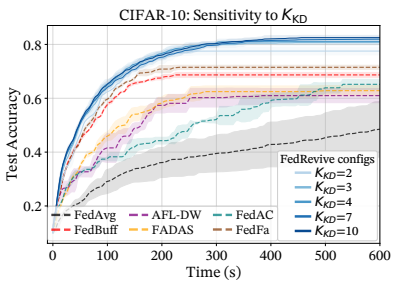


Figure 3: **Sensitivity of FedRevive to the number of KD steps K_{KD} on CIFAR-10,** showing that the gain does not depend on heavy server-side computation.

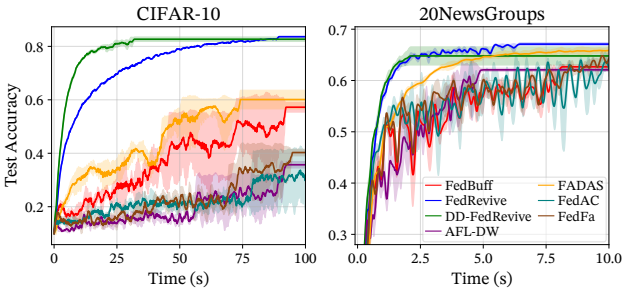


Figure 4: **Robustness to a delay distribution.** Test accuracy on CIFAR-10 and 20NewsGroups under an alternative delay scheme with a shifted staleness distribution. Hyperparameters are tuned under the default delay model. FedRevive remains robust, while baselines degrade significantly.

On the harder CIFAR-100 task with 100 classes, class-conditioned synthesis and distillation are more challenging, especially early in training when the teachers are still weak. Once client models become sufficiently accurate, the advantage of FedRevive becomes clearer and the gap with AFL baselines widens. The ablation variant DD-FedRevive converges fastest due to immediate access to real server-side data and client label proportions. FedRevive achieves the highest final accuracy. Finally, on FEMNIST, stale updates are less detrimental, likely due to simpler visual patterns, so AFL baselines exhibit similar convergence. Even in this regime, FedRevive is competitive.

Text task results. For the text-domain task, 20NewsGroups, we adopt the T5-small architecture [42] as the backbone and attach a lightweight classification head composed of linear layers. The head reads the final hidden state of a prepended dummy prediction token and outputs class logits. We truncate inputs to length 256 after adding this token. For discrete text inputs, we replace the image generator with a prompt-based synthesis mechanism. Unlike images, text is discrete and highly structured; and prior data-free distillation for language often relies on external or generated corpora [40, 53, 47]. In FedRevive, rather than training a generator, we meta-learn four soft prompt vectors inserted immediately after the prediction token, following soft prompt learning [25, 14]. For the remaining latent prompt positions, we initialize from embeddings of short high-temperature synthetic snippets from a small LM [59], using them only as a linguistic warm start for optimization rather than as client data, labels, or an auxiliary training corpus. Figure 2 (third panel) shows test accuracy versus simulated time on 20NewsGroups. FedRevive converges faster than the baselines. We verify that the LM-generated snippets do not provide meaningful task leakage. Centralized training on only the initialization snippets achieves a best test accuracy of $48.08\% \pm 0.22\%$ on 20NewsGroups, which is far below the federated performance of all methods reported in Table 1. Thus, the initialization snippets only serve as a weak linguistic prior and a neutral initialization that stabilizes the synthetic optimization, rather than a competitive task dataset. In this setup, the generator parameters $\boldsymbol{\theta}$ correspond to four meta-learned soft prompt vectors, while the latent variable $\mathbf{z}$ denotes the remaining embedding vectors initialized from the synthetic snippets.

Robustness to shift in delay schemes. All main experiments use a fixed stochastic delay model for client-side computation and communication (details in Appendix D), inducing a long-tailed staleness distribution in Fig. 6. We tune all hyperparameters under this default delay scheme. To test robustness to system changes, we re-run the asynchronous methods on CIFAR-10 and 20NewsGroups under an alternative delay schedule that shifts the induced staleness distribution (see Fig. 8 in Appendix D.2) without re-tuning any hyperparameters. These runs use Dirichlet $\alpha = 1$. Fig. 4 shows that the DFKD-based methods (FedRevive and DD-FedRevive) remain substantially more robust to this delay shift, while asynchronous baselines (e.g., FedBuff and AFL-DW) show slower convergence and lower final accuracy.

Sensitivity to KD budget. Since distillation is the main server-side cost, we vary the number of KD steps K_{KD} on CIFAR-10. As shown in Fig. 3, FedRevive remains strong even with substantially

fewer KD steps, indicating that its gains are not tied to a heavy server-side KD budget. Appendix D.1 evaluates wall-clock performance with server overhead under several hardware configurations.

Update contributions in DFKD and β function. We analyze the contributions of stale client and KD-revived updates in Eq. (3), as well as the role of the staleness-dependent mixing schedule $\beta(\tau)$. For FedRevive and DD-FedRevive, $\beta(\tau)$ controls the interpolation between the parameter-space update and the KD update: when staleness is small, the server should rely more on the raw client update, whereas for highly stale updates, the KD component should be emphasized. Results show that the KD-revived component provides strong gains, while over-reliance throughout training degrades overall performance, indicating that sustained progress still requires preserving the contribution of the original client updates across rounds. Full experimental details and plots are provided in Appendix G. We consider several monotone scheduling families for β (Figure 15) and use the $1 - \cos$ine schedule for FedRevive, while DD-FedRevive uses a constant mixing coefficient $\beta = 0.5$. FedRevive is also broadly robust to the choice of $\beta(\tau)$, with adaptive schedules yielding better overall training behavior. See Appendix G.1 for a detailed comparison.

4 Conclusion

Asynchronous federated learning algorithms offer the scalability required for large-scale decentralized training and cross-device FL, yet suffer from staleness. Although stale updates can hinder convergence, they still contain valuable information about the clients' local data. We introduce FedRevive, a scalable asynchronous FL framework that extracts and utilizes this information through data-free knowledge distillation. By combining lightweight meta-learned synthetic data generation with multi-teacher distillation, our approach effectively mitigates staleness while preserving data privacy and system efficiency. Experiments on vision and text tasks demonstrate that FedRevive significantly improves convergence speed and final accuracy over baselines. Extending FedRevive with pretrained generative models for applicable data modalities presents a promising direction.

Acknowledgements

This work was partially supported by the US National Science Foundation under grants CNS-2106891 and CNS-2409138 to CJW and CCF 2045694, CNS-2112471, CPS-2111751, and ONR N00014-23-1-2149 to GJ.

References

[1] Durmus Alp Emre Acar, Yue Zhao, Ramon Matas, Matthew Mattina, Paul Whatmough, and Venkatesh Saligrama. Federated learning based on dynamic regularization. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=B7v4QMR6Z9w`.

[2] Baris Askin, Pranay Sharma, Carlee Joe-Wong, and Gauri Joshi. Fedast: federated asynchronous simultaneous training. In *Proceedings of the Fortieth Conference on Uncertainty in Artificial Intelligence*, UAI '24. JMLR.org, 2024.

[3] Kallista A. Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloé Kiddon, Jakub Konečný, Stefano Mazzocchi, H. Brendan McMahan, Timon Van Overveldt, David Petrou, Daniel Ramage, and Jason Roselander. Towards federated learning at scale: System design. *CoRR*, abs/1902.01046, 2019. URL `http://arxiv.org/abs/1902.01046`.

[4] Sebastian Caldas, Sai Meher Karthik Duddu, Peter Wu, Tian Li, Jakub Konečný, H. Brendan McMahan, Virginia Smith, and Ameet Talwalkar. Leaf: A benchmark for federated settings, 2019. URL `https://arxiv.org/abs/1812.01097`.

[5] Ming Chen, Bingcheng Mao, and Tianyi Ma. Fedsa: A staleness-aware asynchronous federated learning algorithm with non-iid data. *Future Generation Computer Systems*, 120:1–12, 2021.

[6] Yang Chen, Xiaoyan Sun, and Yaochu Jin. Communication-efficient federated deep learning with layerwise asynchronous model update and temporally weighted aggregation. *IEEE transactions on neural networks and learning systems*, 31(10):4229–4238, 2019.

[7] Yujing Chen, Yue Ning, Martin Slawski, and Huzefa Rangwala. Asynchronous Online Federated Learning for Edge Devices with Non-IID Data . In *2020 IEEE International Conference on Big Data (Big Data)*, pages 15–24, Los Alamitos, CA, USA, December 2020. IEEE Computer Society. doi: 10.1109/BigData50022.2020.9378161. URL https://doi.ieeecomputersociety.org/10.1109/BigData50022.2020.9378161.

[8] Henggang Cui, James Cipar, Qirong Ho, Jin Kyu Kim, Seunghak Lee, Abhimanu Kumar, Jinliang Wei, Wei Dai, Gregory R Ganger, Phillip B Gibbons, et al. Exploiting bounded staleness to speed up big data analytics. In *USENIX Annual Technical Conference (ATC)*, pages 37–48, 2014.

