FedSAN: Replayable DP Accounting for Asynchronous Federated LLM Tuning under Out-of-Order Updates

Anonymous ACL submission

Abstract

Asynchronous federated learning (FL) is a practical default for scaling instruction tuning of large language models (LLMs), but out-of-order (OOO) arrivals break the audit assumptions of deployed differential privacy (DP) accounting. We show that arrival-order accounting can mis-attribute snapshot-tied updates to privacy events, yielding a ledger that is not replayable from execution provenance and can violate strict budgets. We propose FedSAN-LLM: clients attach authenticated snapshot provenance, and the server deterministically reconstructs a snapshot-consistent released event stream for reconstructed-order DP accounting (R-RDP), producing an auditable ledger under OOO delivery; optionally, clients apply randomized subspace projection as DP post-processing to compress privatized LoRA updates without changing the privacy guarantee. On Llama-3-8B LoRA tuning for PubMedQA and FiQA, FedSAN-LLM matches near-synchronous accuracy while providing a $2.6\times$ wall-clock speedup over synchronized training with small audit deviation.

1 Introduction

Federated learning (FL) is increasingly used to *instruction-tune* large language models (LLMs) across organizations in high-stakes domains (e.g., healthcare and finance) without centralizing raw text (Ning et al., 2025; McMahan et al., 2016). At LLM scale, however, **synchronization latency** becomes a dominant systems cost: heterogeneous compute and network conditions create stragglers, leaving accelerators idle in synchronous rounds (Wei and Liu, 2025). Asynchronous FL is therefore a common engineering choice, updating the server continuously under delayed and out-of-order (OOO) arrivals to improve throughput (Alasmari et al., 2025).

Problem: OOO arrivals break deployable DP accounting. Privacy-sensitive FL typically relies on differential privacy (DP) with clipping/noise and a *stateful* accountant (e.g., RDP) that composes a sequence of subsampled mechanisms (Shi et al., 2025; Abadi et al., 2016). In asynchronous training, each client update is computed on a *server-issued snapshot* but may arrive OOO (Li et al., 2025b; Xie et al., 2025). The prevailing implementation—*arrival-order accounting*—increments the accountant by arrival index (Zhao et al., 2025). Under OOO, this can **mis-attribute snapshot-tied events**: a straggler update generated from an old snapshot is charged as a fresh event, so the *claimed* ledger no longer matches the *realized* computation schedule. This gap is invisible without audit replay and becomes critical under strict budgets ($\epsilon_{\text{target}} \leq 2$), where even small bookkeeping errors translate into either silent budget violations or severe utility collapse when corrected. We quantify auditability by the audit deviation $\Delta\varepsilon_{\text{audit}} := \left|\varepsilon_{\text{replay}} - \varepsilon_{\text{ledger}}\right|$, where $\varepsilon_{\text{ledger}}$ is the privacy loss reported by the system, and $\varepsilon_{\text{replay}}$ is recomputed by deterministically replaying the released event stream from immutable logs under the declared accountant. We mark FAIL if either (i) $\varepsilon_{\text{replay}} > \varepsilon_{\text{target}}$ (budget violation) or (ii) $\Delta\varepsilon_{\text{audit}} > \tau$ (non-replayable ledger).

Approach: provenance-aligned accounting with optional compression. We propose FEDSAN-LLM, which makes DP accounting **provenance-aligned** under OOO delivery. Each update carries authenticated snapshot provenance; the server buffers arrivals and *deterministically* releases updates in a snapshot-consistent order, composing privacy on this reconstructed stream (**R-RDP**). This yields a *replayable* ledger: auditors can reconstruct the released stream from logs and reproduce the accountant state exactly. To reduce LLM-scale uplink cost, FEDSAN-LLM optionally applies randomized projection *after* the DP mechanism; by DP post-processing, projection does not affect the privacy guarantee and changes only payload size.

Contributions.

- **Audit failure mode.** We show that under OOO arrivals, arrival-indexed DP ledgers can mis-attribute snapshot-tied events, yielding non-replayable accounting traces (large $\Delta\epsilon_{\text{audit}}$) and degraded utility under strict budgets.
- **Method.** We introduce FEDSAN-LLM, which reconstructs a snapshot-consistent released event stream from authenticated provenance and performs reconstructed-order accounting (R-RDP), producing a replayable DP ledger under asynchrony.
- **LLM evidence under strict DP.** On Llama-3-8B LoRA tuning for PubMedQA and FiQA with $(\epsilon{=}2, \delta{=}10^{-5})$, FEDSAN-LLM achieves near-synchronous accuracy while providing a $2.6\times$ wall-clock speedup over synchronized training and a small $\Delta\epsilon_{\text{audit}}$ under severe disorder.

2 Related Work

Federated instruction tuning and PEFT for LLMs. Recent FL research has progressed from small models to adapting foundation models on decentralized text, including federated instruction tuning and task adaptation with parameter-efficient fine-tuning (PEFT) such as LoRA/adapters(Liu et al., 2025; Qin et al., 2025). Most existing PEFT-based FL pipelines, however, implicitly assume *round-consistent* execution (synchronous aggregation or bounded staleness) so that the training log naturally induces a well-defined sequence of server steps and evaluation checkpoints(Gao et al., 2025). Asynchrony is widely used to improve throughput under stragglers, but prior systems primarily optimize convergence/latency and do not specify a *verifier-facing accounting interface* that remains valid when updates are generated from stale snapshots and delivered out of order(Ghiasvand et al., 2025; Li et al., 2025c). FedSAN-LLM targets this missing piece: it provides an explicit, replayable DP accounting trace for high-throughput asynchronous LoRA fine-tuning under severe out-of-order (OOO) arrivals.