[9] Jeffrey Dean et al. Large scale distributed deep networks. In *Advances in Neural Information Processing Systems*, pages 1223–1231, 2012.

[10] Sanghamitra Dutta, Gauri Joshi, Soumyadip Ghosh, Parijat Dube, and Priya Nagpurkar. Slow and Stale Gradients Can Win the Race: Error-Runtime Trade-offs in Distributed SGD. *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*, April 2018.

[11] Boyu Fan, Siyang Jiang, Xiang Su, Sasu Tarkoma, and Pan Hui. A Survey on Model-heterogeneous Federated Learning: Problems, Methods, and Prospects . In *2024 IEEE International Conference on Big Data (BigData)*, pages 7725–7734, Los Alamitos, CA, USA, December 2024. IEEE Computer Society. doi: 10.1109/BigData62323.2024.10825769. URL https://doi.ieeecomputersociety.org/10.1109/BigData62323.2024.10825769.

[12] Gongfan Fang, Kanya Mo, Xinchao Wang, Jie Song, Shitao Bei, Haofei Zhang, and Mingli Song. Up to 100x faster data-free knowledge distillation. *Proceedings of the AAAI Conference on Artificial Intelligence*, 36(6):6597–6604, Jun. 2022. doi: 10.1609/aaai.v36i6.20613. URL https://ojs.aaai.org/index.php/AAAI/article/view/20613.

[13] Lingzhi Gao, Zhenyuan Zhang, and Chao Wu. Feddtg: Federated data-free knowledge distillation via three-player generative adversarial networks, 2025. URL https://arxiv.org/abs/2201.03169.

[14] Yuxian Gu, Xu Han, Zhiyuan Liu, and Minlie Huang. PPT: Pre-trained prompt tuning for few-shot learning. In Smaranda Muresan, Preslav Nakov, and Aline Villavicencio, editors, *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 8410–8423, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.acl-long.576. URL https://aclanthology.org/2022.acl-long.576/.

[15] Suyog Gupta, Wei Zhang, and Fei Wang. Model accuracy and runtime tradeoff in distributed deep learning: A systematic study. In *International Conference on Data Mining*, pages 171–180, 2016.

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[17] Geoffrey E. Hinton, Oriol Vinyals, and Jeffrey Dean. Distilling the knowledge in a neural network. *CoRR*, abs/1503.02531, 2015. URL http://arxiv.org/abs/1503.02531.

[18] Qirong Ho, James Cipar, Henggang Cui, Seunghak Lee, Jin Kyu Kim, Phillip B Gibbons, Garth A Gibson, Greg Ganger, and Eric P Xing. More effective distributed ml via a stale synchronous parallel parameter server. In *Advances in Neural Information Processing Systems*, pages 1223–1231, 2013.

[19] Sohei Itahara, Takayuki Nishio, Yusuke Koda, Masahiro Morikura, and Koji Yamamoto. Distillation-based semi-supervised federated learning for communication-efficient collaborative training with non-iid private data. *IEEE Transactions on Mobile Computing*, 22(1): 191–205, 2021.

[20] Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. TinyBERT: Distilling BERT for natural language understanding. In Trevor Cohn, Yulan He, and Yang Liu, editors, *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4163–4174, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.findings-emnlp.372. URL https://aclanthology.org/2020.findings-emnlp.372/.

[21] Anastasiia Koloskova, Sebastian U Stich, and Martin Jaggi. Sharper convergence guarantees for asynchronous sgd for distributed and federated learning. *Advances in Neural Information Processing Systems*, 35:17202–17215, 2022.

[22] Fan Lai, Yinwei Dai, Xiangfeng Zhu, Harsha V. Madhyastha, and Mosharaf Chowdhury. Fedscale: Benchmarking model and system performance of federated learning. In *Proceedings of the First Workshop on Systems Challenges in Reliable and Secure Federated Learning*, ResilientFL '21, page 1–3, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450387088. doi: 10.1145/3477114.3488760. URL https://doi.org/10.1145/3477114.3488760.

[23] Fan Lai, Xiangfeng Zhu, Harsha V. Madhyastha, and Mosharaf Chowdhury. Oort: Efficient federated learning via guided participant selection. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2021.

[24] Sangho Lee, Kiyoon Yoo, and Nojun Kwak. Edge bias in federated learning and its solution by buffered knowledge distillation. *arXiv preprint arXiv:2010.10338*, 2020.

[25] Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning. In Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih, editors, *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 3045–3059, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.243. URL https://aclanthology.org/2021.emnlp-main.243/.

[26] Daliang Li and Junpu Wang. Fedmd: Heterogenous federated learning via model distillation, 2019. URL https://arxiv.org/abs/1910.03581.

[27] Lin Li, Jianping Gou, Baosheng Yu, Lan Du, Zhang Yi, and Dacheng Tao. Federated distillation: A survey. *arXiv preprint arXiv:2404.08564*, 2024.

[28] Yichen Li, Hang Su, Huifa Li, Haolin Yang, Xinlin Zhuang, Haochen Xue, Haozhao Wang, and Imran Razzak. Fedcd: towards consolidated distillation for heterogeneous federated learning. AAAI'26/IAAI'26/EAAI'26. AAAI Press, 2026. ISBN 978-1-57735-906-7. doi: 10.1609/aaai.v40i28.39494. URL https://doi.org/10.1609/aaai.v40i28.39494.

[29] Xiangru Lian, Yijun Huang, Yuncheng Li, and Ji Liu. Asynchronous parallel stochastic gradient for nonconvex optimization. In *Advances in Neural Information Processing Systems*, pages 2737–2745, 2015.

[30] Tao Lin, Lingjing Kong, Sebastian U Stich, and Martin Jaggi. Ensemble distillation for robust model fusion in federated learning. *Advances in neural information processing systems*, 33: 2351–2363, 2020.

[31] Raphael Gontijo Lopes, Stefano Fenu, and Thad Starner. Data-free knowledge distillation for deep neural networks, 2017. URL https://arxiv.org/abs/1710.07535.

[32] Chaoyi Lu, Yiding Sun, Pengbo Li, and Zhichuan Yang. Corrected with the latest version: Make robust asynchronous federated learning possible, 2025. URL https://arxiv.org/abs/2504.04081.

[33] Xiaofeng Lu, Yuying Liao, Pietro Lio, and Pan Hui. Privacy-preserving asynchronous federated learning mechanism for edge network computing. *IEEE Access*, 8:48970–48981, 2020. doi: 10.1109/ACCESS.2020.2978082.

[34] Bing Luo, Wenli Xiao, Shiqiang Wang, Jianwei Huang, and Leandros Tassiulas. Tackling system and statistical heterogeneity for federated learning with adaptive client sampling. In *IEEE INFOCOM 2022 - IEEE Conference on Computer Communications*, page 1739–1748. IEEE Press, 2022. doi: 10.1109/INFOCOM48880.2022.9796935. URL https://doi.org/10.1109/INFOCOM48880.2022.9796935.

[35] Amir M. Mansourian, Rozhan Ahmadi, Masoud Ghafouri, Amir Mohammad Babaei, Elaheh Badali Golezani, Zeynab yasamani ghamchi, Vida Ramezanian, Alireza Taherian, Kimia Dinashi, Amirali Miri, and Shohreh Kasaei. A comprehensive survey on knowledge distillation. *Transactions on Machine Learning Research*, 2025. ISSN 2835-8856. URL https://openreview.net/forum?id=3cbJzdR78B.

[36] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-Efficient Learning of Deep Networks from Decentralized Data. In Aarti Singh and Jerry Zhu, editors, *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, volume 54 of *Proceedings of Machine Learning Research*, pages 1273–1282. PMLR, 20–22 Apr 2017. URL https://proceedings.mlr.press/v54/mcmahan17a.html.

[37] Ioannis Mitliagkas, Ce Zhang, Stefan Hadjis, and Christopher Ré. Asynchrony begets momentum, with an application to deep learning. In *Allerton Conference on Communication, Control, and Computing*, pages 997–1004. IEEE, 2016.

[38] John Nguyen, Kshitiz Malik, Hongyuan Zhan, Ashkan Yousefpour, Mike Rabbat, Mani Malek, and Dzmitry Huba. Federated learning with buffered asynchronous aggregation. In Gustau Camps-Valls, Francisco J. R. Ruiz, and Isabel Valera, editors, *Proceedings of The 25th International Conference on Artificial Intelligence and Statistics*, volume 151 of *Proceedings of Machine Learning Research*, pages 3581–3607. PMLR, 28–30 Mar 2022. URL https://proceedings.mlr.press/v151/nguyen22b.html.

[39] Alex Nichol, Joshua Achiam, and John Schulman. On first-order meta-learning algorithms. *arXiv preprint arXiv:1803.02999*, 2018.

[40] Flavio Di Palo, Prateek Singhi, and Bilal H Fadlallah. Performance-guided LLM knowledge distillation for efficient text classification at scale. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen, editors, *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 3675–3687, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-main.215. URL https://aclanthology.org/2024.emnlp-main.215/.

[41] Laiqiao Qin, Tianqing Zhu, Wanlei Zhou, and Philip S. Yu. Knowledge distillation in federated learning: a survey on long lasting challenges and new solutions, 2024. URL https://arxiv.org/abs/2406.10861.

[42] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer, 2023. URL https://arxiv.org/abs/1910.10683.

[43] Peter Richtarik. Handling device heterogeneity in federated learning: The first optimal parallel sgd in the presence of data, compute and communication heterogeneity. In *Proceedings of the International Workshop on Secure and Efficient Federated Learning*, pages 1–1, 2025.

[44] Nicola Rieke, Jonny Hancox, Wenqi Li, Fausto Milletari, Holger R Roth, Shadi Albarqouni, Spyridon Bakas, Mathieu N Galtier, Bennett A Landman, Klaus Maier-Hein, et al. The future of digital health with federated learning. *NPJ digital medicine*, 3(1):119, 2020.

[45] Ertong Shang, Hui Liu, Zhuo Yang, Junzhao Du, and Yiming Ge. Fedbikd: Federated bidirectional knowledge distillation for distracted driving detection. *IEEE Internet of Things Journal*, 10(13):11643–11654, 2023. doi: 10.1109/JIOT.2023.3243622.

[46] Guomei Shi, Li Li, Jun Wang, Wenyan Chen, Kejiang Ye, and ChengZhong Xu. Hysync: Hybrid federated learning with effective synchronization. In *2020 IEEE 22nd International Conference on High Performance Computing and Communications; IEEE 18th International Conference on Smart City; IEEE 6th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, pages 628–633, 2020. doi: 10.1109/HPCC-SmartCity-DSS50907.2020.00080.

[47] Yijun Tian, Yikun Han, Xiusi Chen, Wei Wang, and Nitesh V. Chawla. Beyond answers: Transferring reasoning capabilities to smaller llms using multi-teacher knowledge distillation. In *Proceedings of the Eighteenth ACM International Conference on Web Search and Data Mining*, WSDM '25, page 251–260, New York, NY, USA, 2025. Association for Computing Machinery. ISBN 9798400713293. doi: 10.1145/3701551.3703577. URL https://doi.org/10.1145/3701551.3703577.

[48] Haozhao Wang, Yichen Li, Wenchao Xu, Ruixuan Li, Yufeng Zhan, and Zhigang Zeng. Dafkd: Domain-aware federated knowledge distillation. In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 20412–20421, 2023. doi: 10.1109/CVPR52729.2023.01955.

[49] Yujia Wang, Shiqiang Wang, Songtao Lu, and Jinghui Chen. Fadas: Towards federated adaptive asynchronous optimization. In *International Conference on Machine Learning*, pages 51701–51733. PMLR, 2024.

[50] Cong Xie, Sanmi Koyejo, and Indranil Gupta. Asynchronous federated optimization. *arXiv preprint arXiv:1903.03934*, 2019.

[51] Chenhao Xu, Youyang Qu, Yong Xiang, and Longxiang Gao. Asynchronous federated learning on heterogeneous devices: A survey. *Computer Science Review*, 50:100595, 2023. ISSN 1574-0137. doi: https://doi.org/10.1016/j.cosrev.2023.100595. URL https://www.sciencedirect.com/science/article/pii/S157401372300062X.

[52] Haotian Xu, Zhaorui Zhang, Sheng Di, Benben Liu, Khalid Ayed Alharthi, and Jiannong Cao. Fedfa: a fully asynchronous training paradigm for federated learning. *arXiv preprint arXiv:2404.11015*, 2024.

[53] Xiaohan Xu, Ming Li, Chongyang Tao, Tao Shen, Reynold Cheng, Jinyang Li, Can Xu, Dacheng Tao, and Tianyi Zhou. A survey on knowledge distillation of large language models, 2024. URL https://arxiv.org/abs/2402.13116.

[54] Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. Federated machine learning: Concept and applications. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 10(2):1–19, 2019.

[55] Xiaofan Yu, Lucy Cherkasova, Harsh Vardhan, Quanling Zhao, Emily Ekaireb, Xiyuan Zhang, Arya Mazumdar, and Tajana Rosing. Async-hfl: Efficient and robust asynchronous federated learning in hierarchical iot networks. In *Proceedings of the 8th ACM/IEEE Conference on Internet of Things Design and Implementation*, pages 236–248, 2023.

[56] Sergey Zagoruyko and Nikos Komodakis. Paying more attention to attention: Improving the performance of convolutional neural networks via attention transfer. In *International Conference on Learning Representations*, 2017.

[57] Yu Zang, Zhe Xue, Shilong Ou, Lingyang Chu, Junping Du, and Yunfei Long. Efficient asynchronous federated learning with prospective momentum aggregation and fine-grained correction. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(15):16642–16650, Mar. 2024. doi: 10.1609/aaai.v38i15.29603. URL https://ojs.aaai.org/index.php/AAAI/article/view/29603.

[58] Lin Zhang, Li Shen, Liang Ding, Dacheng Tao, and Ling-Yu Duan. Fine-tuning global model via data-free knowledge distillation for non-iid federated learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10174–10183, 2022.

[59] Peiyuan Zhang, Guangtao Zeng, Tianduo Wang, and Wei Lu. Tinyllama: An open-source small language model, 2024.

[60] Wei Zhang, Suyog Gupta, Xiangru Lian, and Ji Liu. Staleness-aware async-sgd for distributed deep learning. In *International Joint Conference on Artificial Intelligence*, pages 2350–2356. AAAI Press, 2016.

[61] Ligeng Zhu, Hongzhou Lin, Yao Lu, Yujun Lin, and Song Han. Delayed gradient averaging: Tolerate the communication latency for federated learning. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021. URL `https://openreview.net/forum?id=ACFHNxVNvfk`.

[62] Zhuangdi Zhu, Junyuan Hong, and Jiayu Zhou. Data-free knowledge distillation for heterogeneous federated learning. In *International conference on machine learning*, pages 12878–12889. PMLR, 2021.

A Extended Related Work

Synchronous FL. Since the seminal work of McMahan et al. [36], FL has proven effective for collaboratively training models from decentralized and private data sources. Most existing FL algorithms employ *synchronous* aggregation, where the server waits for all selected clients to complete their local training and then averages their model updates. However, this synchronization scheme creates a major bottleneck. In realistic environments, edge devices exhibit high variability in computation and communication capabilities (e.g., mobile phones versus embedded edge devices, Wi-Fi versus cellular connections), resulting in widely varying local training times. Consequently, in each round, the server must wait for the slowest client (the "straggler"), leaving faster clients idle and underutilized. Several methods have been proposed to alleviate the straggler problem, such as aggregating only the updates that arrive before a predetermined time [3] or randomly subsampling available clients [34]. However, these approaches can bias training toward inherently faster clients and still underutilize slower ones. The straggler issue is further exacerbated in large-scale deployments, where the growing number of clients increases the likelihood of slow update returns and communication delays [55]. As a result, synchronous aggregation inherently limits scalability and resource efficiency in federated networks.

Asynchronous FL. To overcome the straggler bottleneck and scalability issues of synchronous aggregation, a growing body of work has explored *asynchronous FL* (AFL) [50, 51, 38, 7, 43, 39]. AFL builds upon a line of previous works on asynchronous distributed stochastic gradient descent (SGD) in the data-center setting [9, 15, 8, 18, 29, 60, 37, 10]. In AFL, the server integrates client updates as soon as they arrive, either individually or in small mini-batches, without waiting for global synchronization. This design enables faster convergence by mitigating the straggler effect and improving resource utilization. However, asynchronicity introduces a new challenge: *stale updates*. Because clients train on outdated versions of the global model, their updates may become obsolete by the time they reach the server. Incorporating these stale updates can inject inconsistency or even destabilize training [51].

To mitigate staleness, several strategies have been proposed. Nguyen et al. [38] employ a buffer for incoming updates and averages them before aggregation, which smooths model update frequency and reduces variance within the server updates due to data heterogeneity despite the cost of slower server updates. Another line of work assigns lower weights to stale updates as a function of their staleness [6, 46, 33, 60, 10, 8], reducing the influence of outdated gradients but potentially discarding valuable information from slower clients. Importantly, each client's update encapsulates rich information about its local data distribution. In AFL, the challenge lies in how to effectively utilize this information despite staleness. In our work, we leverage DFKD to extract and transfer the informative content from stale client updates to the current server model.

KD and DFKD. KD is a powerful technique for transferring learned representations of a better-performing *teacher* model to a *student* [17]. The standard formulation trains the student to mimic the teacher's output logits, which encode class probabilities, inter-class similarity, and uncertainty. KD has since been widely adopted across various modalities, including vision, speech, and natural language

processing, where it consistently improves generalization and model compactness [35, 56, 20]. The vanilla KD framework, however, fundamentally requires access to a dataset on which both the teacher and the student are evaluated during distillation. In many practical cases, such as privacy-preserving learning or FL, direct access to data is unavailable, making conventional KD infeasible. This has motivated the emergence of DFKD [31], which aims to perform distillation without real data. Methods typically optimize a generator or latent codes to synthesize pseudo-samples under objectives that align teacher predictions, match intermediate statistics (e.g., batch statistics), or introduce adversarial criteria to amplify teacher–student discrepancies [35, 31, 12]. Then, these samples are utilized for distillation.