Differential privacy accounting in federated learning under asynchrony. Differential Privacy (DP) in FL is typically realized via clipping and Gaussian noise, with privacy tracked by stateful accountants such as RDP under a prescribed sampling model(Koskela and Mohammadi, 2025). These accountants compose privacy over a *logical event stream* (i.e., a sequence of randomized mechanisms with specified parameters)(Wang et al., 2025). In practice, many asynchronous implementations advance the accountant in *arrival order* , effectively treating network arrivals as the DP event sequence(Zhao et al., 2025). The key gap is that under OOO delivery, the arrival stream can diverge from the realized computation schedule implied by snapshot provenance, making the recorded ledger *non-replayable* from execution logs(Price, 2025). While some theoretical works study noisy/asynchronous optimization or bounded staleness, they rarely address *system-level auditability*: whether an independent verifier can reproduce the claimed privacy expenditure from the logged artifacts(Ma et al., 2025). Our work focuses precisely on this auditability gap and introduces provenance-driven reconstruction so the accountant consumes the replayed, snapshot-consistent event stream rather than the raw arrival stream.

Auditable and trustworthy federated learning. A growing line of work aims to make FL trustworthy by verifying aspects of the training pipeline, e.g., using Trusted Execution Environments (TEEs) or cryptographic proofs to attest that clients executed prescribed computations(Li et al., 2025a; Xing et al., 2025). These approaches primarily verify *computation integrity* (did the client run the code / produce a valid update?), which is complementary to our goal(Guan et al., 2025). FedSAN-LLM verifies *accounting integrity*: did the server charge privacy according to the same logical DP events that were actually released and applied? By defining an append-only audit log and a deterministic replay rule, our method yields a verifier-replayable DP ledger under OOO arrivals(Chen et al., 2025). This accounting layer is orthogonal to, and composable with, computation-verification mechanisms (TEE/ZK), enabling end-to-end auditable DP-FL systems.

3 Method

We present FEDSAN-LLM, an asynchronous federated instruction-tuning framework that produces a *replayable* DP accounting trace under out-of-order (OOO) arrivals. We first define private federated LoRA fine-tuning and then formalize why arrival-indexed accounting is not auditable in asyn-

chronous deployments.

3.1 Preliminaries: Private Federated Instruction Tuning

Federated LoRA fine-tuning. We consider K clients; client i holds a private instruction dataset $\mathcal{D}_i = \{(x, y)\}$. The server hosts a frozen LLM backbone Θ_{frozen} and trains only LoRA adapters and the task head, denoted by $\theta \in \mathbb{R}^{d'}$ with $d' \ll d$. The objective is

$$\ell_i(\theta) := \mathbb{E}_{(x,y)\sim\mathcal{D}_i}\Big[-\log P\big(y \mid x; \Theta_{\text{frozen}}, \theta\big)\Big], \tag{1}$$

$$\min_{\theta}\ \mathcal{L}(\theta) = \sum_{i=1}^{K} p_i\, \ell_i(\theta). \tag{2}$$

where p_i is a client weight.

Round-level DP event (one ledger entry per released round). At logical round s, the server samples a subset of clients $\mathcal{S}_s$ with rate $q = |\mathcal{S}_s|/K$ and broadcasts snapshot (θ_s, s). Each selected client $i \in \mathcal{S}_s$ computes a local update $\Delta_{i\to s}$ and applies the Gaussian mechanism

$$\widetilde{\Delta}_{i\to s} = \text{Clip}(\Delta_{i\to s}, C) + \mathcal{N}(0, \sigma^2 C^2 \mathbf{I}). \tag{3}$$

The server forms a *round aggregate* (e.g., mean) over the received privatized updates for this snapshot, $\bar{\Delta}_s = \frac{1}{|\mathcal{R}_s|}\sum_{i\in\mathcal{R}_s} \widetilde{\Delta}_{i\to s}$, where $\mathcal{R}_s \subseteq \mathcal{S}_s$ is the subset of clients whose updates are *released* for round s. **DP accounting is performed at the round level:** each *released* round s contributes exactly one DP event to the ledger under the public parameters (q, C, σ, δ). Dropped (unreleased) client updates do not affect the model and are not charged to the ledger; drop decisions are logged for audit replay.

3.2 Failure Mode: Arrival-Indexed Accounting Is Not Auditable under OOO

Logical DP events vs. network arrivals. A DP accountant composes privacy over a sequence of *logical DP events*—each event corresponds to a server-applied update (i.e., a released step) under a specified sampling model (rate q) and mechanism parameters (Eq. (3)). In asynchronous FL, however, updates are generated from *server-issued model snapshots* and may arrive OOO due to heterogeneous delays. Crucially, the arrival stream is a property of the network, while the DP ledger is meant to reflect the *server's realized computation schedule* (which updates were actually applied and in what logical order).

Standard practice: arrival-order accounting. Many asynchronous implementations increment the accountant whenever an update arrives, effectively defining the event sequence by the arrival permutation $\pi(\cdot)$:

$$\varepsilon_{\text{arr}}(T) = \text{Compose}\big(\mathcal{M}_{\pi(1)}, \ldots, \mathcal{M}_{\pi(T)}\big), \tag{4}$$

where each $\mathcal{M}_{\pi(t)}$ is treated as the t-th DP event.

Auditability gap and audit deviation. A privacy *audit* requires that an independent verifier can reproduce the claimed accountant state from immutable execution logs. Let ϵ_{ledger} denote the privacy loss reported by the system during training, and let ϵ_{replay} denote the privacy loss recomputed by a verifier who *replays the released event stream* from logs under the same public DP parameters (q, C, σ, δ) and the declared accountant. We define the audit deviation as

$$\Delta\varepsilon_{\text{audit}} := \big|\varepsilon_{\text{replay}} - \varepsilon_{\text{ledger}}\big|. \tag{5}$$

We mark FAIL if either $\varepsilon_{\text{replay}} > \varepsilon_{\text{target}}$ or $\Delta\varepsilon_{\text{audit}} > \tau$.

Large $\Delta\varepsilon$ indicates that the claimed ledger is not reproducible from execution logs, hence the system cannot *demonstrate* compliance with the intended privacy budget under OOO arrivals (see Section 4).

Takeaway. The issue is not that DP composition is "order-sensitive" in general, but that asynchronous deployments require the accountant to consume the *correct logical DP event stream*. When arrivals decouple from snapshot provenance, arrival-indexed accounting yields a ledger that is not replayable and thus not auditable.

What the verifier replays. Our audit assumes that the server publishes an append-only log containing, for each received update: (i) authenticated provenance (s, τ), (ii) a hash of the payload u_i, and (iii) the deterministic release/drop decisions. Given the public DP parameters and the declared accountant, a verifier reconstructs the released event stream $\pi^\star$ and recomputes ϵ_{replay}. Auditability is exactly the requirement that ϵ_{replay} matches the claimed ϵ_{ledger} up to tolerance τ.

3.3 FedSAN-LLM: Reconstructed-Order Accounting with Projected Updates

To close the auditability gap under out-of-order (OOO) arrivals, FEDSAN-LLM (i) logs snapshot

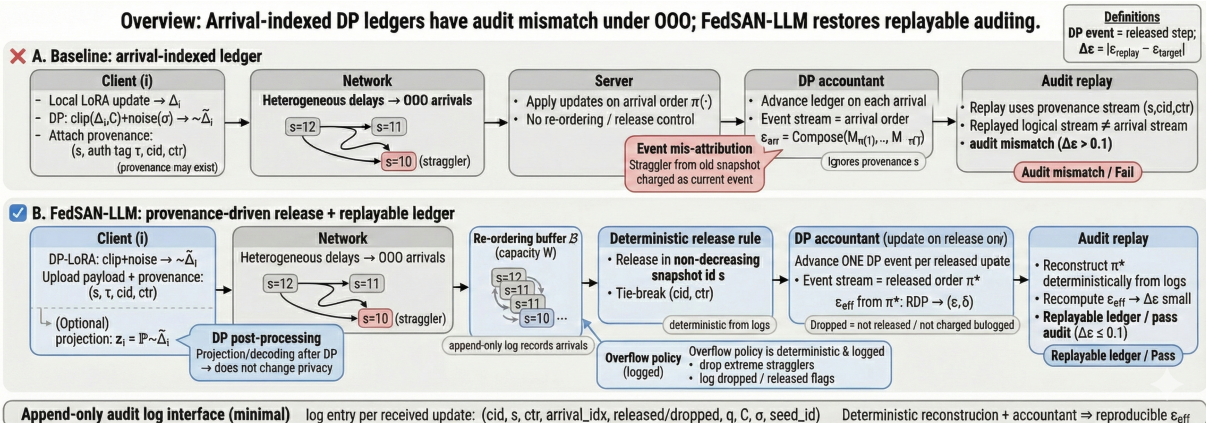

Figure 1: **Why arrival-indexed DP ledgers are not replayable under OOO, and how FedSAN-LLM restores auditable accounting.** (A) Baseline advances the accountant on arrivals, causing event mis-attribution and audit mismatch ($\Delta\epsilon > 0.1$). (B) FedSAN-LLM logs authenticated provenance and advances the accountant only on deterministically released updates in snapshot-consistent order, yielding a replayable ledger ($\Delta\epsilon \leq 0.1$).

provenance and reconstructs a *snapshot-consistent* DP event stream for accounting (R-RDP), and (ii) optionally compresses privatized LoRA updates via projection to reduce uplink cost. Importantly, accounting robustness depends only on provenance, while projection is pure post-processing.

3.3.1 R-RDP: Provenance-Driven Reconstruction of DP Events

Provenance metadata. When client i downloads the current adapter, the server attaches a monotone snapshot id s and an authentication tag $\tau = \mathrm{Auth}(s, \texttt{cid})$. The client uploads (u_i, s, τ), where u_i is the (optionally compressed) privatized update payload. Authenticated provenance prevents clients from re-labeling an update as coming from a different snapshot.

Deterministic release and ledger update. The server maintains (i) a bounded buffer $\mathcal{B}$ and (ii) a pointer $\hat{s}$ for the latest *accounted and applied* snapshot id. Upon each arrival, the update is inserted into $\mathcal{B}$. The server then repeatedly releases the unique buffered update whose provenance matches the *next expected* snapshot id $s = \hat{s}+1$ (ties broken deterministically, e.g., by arrival hash). For each *released* update, the server (a) applies the model update and (b) advances the accountant by *one* DP event using the public parameters (q, C, σ, δ). Thus, the DP ledger is defined by the released sequence $\pi^\star$ (the realized computation schedule), not by network arrival order.