KD and DFKD in FL. KD has also been incorporated into the FL paradigm for various purposes, such as mitigating client bias and data heterogeneity, enabling architecture-heterogeneous models, and enhancing data privacy [27, 26, 30, 41, 11]. In FL, KD can be performed on either the *server side*, where the global model distills knowledge from client models [24, 26], or on the *client side*, where clients distill knowledge from the server model or peer models [19, 45]. While some of these methods assume the presence of a shared public dataset for knowledge transfer [19], some approaches adopt DFKD to remove this dependency [13, 58, 62]. However, existing works applying KD or DFKD have been primarily designed for *synchronous* FL, and the use of KD in *asynchronous* FL remains largely unexplored. The few existing efforts [32] rely on the unrealistic assumption of public data availability. In this work, we leverage DFKD in the asynchronous FL setting to enhance both scalability and convergence stability.

B Deriving Client Class-Proportion Proxies

FedRevive uses proxy per-client class-proportion vectors to (i) construct client-conditioned synthesis targets and (ii) perform class-aware multi-teacher distillation. To avoid requiring access to private client label histograms, we derive a coarse per-client class-proportion proxy on the server using only the first few *regular* model updates (no extra client computation, metadata, or communication).

Deriving class-proportion proxies from early uploads. Let $\mathbf{y}_i^{(u)}$ denote the u-th model upload received from client i (ordered by server reception). We find proxy class proportions from the first $T_{\text{est}} = 2$ uploads. For each upload $\mathbf{y}_i^{(u)}$, the server probes the model with a batch of random inputs $\mathbf{r}$ and computes

$$\widetilde{\boldsymbol{\pi}}_i^{(u)} = (1/|\mathbf{r}|) \sum_{r \in \mathbf{r}} \sigma\Big(\mathbf{y}_i^{(u)}(r)/\mathrm{T}_{\text{probe}}\Big), \qquad u = 1, \ldots, T_{\text{est}},$$

where $\sigma(\cdot)$ is the $\mathrm{softmax}$ function. For vision models, r is i.i.d. Gaussian noise with the same input shape as the model. For language models, r is a random token sequence with the same length as the model input (tokens sampled uniformly from the vocabulary). The final proxy is the average across early uploads,

$$\hat{\boldsymbol{\pi}}_i = (1/T_{\text{est}}) \sum_{u=1}^{T_{\text{est}}} \widetilde{\boldsymbol{\pi}}_i^{(u)},$$

and is fixed for the remainder of training. We use random-input probing as a weak-signal heuristic: local training generally induces class-dependent logit biases. As a result, even under out-of-distribution probes (Gaussian noise for vision, random tokens for text), the model's predictive distribution is not arbitrary; it is typically closer to uniform when the client's label histogram is near-uniform, and becomes increasingly skewed toward the majority classes as the client's data become more imbalanced. We do not claim this yields exact histograms, but only a coarse ranking of dominant classes, which we find sufficient to weight buffered teachers.

This estimation is performed online as uploads arrive and requires only server-side forward passes. We apply temperature scaling in the probing softmax. We use $\mathrm{T}_{\text{probe}} = 0.8$ on CIFAR-10, $\mathrm{T}_{\text{probe}} = 1$ on CIFAR-100, $\mathrm{T}_{\text{probe}} = 3$ on FEMNIST, and $\mathrm{T}_{\text{probe}} = 1.6$ on 20NewsGroups.

Proxy quality. For analysis only, we compare the proxy $\hat{\boldsymbol{\pi}}_i$ against the ground-truth client label distribution $\boldsymbol{\pi}_i$ via $D_{\mathrm{KL}}(\boldsymbol{\pi}_i \| \hat{\boldsymbol{\pi}}_i)$. For each seed, we compute the mean KL across all clients ($N = 1000$), then report mean $\pm$ std across seeds in Table 2.

Table 2: KL divergence between true client label proportions and the server-derived proxy proportions, reported as mean ± std across seeds. We report the average client KL after the second received update.

Dataset	**Average** $D_{\mathrm{KL}}(\boldsymbol{\pi}_i \| \hat{\boldsymbol{\pi}}_i)$
CIFAR-10	0.4240 ± 0.0103
CIFAR-100	0.5539 ± 0.0174
20NewsGroups	0.5889 ± 0.0312
FEMNIST	0.4561 ± 0.0554

Sensitivity to label-proportion estimation errors. While the probing-based proxy is only a coarse heuristic, it is designed to recover *relative* class dominance rather than exact client histograms. To evaluate whether estimation errors affect FedRevive, we run an ablation where the class-aware teacher weighting uses either the estimated proportions or the ground-truth client label proportions (just for an ablation purpose). As shown in Fig. 5, using estimated proportions matches the training dynamics and final accuracy across CIFAR-10, CIFAR-100, and FEMNIST, indicating that FedRevive is not overly sensitive to estimation noise and mainly benefits from a rough identification of dominant classes.

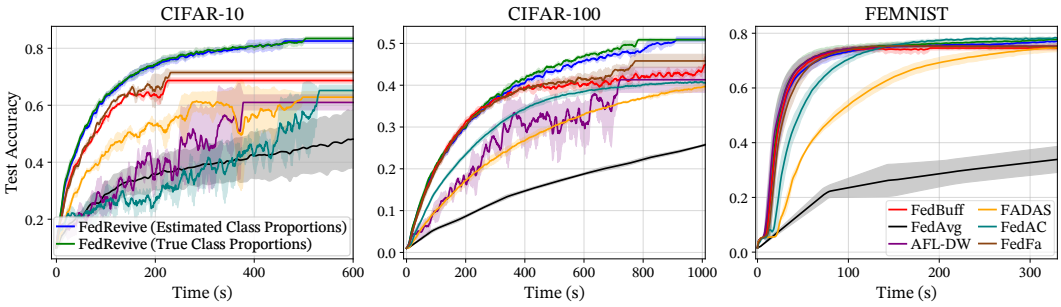


Figure 5: **FedRevive with estimated client label proportions on CIFAR-10, CIFAR-100, and FEMNIST.** Replacing ground-truth client label proportions with our probing-based proxies yields similar training dynamics and final performance across datasets.

Dominant-class recovery. Because the proxy is used only to provide a coarse dominant-class cue, exact histogram matching is not the most relevant metric. We therefore additionally measure whether the proxy correctly identifies the client's dominant class (top-1) and whether the true dominant class is contained in the proxy's top-2 predicted classes. Table 3 reports these accuracies across clients. These metrics better reflect the role of the proxy in class-aware teacher weighting than KL alone.

Table 3: Top-1 and top-2 dominant-class recovery of the server-derived proxy proportions, reported as mean ± std across seeds. Top-1 accuracy checks whether $\arg\max_j \hat{\pi}_{i,j}$ matches the true dominant class of client i; top-2 accuracy checks whether the true dominant class appears in the proxy's top-2 classes.

Dataset	**Nb. of classes**	**Top-1 acc.**	**Top-2 acc.**
CIFAR-10	10	0.456 ± 0.022	0.673 ± 0.032
CIFAR-100	100	0.135 ± 0.001	0.219 ± 0.010
20NewsGroups	20	0.198 ± 0.015	0.276 ± 0.062
FEMNIST	62	0.175 ± 0.045	0.338 ± 0.014

Effect of label skew on dominant-class recovery. To isolate the effect of label skew, we evaluate the proxy on 1000 clients under Dirichlet partitions with varying concentration α. For each setting, clients perform one local training round from a common randomly initialized model, and we estimate each proxy from the resulting upload without server training. We use the same top-1 and top-2

Table 4: Dominant-class recovery across Dirichlet concentration levels after one local training round from random initialization. Entries are recovery rates across 1000 clients; smaller α indicates stronger label skew.

Dataset	**Metric**	Dirichlet concentration α					
		0.05	0.10	0.30	0.50	1.00	5.00
CIFAR-10	Top-1	0.930	0.899	0.822	0.741	0.691	0.437
	Top-2	0.995	0.989	0.960	0.915	0.872	0.611
CIFAR-100	Top-1	0.708	0.671	0.528	0.472	0.382	0.289
	Top-2	0.876	0.847	0.679	0.626	0.551	0.468

recovery metrics as above. This controlled study differs from Table 3, which evaluates proxies from early received uploads during full asynchronous training. The numerical results are therefore not directly comparable.

As shown in Table 4, both recovery metrics improve as label skew increases, consistent with stronger class-dependent logit biases after local training. Near-uniform client distributions provide weaker dominant-class cues. These results indicate that stronger label heterogeneity improves, rather than degrades, the proxy's ability to identify dominant classes in the evaluated settings.

C FLOPs Analysis and Server Overhead Discussion

We quantify the additional computation introduced by FedRevive on the server by reporting FLOPs per processed sample for the dominant components of the pipeline: (i) client local training, (ii) server generator training and pseudo-sample synthesis, and (iii) server-side multi-teacher distillation. Table 5 compares these costs under our default hyperparameters.