Stale arrivals and overflow policy. Any update with $s < \hat{s}$ is *stale* (its target snapshot has already been accounted) and is dropped immediately; it is neither applied nor charged to the ledger. If inserting a new update makes $\mathcal{B}$ overflow, the server deterministically drops the most *future* update (largest s, ties by $(\texttt{cid}, \texttt{ctr})$) and logs the drop. Dropped updates are not released and thus consume zero privacy budget; drop/release decisions are append-only logged for audit replay.

Overflow, replayability, and budget compliance. Charge iff released. The server maintains a buffer $\mathcal{B}$ of capacity W and a pointer $\hat{s}$ to the latest *accounted* snapshot. Upon each arrival, the update is buffered; the server releases buffered updates in deterministic, snapshot-consistent order. An update is charged *iff* it is *released and applied*; the accountant is advanced exactly once per released step. **Missing snapshots and drops.** If $\mathcal{B}$ is full and no update with $s = \hat{s} + 1$ appears within a fixed timeout, the server deterministically marks snapshot $\hat{s} + 1$ as *dropped* (logged), advances $\hat{s} \leftarrow \hat{s} + 1$ *without* advancing the accountant, and continues. Any later arrival with provenance $s \leq \hat{s}$ is treated as stale and discarded (logged), and is neither applied nor charged, preventing extreme stragglers from perturbing either the model state or the ledger. **Replayability guarantee.** Given authenticated provenance, an append-only audit log (including $(\texttt{cid}, s, \texttt{ctr}, \texttt{arrival\_idx}, \texttt{released/dropped})$ and public DP parameters (q, C, σ, δ)), and a deterministic release/drop rule, any third party can replay the log to recover the released stream $\pi^\star$ and reproduce the accountant state exactly. **Compliance by policy.** Replayability makes the

ledger verifiable; budget compliance is enforced operationally by a release policy (e.g., stop releasing once $\varepsilon(\mathcal{A}) \geq \varepsilon_{\text{target}}$ or pre-calibrate a maximum number of releases), ensuring $\varepsilon_{\text{eff}} \leq \varepsilon_{\text{target}}$ in the realized execution.

3.3.2 Projected Privatized LoRA Updates for LLM-Scale Uplink

Client-side projection (optional). Let $\widetilde{\Delta}_i \in \mathbb{R}^{d'}$ be the privatized LoRA update from Eq. (3). To reduce communication, the client transmits a sketch

$$z_i = P\,\widetilde{\Delta}_i \in \mathbb{R}^m, \qquad m \ll d', \tag{6}$$

where P is generated from a public seed broadcast by the server (per run/window).

Server-side decoding and post-processing. Upon release, the server forms a decoded update $\widehat{\Delta}_i = \text{Decode}(z_i; P)$ (identity if projection is disabled) and applies it. Because projection/decoding is applied *after* the DP mechanism, it is DP post-processing and does not change the privacy guarantee or the accountant inputs (same (q, C, σ, δ)).

Decoupling accounting from compression. R-RDP uses only provenance metadata to reconstruct $\pi^\star$; therefore, enabling/disabling projection affects bandwidth and runtime but leaves accounting robustness unchanged.

Client pulls $(\theta, s, \mathsf{seed})$, computes Δ_i, applies DP to get $\widetilde{\Delta}_i$, optionally sends $u_i = P\widetilde{\Delta}_i$ (sketch), and uploads $(u_i, s, \tau, \texttt{cid}, \texttt{ctr})$.

Algorithm 1 FedSAN-LLM server pipeline (round-level reconstructed-order accounting)

Require: Target $(\varepsilon_{\text{target}}, \delta)$; DP params (q, C, σ); step size η; target clients/round m; reorder window W; optional projection dim m_{proj}
Init: $\theta \leftarrow \theta_0$; next round id $\hat{s} \leftarrow 0$; round-bucket buffer $\mathcal{B}[\cdot] \leftarrow \emptyset$; accountant $\mathcal{A} \leftarrow \textsc{Init}(\delta)$
Broadcast: public seed (defines P if projection enabled)
while training is active **do**
  Receive payload $u_{i\to s}$ with provenance $\texttt{meta} = (\texttt{cid}_i,\ s,\ \tau)$
  Verify τ; if invalid then discard
  if $s < \hat{s}$ **then** ▷ late for an already released round
    Log drop; continue
  end if
  if $s > \hat{s} + W$ **then** ▷ outside reorder window
    Log drop (or evict by policy); continue
  end if
  Insert $u_{i\to s}$ into round bucket $\mathcal{B}[s]$
  while $\textsc{ReadyToRelease}(\mathcal{B}[\hat{s}], m)$ **do**
    $\bar{u}_{\hat{s}} \leftarrow \textsc{Aggregate}(\mathcal{B}[\hat{s}])$ ▷ one round aggregate (mean/sum)
    $\mathcal{B}[\hat{s}] \leftarrow \emptyset$
    $\mathcal{A} \leftarrow \textsc{Accumulate}(\mathcal{A}; q, C, \sigma)$ ▷ **one DP event per released round**
    $\widehat{\Delta} \leftarrow \textsc{Decode}(\bar{u}_{\hat{s}})$ ▷ identity if no projection; otherwise decode using seed
    $\theta \leftarrow \theta + \eta\,\widehat{\Delta}$; $\hat{s} \leftarrow \hat{s} + 1$
    if $\varepsilon_{\text{effective}}(\mathcal{A}) \geq \varepsilon_{\text{target}}$ **then**
      stop (or adjust schedule)
    end if
  end while
end while

4 Experiments

We evaluate FedSAN on privacy-sensitive NLP with LLMs (LoRA finetuning) in biomedical QA and financial sentiment analysis. We ask: (1) Under strict privacy budgets ($\epsilon \leq 2$) and severe out-of-order (OOO) arrivals, can FEDSAN preserve both utility and an auditable DP ledger? (2) Can projection reduce LLM-scale uplink cost *without changing* the accounting pipeline or audit outcome?