Table 5: **Compute overhead breakdown (FLOPs) per sample for CIFAR-10 experiments.** FLOPs (G) for dominant components. c denotes the teacher buffer size.
†Rows report total compute except the last row, which reports effective blocking compute with concurrent implementation (total FLOPs unchanged).

Component	FLOPs (G) per iter†	Total across iters
Client local training (25 iters)	1.65	$\mathbf{41.25}$
Generator train&synth. (every T_{gen} rounds, amortized)	$0.14 + 0.11\,c$	$\mathbf{0.28 + 0.22\,c}$
Distillation (sequential implementation, 10 iters)	$1.65 + 0.55\,c$	$16.5 + 5.5\,c$
Distillation (effective cost w/ concurrent implement.)	2.2	$\mathbf{22}$

First, generator training and synthesis is performed for only $K_{\text{synth}} = 2$ iterations and only once every $T_{\text{gen}} = 10$ server updates, so its average cost is amortized by a factor of $T_{\text{gen}} = 10$ (second row block in Table 5). Second, distillation is run for K_{KD}=10 iterations, and its cost decomposes into student updates plus teacher forward passes. Since the server maintains a buffer of teachers, forward passes for all buffered teachers except the most recent arrival can be executed asynchronously while waiting for the next client upload; thus, the effective blocking compute can be close to the student-update cost (last row in Table 5).

Third, server computation can be pipelined with client progress in asynchronous implementations. In a fully pipelined implementation, upon receiving a client update, the server may immediately dispatch the next local-training request using the current global state, and then perform synthesis, distillation, and hybrid aggregation in parallel before subsequent dispatches. This trades blocking server time for a small amount of additional staleness, which is already intrinsic to AFL. Teacher forward passes for buffered models and periodic synthesis can also be executed while the server is waiting for subsequent client uploads. We therefore model server-side KD as overlapped/non-blocking in the main simulator, but do not rely on this assumption being exact in every deployment. To account for settings where server computation is partially or fully blocking, Appendix D.1 reports a wall-clock evaluation with measured server overhead under several hardware configurations.

Fourth, moderate server-side overhead is typically less critical than client-side computation and straggling, since modern servers are substantially more compute-capable than edge devices [3]. Moreover, in many practical FL deployments, especially wireless settings, communication latency can dominate end-to-end runtime [36, 61], so additional server compute often has limited impact on overall wall-clock training time.

Finally, we did not tune $K_{\text{synth}}, K_{\text{KD}}, T_{\text{gen}}$ to minimize FLOPs, since our focus is better convergence performance. In deployments where the server budget is tighter, K_{synth} and K_{KD} can be reduced and T_{gen} increased to lower the overhead.

D Details on Time Simulation in Experiments

Delay simulation model. Our simulator assigns each client i three independent delay generators: local computation delay d_i^{train} (local training), downlink delay d_i^{down} (model download), and uplink delay d_i^{up} (update upload). A client's realized end-to-end round-trip time is the sum of these components, and the staleness τ is induced by how many server updates occur while an update is in transit and being computed. Since the servers are generally equipped with much faster hardware compared to client computation and communication delays, we model server-side operation time as non-blocking or overlapped in our simulation, supported by measured runtimes discussed in detail next. To model persistent heterogeneity, we first randomly sample a *client-specific mean* for each delay component, and keep it fixed for that client throughout training. The instantaneous delay values then vary around this mean according to the chosen distribution. The three components are sampled independently per client, so a client may be slow in computation but fast in communication, or vice versa.

Server-side runtime. We benchmark the wall-clock cost of the main computation blocks in our CIFAR-10 setup with batch size 32 under three representative configurations. On a 16-core CPU, a single client local-training run takes 4.874 s in total. Even when clients are equipped with a Tesla T4, local training still takes 0.657 s per run. In contrast, when the server is equipped with a modern accelerator (H100), the amortized synthesis runtime is 0.038 s per server update (with one generator update every 10 rounds), while distillation takes 0.110 s with a sequential implementation and 0.0569 s with an efficient non-blocking implementation. Furthermore, communication is typically the dominant bottleneck in cross-device FL [22, 61]. Therefore, we model comparatively small server-side operation time as non-blocking or overlapped in our simulation. Moreover, the server-side pipeline can be arranged to avoid introducing any additional wall-clock delay. The server can dispatch the next local-training request immediately upon receiving an update and run synthesis, distillation, and hybrid aggregation asynchronously afterward, which hides most server compute behind client-side latency at the cost of only a small additional staleness.

Mixture over client speeds. For each delay component, the client-specific mean is drawn from a discrete mixture distribution specified by a set of pairs $\{(p_k, \mu_k)\}_k$, where $\sum_k p_k = 1$. Concretely, with probability p_k the client is assigned mean delay μ_k for that component. This mixture produces multiple "speed tiers" and yields a long-tailed staleness distribution when combined with asynchronous execution.

Delay distributions and parameters. In our default configuration, the three delay components use different distributions: (i) *Local training delay* is exponential with a client-specific mean drawn from $\{(0.25, 1.0), (0.5, 1.3), (0.25, 1.6)\}$. (ii) *Download delay* is a constant value 0.1 for all clients. (iii) *Upload delay* is uniform around a client-specific mean drawn from $\{(0.5, 0.15), (0.5, 0.25)\}$, using a half-width of 0.02 (i.e., sampled from $[\max(0, \mu - 0.02),\ \mu + 0.02]$). All delay quantities are in the simulator's time units, and the above settings are used for all main experiments unless stated otherwise.

We plot the distribution of update staleness in server rounds from a sample experiment for a fully asynchronous algorithm in Figure 6. It is shown in the figure that the staleness distribution is long-tailed due to slow clients. The figure further validates the simulation's alignment with real practice, matching the staleness distribution observed in real-world experiments with millions of clients [38].

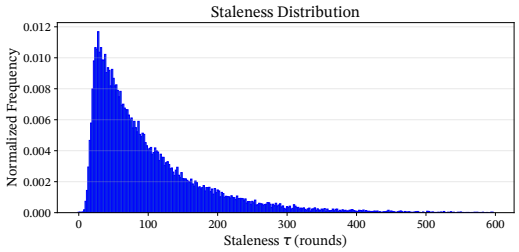


Figure 6: Distribution of update staleness across server rounds for a fully asynchronous setup. The distribution aligns with real-world AFL systems reported by Nguyen et al. [38].

D.1 Wall-Clock Evaluation with Server Overhead under Different Hardware Configurations

Experiments in Section 3 treat server-side KD work as overlapped or non-blocking. To evaluate the effect of explicit server blocking time, we adjust the CIFAR-10 wall-clock timelines post hoc with measured per-aggregation server runtimes under several server configurations. For an evaluation record at server round N and simulated timestamp T, we compute the server-overhead-adjusted timestamp as

$$T' = T + N d_{\text{sim}},$$

where d_{sim} is the measured per-aggregation server runtime converted into simulator time units. For this conversion, we assume edge-device clients that are slower than the servers. We take the communication-to-computation time ratio of mobile clients from [23]. Only FedRevive and DD-FedRevive are adjusted in this evaluation; the baselines are left unchanged because they do not perform server-side KD. This preserves event order, staleness, and all training dynamics, and only asks what wall-clock curve the same run would report if each aggregation incurred an additional blocking server delay. Results can be seen in Fig. 7.

The blocking server time depends on the server accelerator, implementation parallelism, client hardware, and communication regime. Therefore, we do not insert one fixed server delay into the main simulator, which would tie the wall-clock results to one deployment choice. Instead, the main simulator models the client-side delays that cause asynchrony and staleness. We account for server-side cost separately through measured runtimes, FLOPs, and Fig. 7.

D.2 Modified Delay Schedule and Shifted Staleness Distribution

In the experiment for robustness to change in delay scheme in Section 3.2, to evaluate robustness to system changes, we modify the parameters of the computation and communication delay distributions in the simulator, which yields a different induced staleness distribution (Fig. 8) than the default setting used throughout the main experiments (Fig. 6). In this robustness evaluation, we keep exactly the same hyperparameters for every method as tuned under the default delay scheme, and only change the delay sampling mechanism.

Modified delay parameters. Under the modified delay schedule, we use: (i) an exponential local training delay whose client-specific mean is set to 0.1, (ii) a constant download delay with mean 0.02 for all clients, (iii) a uniform upload delay with client-specific mean sampled from $\{(0.5, 0.05), (0.5, 0.10)\}$ and half-width 0.01 (i.e., sampled from $[\max(0, \mu - 0.01),\ \mu + 0.01]$). All other aspects of the simulator remain unchanged.

E Results with a Milder Heterogeneity Level

We provide additional experimental results under different levels of data heterogeneity. For the CIFAR-10, CIFAR-100, and 20NewsGroups datasets, we show the test accuracy curves when client data is distributed with milder non-IID partition ($\alpha = 1$). See Figure 9 for the results. Note that the FEMNIST dataset is naturally partitioned by real users and thus does not allow controlling its heterogeneity. Hence, it is excluded from this analysis. All hyperparameters except data heterogeneity are kept the same as those of main experiments.

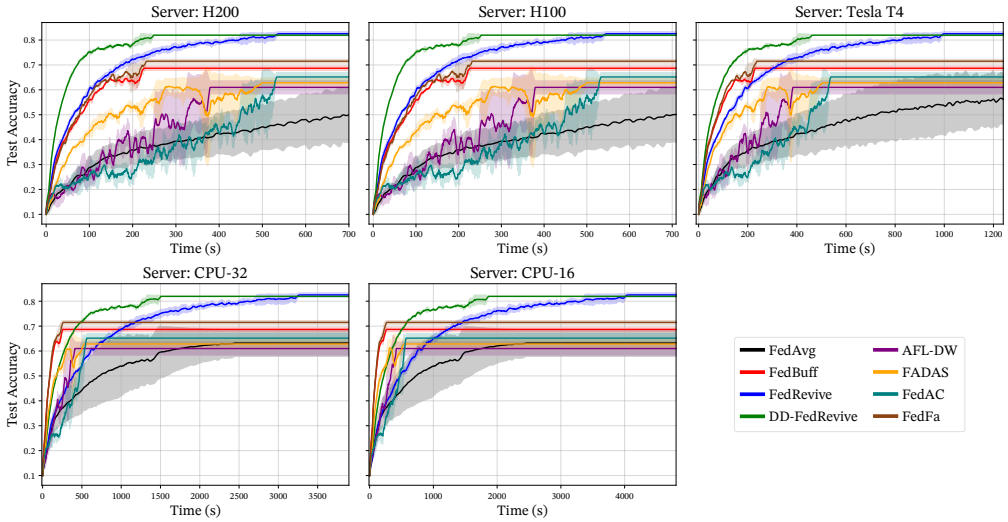


Figure 7: Wall-clock evaluation with server overhead on CIFAR-10 under different hardware assumptions. We adjust the CIFAR-10 timelines by adding measured and converted per-aggregation server runtimes for several server configurations (H200, H100, Tesla T4, CPU-32, and CPU-16), while leaving the underlying update sequence and training dynamics unchanged. With strong accelerators, the conclusions are essentially unchanged; with weaker servers, FedRevive and DD-FedRevive slow down in wall-clock time but still retain a clear accuracy advantage over the asynchronous baselines.

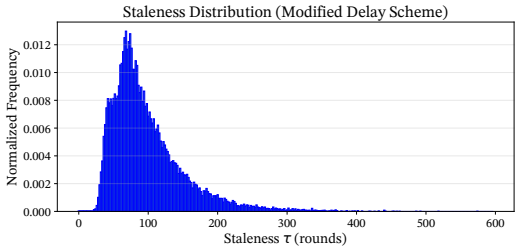


Figure 8: Staleness distribution induced by the modified delay schedule used in the experiment for robustness to change in delay scheme in Section 3.2. For reference, the default distribution is shown in Fig. 6.

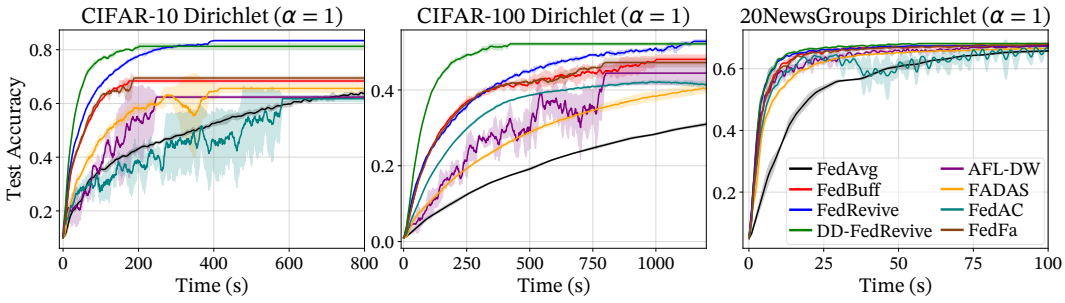
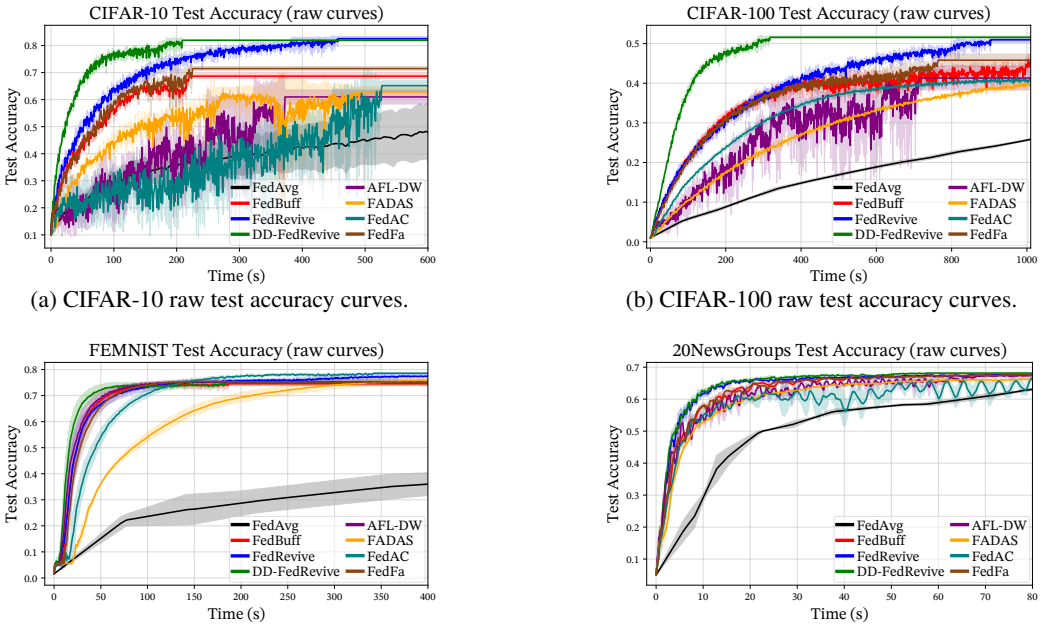


Figure 9: Test accuracy over simulated time on CIFAR-10, CIFAR-100, and 20NewsGroups under milder heterogeneity ($\alpha = 1$). FedRevive converges faster than FedBuff and achieves higher final accuracy across datasets.

F Raw Training Curves without Running Average Processing

As described in the main text, asynchronous algorithms often display fluctuations in performance due to delayed updates and highly heterogeneous model aggregation. To provide a clearer comparison, the main figures report the moving-averaged test accuracy over time. Here, we present the original unprocessed curves showing the raw test accuracy trajectories. Figure 10 illustrates these curves.



(a) CIFAR-10 raw test accuracy curves.

(b) CIFAR-100 raw test accuracy curves.

(c) FEMNIST raw accuracy curves.

(d) 20NewsGroups raw accuracy curves.

Figure 10: Raw accuracy curves across datasets ($\alpha = 0.5$).

G Update Ablations for DFKD and the Role of $\beta(\tau)$

This section provides a more detailed analysis of the update components in Eq. (3) and of the role of the staleness-dependent mixing schedule $\beta(\tau)$. We study these effects on CIFAR-10 by separating the contribution of the stale client update and the KD-revived update produced by KD-Revive$(\cdot)$.

At each server round, we consider three update variants: (i) only the stale client update $\boldsymbol{\Delta}_i^{(t-\tau_i^t)}$, (ii) only the KD-revived update $\boldsymbol{\Delta}_i^{\mathrm{KD}}$, and (iii) their combined update as in Eq. (3). We evaluate these variants under three mixing regimes: (a) the adaptive staleness-dependent schedule used in the main experiments, (b) a fixed schedule with $\beta(\tau) = 0.75$, and (c) a fixed schedule with $\beta(\tau) = 1$.

Fig. 11 summarizes the main findings. The left panel shows that the KD-revived update yields large per-round improvements, indicating that the DFKD component is a major source of the gains of FedRevive. At the same time, these per-update gains do not imply that training should rely on KD alone. The right panel shows that fixed high values of β lead to worse end-to-end training trajectories than the adaptive schedule. This indicates that sustained progress still requires preserving the contribution of the original client updates across rounds, while using KD more aggressively only when staleness is high.

For completeness, we also provide the detailed breakdown for each mixing regime separately in Fig. 12. Under the adaptive schedule, the combined update gives the strongest overall behavior, showing that the stale client update and the KD-revived update play complementary roles. When β is fixed to 0.75 or 1, the training dynamics deteriorate because the original client update is persistently underweighted or entirely ignored. These results support the use of a staleness-aware schedule that balances parameter-space aggregation and KD-based revival rather than relying on a constant mixing rule.