4.1 Experimental Setup

Tasks and Datasets. We select two high-stakes, privacy-sensitive domains:

- **PubMedQA (Biomedical):** We cast PubMedQA as 3-way classification (Yes/No/Maybe) over question–context pairs and report Accuracy.
- **FiQA (Financial):** We cast FiQA as label-based sentiment classification and report Accuracy.

We use the official train/dev/test splits and keep dev/test centralized for evaluation only.

Instruction-style formulation and evaluation. To avoid treating the tasks as mere classifier training, we cast both datasets as *instruction-following generation*. Each example is formatted as an instruction prompt and the model is trained with the standard next-token loss to generate a short answer string from a constrained label set. For PubMedQA, the target output is one of {`Yes`, `No`, `Maybe`}; for FiQA, the target output is one of {`Positive`, `Negative`, `Neutral`}. At evaluation, we decode with greedy decoding and map the first generated label token to the class. We report (i) **Label EM/Accuracy** and (ii) **Format adherence** (percentage of outputs that match the required label format), which is critical for downstream deployment of instruction-tuned models.

LLM Backbone and Federated Protocol. We use **Llama-3-8B** with LoRA (rank r=16, α=32)

and update only LoRA adapters and the task head (the backbone is frozen). We simulate cross-silo FL with K=200 clients and sample m=10 clients per logical round (q=m/K=0.05) for T=500 rounds. Each selected client performs the same number of local steps with identical optimizer hyperparameters across all methods (AdamW; details in Appendix) so that methods differ only in (i) arrival/release policy and (ii) accounting basis.

Wall-clock Measurement (Auditable and Comparable). All methods run on the same 8×NVIDIA A100 (80GB) cluster with identical code paths for model forward/backward, local training, DP clipping/noise, and communication serialization. Reported wall-clock time *includes* communication, buffering/release, and accounting. To ensure fair comparison under buffering, we normalize time by completing the same number of *released server steps* (i.e., updates that are actually applied to θ and charged to the DP ledger).

Communication metric (uplink, released-update normalized). We report *uplink only* (client→server), since downlink broadcasting of the current LoRA adapter is identical across methods under the same logical released steps. For server step t, let R_t be the set of client updates that are *released and applied* at step t (typically $|R_t| = m$). We define the per-step uplink as

$$\mathrm{Comm/step}(t) := \sum_{i \in R_t} \Big(\mathrm{bytes}(u_i) + \mathrm{bytes}(\mathrm{meta}_i) \Big), \tag{7}$$

where u_i is the uploaded payload (raw DP update or projected sketch), and meta_i includes authenticated provenance (e.g., $(\mathrm{cid}, s, \mathrm{ctr}, \tau, \mathrm{seed})$). Dropped/expired updates are *not applied* and *not charged* to the DP ledger; therefore they are excluded from communication totals (but their drop events are logged for audit replay). Total uplink over T released steps is $\sum_{t=1}^{T} \mathrm{Comm/step}(t)$.

Asynchrony and OOO Delivery. We emulate asynchrony by assigning each client update a virtual delay δ_i and defining an arrival stream at the server. We control OOO severity via a disorder ratio $\rho \in [0, 1]$; unless noted otherwise, we use $\rho = 0.5$ (severe disorder). Baselines apply and account updates in arrival order $\pi(\cdot)$, whereas FEDSAN releases updates in a reconstructed snapshot-consistent order $\pi^\star(\cdot)$ via R-RDP.

Differential privacy and accounting view. We enforce client-level DP with ℓ_2 clipping (threshold C) and Gaussian noise, and track privacy using an RDP accountant with $\delta = 10^{-5}$. We target a strict regime $\varepsilon_{\text{target}} = 2.0$. During training, the system reports $\varepsilon_{\text{ledger}}$ from its accountant state. For auditability, we also compute $\varepsilon_{\text{replay}}$ by deterministically replaying the *released and applied* event stream from logs under the same public parameters (q, C, σ, δ) and the declared accountant, and report the audit deviation $\Delta\varepsilon_{\text{audit}} := |\varepsilon_{\text{replay}} - \varepsilon_{\text{ledger}}|$. We mark FAIL if either $\varepsilon_{\text{replay}} > \varepsilon_{\text{target}}$ or $\Delta\varepsilon_{\text{audit}} > \tau$.

Crucially, the accountant's input event stream differs by method: arrival-order baselines feed events indexed by network arrivals, whereas FedSAN advances accounting only on deterministically released updates in snapshot-consistent reconstructed order.

Baselines. We compare against: (i) **Sync DP-FedAvg** (upper bound): round-consistent composition under perfect synchronization; (ii) **Async DP-FedAvg (Arrival)**: applies and accounts updates in arrival order (the common deployment choice); (iii) **DP-FedBuff**: buffered asynchronous aggregation with buffer size K (we use K=10), adapted to DP by accounting upon buffer release. All baselines use the same DP mechanism and hyperparameters; they differ only in buffering/release and accounting basis.