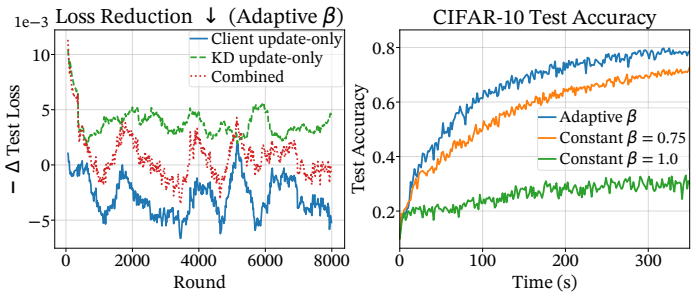
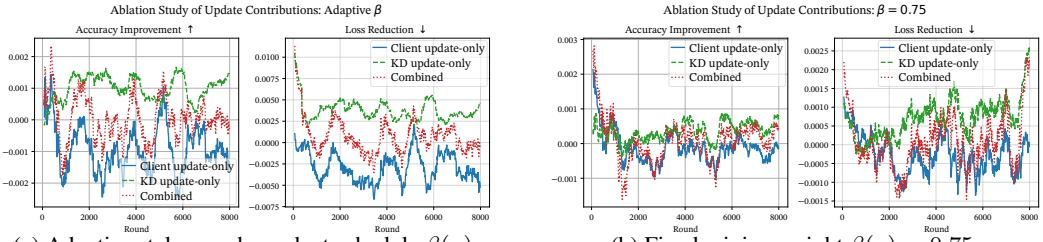
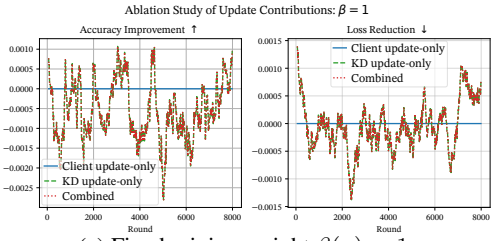


Figure 11: **Update contributions and the role of $\beta(\tau)$ on CIFAR-10. Left:** Per-round effect of applying only the stale client update, only the KD-revived update, or their combination under the adaptive $\beta(\tau)$ schedule. **Right:** Test accuracy vs. simulated time under adaptive and constant (0.75 and 1) $\beta(\tau)$. Constant schedules underweight the client update and degrade performance.



(a) Adaptive staleness-dependent schedule $\beta(\tau)$.

(b) Fixed mixing weight $\beta(\tau) = 0.75$.



(c) Fixed mixing weight $\beta(\tau) = 1$.

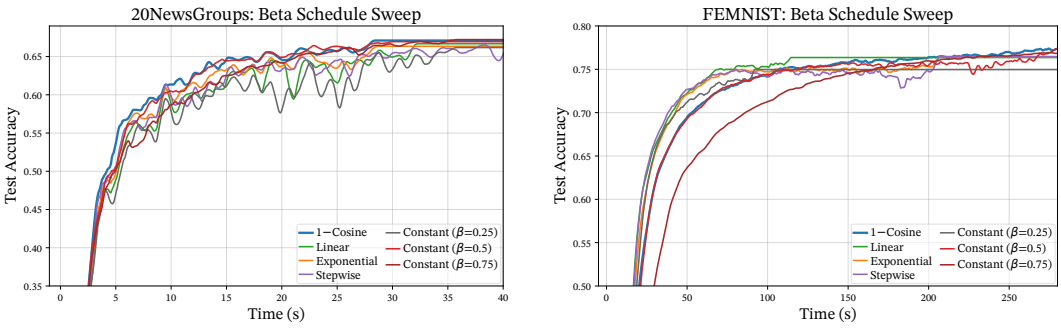
Figure 12: CIFAR-10 update-component ablation under adaptive and fixed mixing-weight schedules.

G.1 Comparison of Different β Functions

We additionally compare several choices of the mixing function $\beta(\tau)$ that controls the interpolation between the stale client update and the KD-revived update in Eq. (3). The purpose of this comparison is to evaluate how sensitive FedRevive is to the specific functional form of the staleness-dependent weighting.

In particular, we consider both adaptive schedules and constant baselines. The adaptive schedules include several monotone families that increase the reliance on the KD component as staleness grows, while the constant baselines apply the same mixing weight at every staleness level. Our main experiments use the $1 - \cos\text{ine}$ schedule. We find that the method is generally robust to the choice of β, and that even simple constant values around 0.5 perform reasonably well. However, adaptive schedules consistently provide the best overall trade-off, since they preserve more of the original client update when staleness is small and rely more on KD only when the update becomes sufficiently outdated.

Fig. 13 reports the detailed comparison. Overall, these results show that FedRevive is not highly sensitive to the exact choice of β, but benefits from using a schedule that increases smoothly with staleness rather than relying on a large constant value throughout training.



(a) 20NewsGroups. (b) FEMNIST.

Figure 13: **Comparison of different $\beta(\tau)$ functions.** Training curves under different adaptive and constant mixing schedules on 20NewsGroups and FEMNIST. The method is generally robust to the choice of β, while adaptive schedules provide the best overall trade-off by relying more on the raw client update at low staleness and shifting more weight to the KD-revived update as staleness increases.

H Server Memory Footprint of FedRevive

Bounded additional server memory. FedRevive does not increase communication cost relative to standard asynchronous FL, but it does introduce bounded additional server-side memory. Beyond the current global model $\mathbf{x}^{(t)}$, the server stores (i) a teacher buffer $\mathcal{C}$ containing c recently received client models, and (ii) the generator parameters $\boldsymbol{\theta}$. Importantly, this overhead is controlled by fixed hyperparameters and does not grow with the total number of clients or the number of server rounds.

I Privacy Discussion and Synthesized Samples

FedRevive does not introduce any additional client-side disclosures beyond the standard uploaded model updates already used in asynchronous FL. In particular, the server does not access raw client data, private labels, or any extra metadata beyond the regular model uploads also required by FedAvg-style methods. The pseudo-samples used by KD-Revive are produced entirely on the server and serve only as an internal transfer set for knowledge distillation.

At the same time, we do not claim a new formal privacy guarantee. Our point is narrower and empirical: in our setting, the uploaded client updates are the result of batched local optimization over multiple steps and are further separated from the current server state by asynchronous delay. This communication interface appears sufficient to expose intermediate predictive structure that supports knowledge transfer, but not enough to synthesize realistic reconstructions of client images. Representative synthesized samples are shown in Fig. 14; they are typically low-fidelity and not visually interpretable as realistic client examples.

This observation is consistent with the role of the synthetic data in FedRevive: the samples are not intended to approximate the real client distribution or serve as a usable training dataset on their own, but rather to provide intermediate variables on which stale client models and the current server model can exchange predictive information through DFKD.

J Experimental Details and Hyperparameters

Compute resources and runtime. All experiments are conducted on one of A100, V100, and H100 GPUs. Each experiment is run on a single GPU, and total runtime ranges from 2 to 20 hours. The dominant cost arises not from model training itself but from the sequential simulation of asynchronous federated learning with 1000 clients, which is required to realistically model client delays and staleness. This simulation introduces additional overhead compared to standard centralized or synchronous training, but does not require distributed multi-GPU execution. Memory requirements fit within standard GPU capacities (e.g., 16–80GB), depending on the model.

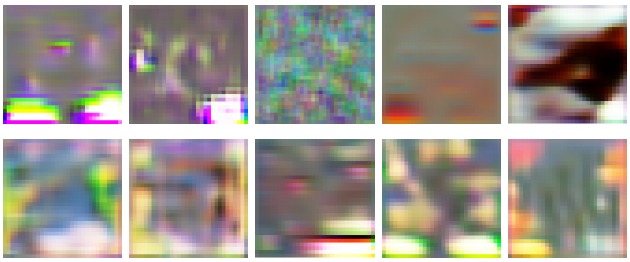


Figure 14: **Representative synthesized pseudo-samples used by FedRevive.** The samples are generated on the server for data-free knowledge distillation and are not intended to be realistic reconstructions of private client data. In our setting, regular asynchronous FL updates appear to provide enough signal for knowledge transfer, but not enough to recover visually realistic client images.

Data partitioning and seeds. We simulate 1000 clients. For all tasks except for FEMNIST, each client holds 350 training examples, and client datasets are sampled using a non-iid Dirichlet partition. For FEMNIST the clients follow natural user partition of the dataset. We use 15% of the full dataset for validation, 15% for test. All results are averaged over three random seeds. Our experiments assume benign clients. Extending FedRevive to address data or model poisoning is left for future work.

Client optimization. Clients train with Adam ($\beta_1 = 0.9$), local batch size 32, and 25 local iterations per client update. In asynchronous runs, 100 clients are active concurrently out of 1000 total clients. In synchronous runs, 100 clients join a single round.

Learning rates. We tune the client learning rate η_c and server learning rate η_s per method and dataset. Table 6 reports the final choices. For AFL-DW, we use the same learning rates as FedRevive. The "Buffer / Window" column reports the buffer size used by buffered async methods (FedBuff, FADAS, FedAC) and the sliding-window size used by FedFa; methods that aggregate every upload (FedAvg, AFL-DW, DD-FedRevive, FedRevive) leave it empty. Method-specific hyperparameters that are not learning rates (e.g., FADAS delay threshold τ_c, FedAC momentum coefficients) are listed next.

FADAS hyperparameters. FADAS uses an AMSGrad-style server optimizer with $\beta_1 = 0.9$, $\beta_2 = 0.99$, and $\epsilon = 10^{-8}$. The delay threshold τ_c that gates the adaptive step is set to $\tau_c = 3$ on CIFAR-10, CIFAR-100, and FEMNIST, and $\tau_c = 2$ on 20NewsGroups.

FedAC hyperparameters. FedAC uses a Nadam-style server optimizer with prospective momentum coefficients $\beta_1 = 0.6$, $\beta_2 = 0.9$, and $\epsilon = 10^{-8}$. Following the original formulation, client weights are derived from the cosine similarity between the global model drift and each client's pseudo-gradient (clipped at zero and normalized). We adopt the Basic-WTG-APWM variant and drop the client-side gradient correction so that all methods share the same client training procedure for a fair comparison.

FedFa hyperparameters. FedFa aggregates with a fixed-size sliding window of $K = 2$ uploads on every dataset; the global model is refreshed on every upload once the window is full, rather than waiting for K *new* updates as in FedBuff.

KD augmentation settings. Both KD-based variants use a teacher buffer of size $c = 8$ and a KD batch size of 32.

DD-FedRevive hyperparameters. We run 10 distillation steps per server update, and fix the mixing coefficient to $\beta = 0.5$. Table 7 reports the student learning rate used inside KD and the distillation temperature.

FedRevive hyperparameters. We use synthesis batch size 64 and KD batch size 32. Per server update, we perform 10 KD steps and update the generator for 2 steps before the meta-learning step, once every 10 server updates. The KD temperature is fixed to T $= 1$. For the synthesis loss in

Table 6: Client and server learning rates used in each experiment, together with the buffer / window size for buffered async methods.

Dataset	Method	η_c	η_s	Buffer / Window
CIFAR-10	FedAvg	0.0003	1.4	–
	FedBuff	0.0003	0.05	2
	FADAS	0.0003	0.004	2
	FedAC	0.0003	0.0001	2
	FedFa	0.0003	0.03	2
	AFL-DW	0.0003	0.10	–
	DD-FedRevive	0.0003	0.20	–
	FedRevive	0.0003	0.10	–
CIFAR-100	FedAvg	0.0003	1.4	–
	FedBuff	0.0003	0.10	3
	FADAS	0.0003	0.003	3
	FedAC	0.0003	3e-5	3
	FedFa	0.0003	0.03	2
	AFL-DW	0.0003	0.10	–
	DD-FedRevive	0.0001	0.30	–
	FedRevive	0.0003	0.10	–
FEMNIST	FedAvg	0.001	1.4	–
	FedBuff	0.001	0.025	1
	FADAS	0.001	0.003	1
	FedAC	0.001	0.0001	2
	FedFa	0.001	0.015	2
	AFL-DW	0.001	0.05	–
	DD-FedRevive	0.001	0.10	–
	FedRevive	0.001	0.05	–
20NewsGroups	FedAvg	0.003	1.0	–
	FedBuff	0.001	0.1	3
	FADAS	0.001	0.005	3
	FedAC	0.001	0.0003	3
	FedFa	0.001	0.03	2
	AFL-DW	0.001	0.1	–
	DD-FedRevive	0.001	0.1	–
	FedRevive	0.001	0.1	–

Table 7: Hyperparameters for DD-FedRevive. Shared: 10 distillation steps per server update, distillation batch size 32, teacher buffer size $c = 8$, and $\beta = 0.5$ (constant for DD-FedRevive).

Dataset	KD learning rate	Distillation temperature
CIFAR-10	0.0003	2.0
CIFAR-100	0.0001	2.0
FEMNIST	0.0003	2.0
20NewsGroups	0.0001	2.0

Eq. (4), we fix the adversarial and target alignment weights to $\alpha_{\text{adv}} = 0.1$ and $\alpha_{\text{target}} = 1.0$, and set the feature matching weight α_{feature} per dataset (Table 8). For the aggregation mixing, FedRevive uses the adaptive schedule described in the main text.

Feature statistics used in $\mathcal{L}_{\text{feature}}$. The feature loss in Eq. (4) matches architecture-specific intermediate statistics via $\phi(\cdot)$. For the vision backbones (ResNet-18 on CIFAR-10/CIFAR-100 and the CNN on FEMNIST), we follow standard DFKD practice and use BatchNorm statistics. For the transformer model (20NewsGroups with T5-small), we use hidden-states at the position of classification head (first token).

Table 8: Dataset-specific DFKD hyperparameters for FedRevive. Shared: synthesis batch size 64, KD batch size 32, teacher buffer size 8, KD temperature T $= 1$, 2 generator steps every 10 server updates, and 10 KD steps per server update. Learning rates (LR) are presented in the table.

Dataset	**KD LR**	**Generator LR**	**Latent noise LR**	α_{feature}
CIFAR-10	1e-4	0.003	0.001	0.003
CIFAR-100	3e-5	0.003	0.001	0.0003
FEMNIST	1e-4	0.01	0.003	0.3
20NewsGroups	3e-4	0.01	0.1	0.0001

J.1 β Scheduling Families

We consider several candidate monotone scheduling families for β, which determines how quickly FedRevive shifts from parameter-space aggregation to KD-dominated aggregation as staleness increases. Fig. 15 visualizes the families we considered. Unless stated otherwise, we use the $1 - \cos$ine schedule in all experiments. The choice is explained in detail in Appendix G.1.

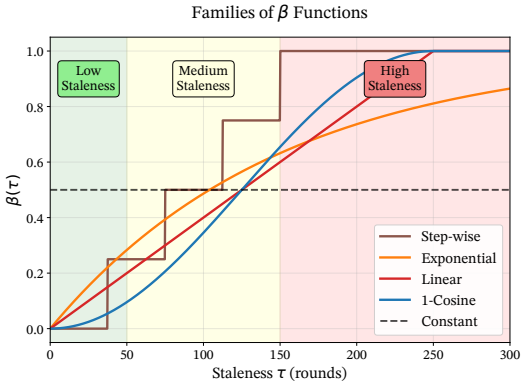


Figure 15: Different candidate β scheduling functions controlling the mix between parameter aggregation and DFKD updates. $1 - \cos$ine schedule is chosen, as it empirically shows the best performance on FEMNIST and 20NewsGroups.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and Section 1 state the scope and contributions of FedRevive, and the claims are supported by the method description in Section 2 and the empirical results in Section 3 and Appendices.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The paper discusses the main limitations and assumptions of FedRevive, including server-side overhead and deployment regimes in Section 2 and 3, and Appendix C, D.1, and I.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [N/A]

Justification: The paper does not introduce formal theoretical results or proofs; its contributions are algorithmic and empirical.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: The experimental setup, datasets, models, baselines, time simulation, seeds, and hyperparameters are described in Section 3 and Appendix D, J. The code is publicly available at `https://github.com/askinb/FedRevive`.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in

some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: The code is publicly available at `https://github.com/askinb/FedRevive`. The experiments use standard public datasets described in Section 3, with implementation and hyperparameter details in Appendix J. The `README` file in the code folder describes how to run experiments with the proposed method and the baselines.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: Training and evaluation details, including datasets, data partitioning, client participation, optimization settings, validation/test splits, and method-specific hyperparameters, are given in Section 3 and Appendix J.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: Main results are averaged over three random seeds, and the training curves report shaded standard-error regions, as described in Section 3.

   Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Compute resources, GPU types, memory range, and per-run runtime are reported in Appendix J. Server-side FLOPs and runtime overhead are further analyzed in Appendix C and D.1.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The study uses standard public benchmark datasets and simulation-based FL experiments, does not involve human-subject data collection.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

Answer: [Yes]

Justification: This work is foundational algorithmic research on asynchronous federated learning and is not tied to a specific deployment or application domain; privacy-related considerations are discussed in Appendix I.

Guidelines:

- The answer [N/A] means that there is no societal impact of the work performed.
- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

Answer: [N/A]

Justification: The paper does not release high-risk pretrained models, image generators for public use, scraped datasets, or other assets with substantial dual-use risk.

Guidelines:

- The answer [N/A] means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: The paper uses standard benchmark datasets and model architectures, which are cited in Section 3.

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [Yes]

Justification: The new asset is the implementation of FedRevive and the experimental scripts, publicly available at `https://github.com/askinb/FedRevive` with the information needed to reproduce the reported results.

Guidelines:

- The answer [N/A] means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [N/A]

Justification: The paper does not involve crowdsourcing, user studies, or new data collection from human subjects.

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [N/A]

Justification: The paper does not involve crowdsourcing, user studies, or new data collection from human subjects, so IRB approval or equivalent review is not applicable.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [Yes]

Justification: A small language model is used only in the 20NewsGroups text experiment to generate short synthetic snippets for warm-starting latent prompt embeddings in the prompt-based synthesis procedure. This usage is described in Section 3.2 under "Text task results."

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.