4.2 Main Results: Utility, Efficiency, and Auditability under Strict Privacy

We evaluate Llama-3-8B (LoRA) under a strict budget (ϵ_{target}=2.0, δ=10^{-5}) with severe out-of-order (OOO) arrivals. Table 1 reports task utility, wall-clock efficiency, and audit deviation. To make auditability operational, we report the audit deviation $\Delta\varepsilon_{\text{audit}} := \big|\varepsilon_{\text{replay}} - \varepsilon_{\text{ledger}}\big|$, where $\varepsilon_{\text{replay}}$ is recomputed by deterministic replay of the released event stream from logs under the declared accountant. A method is marked FAIL if it violates the target budget ($\varepsilon_{\text{replay}} > \varepsilon_{\text{target}}$) or if the ledger is not reproducible ($\Delta\varepsilon_{\text{audit}} > \tau$).

FedSAN is the only method that jointly achieves high utility and a reproducible (auditable) ledger. In Table 1, FEDSAN achieves $73.8\% \pm 0.4$ on PubMedQA and $70.2\% \pm 0.5$ on FiQA, within one point of the synchronous upper bound ($74.5\% \pm 0.3$, $71.0\% \pm 0.4$), while keeping $\Delta\varepsilon_{\text{audit}} = 0.06$ (Pass). By contrast, arrival-order

Table 1: **Main Results on PubMedQA and FiQA (Llama-3-8B + LoRA).** We report Accuracy, wall-clock time, and audit deviation $\Delta\varepsilon_{\text{audit}} := \left|\varepsilon_{\text{replay}} - \varepsilon_{\text{ledger}}\right|$, where $\varepsilon_{\text{replay}}$ is recomputed by replaying the released event stream from logs under the declared accountant. FAIL if $\varepsilon_{\text{replay}} > \varepsilon_{\text{target}}$ or $\Delta\varepsilon_{\text{audit}} > \tau$. Times use identical logical rounds/local steps and include communication+buffering on the same hardware.

Method	**Accounting Basis**	**Accuracy (%)**		**Efficiency**		**Audit $\Delta\epsilon$**
		PubMedQA	**FiQA**	**Time (h)**	**Speedup**	
Target Privacy Budget $\epsilon_{target} = 2.0$ (Strict Regime)						
Sync DP-FedAvg (Upper Bound)	Round-Consistent	74.5 ± 0.3	71.0 ± 0.4	82.0	$1.0\times$	0.01
Async DP-FedAvg (Arrival)	Arrival Order	58.2 ± 1.2	55.0 ± 1.5	**24.5**	**3.3**$\times$	3.82 (Fail)
DP-FedBuff ($K = 10$)	Arrival Order	65.1 ± 0.8	62.4 ± 0.9	26.8	$3.0\times$	2.15 (Fail)
FedSAN (Ours)	**R-RDP**	**73.8** $\pm$ **0.4**	**70.2** $\pm$ **0.5**	31.2	$2.6\times$	**0.06** (Pass)

Table 2: **Instruction-following evaluation under strict DP (ϵ=2.0).** We cast PubMedQA/FiQA as constrained label generation and report label exact match (EM, %) and format adherence (%). Format is the fraction of outputs whose *first non-whitespace token* belongs to the required label set ({Yes, No, Maybe} for PubMedQA; {Positive, Negative, Neutral} for FiQA), case-insensitive.

Method	**PubMedQA**		**FiQA**	
	EM	**Format Adh.**	**EM**	**Format Adh.**
Sync DP-FedAvg	74.5	99.8	71.0	99.5
Async (Arrival)	58.2	64.2	55.0	59.5
DP-FedBuff	65.1	81.7	62.4	76.3
FedSAN (Ours)	**73.8**	**99.1**	**70.2**	**98.8**

accounting suffers both large accuracy degradation ($58.2/55.0$) and a non-reproducible ledger ($\Delta\epsilon = 3.82$, Fail). DP-FedBuff improves utility over naive arrival-order asynchrony ($65.1/62.4$) but still fails audit ($\Delta\epsilon = 2.15$), indicating that buffering *without* provenance-consistent event reconstruction does not resolve accounting integrity under OOO.

Why baselines fail under strict DP, and why FEDSAN does not. Under OOO delivery, arrival-order baselines define DP events by network arrivals rather than by snapshot-tied *released* steps, so the accountant is driven by an event stream that cannot be deterministically replayed from logged provenance; this mismatch surfaces as budget violations ($\epsilon_{\text{replay}} > \epsilon_{\text{target}}$) and large audit deviation $\Delta\varepsilon_{\text{audit}}$. FEDSAN instead enforces a snapshot-consistent released order (R-RDP), making the accounting trace reproducible from the same logs and yielding small deviation and stable utility under identical DP parameters. Crucially, the audit failure is reflected in model behavior: beyond accuracy, constrained label generation exposes instruction-following collapse, where arrival-order methods frequently violate the required output schema (Format $64.2\%/59.5\%$ on PubMedQA/FiQA; Table 2), whereas FEDSAN preserves near-perfect adherence (Format 99.1%/98.8%), consistent with a replayable ledger. Finally, FEDSAN retains most of the async benefit ($2.6\times$ wall-clock speedup, 31.2h vs. 82.0h) while passing audit; the faster arrival-order baseline (24.5h) is not comparable in strict-compliance settings because it fails the budget, leaving FEDSAN as the only method that jointly delivers near-upper-bound utility, strict-budget audit pass, and substantial speedup under severe OOO.

4.3 System Robustness and Component Analysis

We analyze the system-level trade-offs of reconstructed-order release and validate the necessity of FEDSAN's components under strict DP and severe OOO delivery.

4.3.1 The Cost of Integrity: Buffer Size Trade-off

A common concern is that re-ordering buffers may negate the throughput benefits of asynchrony. We vary the buffer window W under severe disorder ($\rho = 0.5$). Table 3 reports (i) the fraction of arrivals that are not released before timeout (Dropped), (ii) wall-clock time normalized to W=0 for the *same* arrival stream, (iii) test Accuracy, and (iv) audit pass/fail.

A practical "sweet spot" for auditable asynchrony. W=0 maximizes throughput but fails audit due to event-stream inconsistency under OOO arrivals. As W increases, more updates can be released in a snapshot-consistent order, restoring auditability at the cost of buffering latency. We

Table 3: **Impact of Re-ordering Buffer Size (W) on Efficiency and Auditability.** Llama-3-8B under severe disorder ($\rho = 0.5$). Dropped updates are not applied and are not charged to the DP ledger; drop events are logged for audit replay. Rel. Time is normalized to W=0 under the same disorder process.

Buffer Size (W)	Dropped (%)	Rel. Time	Accuracy (%)	Audit
0 (Pure Async)	0.0%	**1.00**×	58.2	✗Fail
5	12.4%	1.15×	69.5	✓Pass
15 (Ours)	**2.1%**	**1.28**×	**73.8**	✓**Pass**
50	0.1%	2.50×	74.2	✓Pass
∞ (Sync)	0.0%	3.30×	74.5	✓Pass

find that W=15 achieves a favorable trade-off: it passes audit with only 1.28× wall-clock overhead relative to unsafe pure async, while matching near-upper-bound utility (73.8% vs. 74.5%). Larger windows further reduce drop rate but quickly approach synchronous latency (e.g., W=50 and $W \to \infty$), indicating diminishing returns for throughput.

4.3.2 Robustness under Strict Privacy Budgets

We next test whether the gains persist in high-noise regimes by sweeping $\epsilon_{\text{target}} \in \{0.5, 1.0, 2.0, 4.0\}$ on FiQA (Table 4). For each ϵ_{target}, all methods tune σ to target the same budget under their respective accounting view, and we report test Accuracy.

Table 4: **Utility across privacy regimes on FiQA.** We sweep $\epsilon_{\text{target}} \in \{0.5, 1.0, 2.0, 4.0\}$ with $\delta = 10^{-5}$.

Privacy Budget	Accuracy (%) on FiQA		
	Sync (Upper)	Async (Arrival)	FedSAN (Ours)
$\epsilon = 0.5$ (Extreme)	52.0	28.5 (Random)	**50.8**
$\epsilon = 1.0$ (Strict)	64.5	41.2	**63.1**
$\epsilon = 2.0$ (Standard)	71.0	55.0	**70.2**
$\epsilon = 4.0$ (Relaxed)	73.5	68.0	**72.8**

Correct event attribution is required to learn under high noise. As privacy tightens ($\epsilon \leq 1.0$), arrival-order baselines degrade sharply, reaching near-random performance at $\epsilon = 0.5$. In contrast, FEDSAN remains close to the synchronous upper bound across the sweep (e.g., 50.8% vs. 52.0% at $\epsilon = 0.5$). This indicates that when the signal-to-noise ratio is low, accounting on a snapshot-consistent event stream becomes necessary to preserve useful learning under OOO arrivals.

4.3.3 Ablation: Necessity of R-RDP and Practicality of Projection

Finally, Table 5 isolates the two components: reconstructed-order accounting (R-RDP) and projected updates.

Table 5: **Ablation on Llama-3-8B components.** Full method uses R-RDP + projection. Communication reports *uplink only* and is measured as the sum of serialized bytes of the m=10 *released and applied* client updates per server step (downlink is identical across variants). For our LoRA configuration (Appendix B), the raw DP update is $\approx$ 54.4MB per client (FP32), i.e., $\approx$ 544MB/step for m=10. With projection d_{proj}=4096, each client uploads a $\approx$ 16KB FP32 sketch, i.e., $\approx$ 160KB/step for m=10 (metadata $<$ 1KB/update).

Variant	R-RDP	Proj.	Acc. (%)	Comm./step	Audit
FedSAN (Full)	✓	✓	73.8	160 KB	✓
w/o Projection	✓	✗	74.1	544 MB	✓
w/o R-RDP	✗	✓	59.5	160 KB	✗

R-RDP is necessary for auditability; projection is necessary for feasible communication. Removing projection preserves auditability and yields similar accuracy (74.1%), but increases uplink volume from $\approx$ 160KB/step to $\approx$ 544MB/step (for m=10), which is impractical for typical WAN links in cross-silo deployments. Thus, FEDSAN is minimal in the sense that R-RDP provides accounting integrity, while projection enables LLM-scale communication without affecting the accounting pipeline.

5 Conclusion

As FL moves to LLM-scale tuning, practical deployments must reconcile asynchronous throughput with strict, auditable DP compliance. We showed that under out-of-order delivery, arrival-indexed accounting can decouple the logged event stream from the realized snapshot provenance, yielding large accounting deviation and degraded utility under strict budgets. We proposed FEDSAN-LLM, which (i) logs lightweight provenance and reconstructs a snapshot-consistent composition order (R-RDP) to make the DP ledger replayable from execution logs, and (ii) optionally applies post-processing projection to reduce uplink cost without changing the privacy guarantee. Experiments with Llama-3-8B LoRA on biomedical QA and financial sentiment demonstrate near-synchronous utility while passing strict audits ($\epsilon \leq 2$, $\delta = 10^{-5}$) with modest buffering overhead.

6 Limitations

Log integrity and threat model. Our auditability guarantee is conditional on the integrity of provenance logs (snapshot ids, counters, and DP parameters). We assume an append-only log or externally verifiable logging (e.g., client-signed entries / transparency-style storage). Handling a fully malicious server that can selectively drop/rewrites log entries is outside our scope.

Bounded disorder and buffering. R-RDP relies on a bounded reordering window: with extreme, non-stationary outages, a finite buffer may overflow, causing dropped updates or additional waiting. We log drop/overflow events for replay, but designing adaptive buffering and release policies with formal guarantees under heavy-tailed delays is left to future work.

PEFT-centric evaluation. Our LLM experiments focus on PEFT (LoRA) where communication is feasible; while R-RDP is mechanism-agnostic, full-parameter fine-tuning would require substantially stronger compression and systems support than evaluated here.

DP utility on minority/long-tail clients. Strict DP can amplify the under-representation of clients with small datasets or rare patterns. Quantifying this effect under more diverse client distributions and broader instruction-tuning objectives is an important direction.

References

Martın Abadi, Andy Chu, Ian Goodfellow, H. Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. 2016. Deep learning with differential privacy. *arXiv (Cornell University)*.

Sultan Alasmari, Rayed AlGhamdi, Ghanshyam G Tejani, Sunil Kumar Sharma, and Seyed Jalaleddin Mousavirad. 2025. Federated learning-based multimodal approach for early detection and personalized care in cardiac disease. *Frontiers in Physiology*, 16:1563185.

Chunlu Chen, Ji Liu, Haowen Tan, Xingjian Li, Kevin I-Kai Wang, Peng Li, Kouichi Sakurai, and Dejing Dou. 2025. Trustworthy federated learning: privacy, security, and beyond. *Knowledge and Information Systems*, 67(3):2321–2356.

Zhidong Gao, Zhenxiao Zhang, Yuanxiong Guo, and Yanmin Gong. 2025. Federated adaptive fine-tuning of large language models with heterogeneous quantization and lora. In *IEEE INFOCOM 2025-IEEE Conference on Computer Communications*, pages 1–10. IEEE.

Sajjad Ghiasvand, Yifan Yang, Zhiyu Xue, Mahnoosh Alizadeh, Zheng Zhang, and Ramtin Pedarsani. 2025. Communication-efficient and tensorized federated fine-tuning of large language models. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 24192–24207.

Zhangshuang Guan, Yulin Zhao, Zhiguo Wan, and Wei Wang. 2025. Input integrity and authentic results: Towards trustworthy aggregation in federated learning. In *IEEE INFOCOM 2025-IEEE Conference on Computer Communications*, pages 1–10. IEEE.

Antti Koskela and Jafar Aco Mohammadi. 2025. Auditing differential privacy guarantees using density estimation. In *2025 IEEE Conference on Secure and Trustworthy Machine Learning (SaTML)*, pages 1007–1026. IEEE.

Ding Li, Ziqi Zhang, Mengyu Yao, Yifeng Cai, Yao Guo, and Xiangqun Chen. 2025a. Teeslice: Protecting sensitive neural network models in trusted execution environments when attackers have pre-trained models. *ACM Transactions on Software Engineering and Methodology*, 34(6):1–49.

Tianyu Li, Dezhi Han, Jiatao Li, and Kuan-Ching Li. 2025b. Asynchronous tiered federated learning storage scheme based on blockchain and ipfs. *Computers, Materials & Continua*, 83(3).

Xingyu Li, Lu Peng, Yu-Ping Wang, and Weihua Zhang. 2025c. Open challenges and opportunities in federated foundation models towards biomedical healthcare. *BioData Mining*, 18(1):2.

Xiao-Yang Liu, Rongyi Zhu, Daochen Zha, Jiechao Gao, Shan Zhong, Matt White, and Meikang Qiu. 2025. Differentially private low-rank adaptation of large language model using federated learning. *ACM Transactions on Management Information Systems*, 16(2):1–24.

Qianpiao Ma, Xiaozhu Song, Junlong Zhou, Haibo Wang, Yunming Liao, Jianchun Liu, and Hongli Xu. 2025. Asynchronous federated learning over non-iid data via over-the-air computation. *IEEE Transactions on Networking*.

H. B. McMahan, Eider Moore, Daniel Ramage, S. Hampson, and B. A. Y. Arcas. 2016. Communication-efficient learning of deep networks from decentralized data.

Wanyi Ning, Jingyu Wang, Qi Qi, Haifeng Sun, Daixuan Cheng, Cong Liu, Lei Zhang, Zirui Zhuang, and Jianxin Liao. 2025. Federated fine-tuning on heterogeneous loras with error-compensated aggregation. *IEEE Transactions on Neural Networks and Learning Systems*.

Elowen Price. 2025. Federated learning for privacy-preserving edge intelligence: A scalable systems perspective. *Journal of Computer Science and Software Applications*, 5(5).

Zhen Qin, Zhaomin Wu, Bingsheng He, and Shuiguang Deng. 2025. Federated data-efficient instruction tuning for large language models. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 15550–15568.

Yifan Shi, Kang Wei, Li Shen, Yingqi Liu, Xueqian Wang, Bo Yuan, and Dacheng Tao. 2025. Towards the flatter landscape and better generalization in federated learning under client-level differential privacy. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Shaowei Wang, Sufen Zeng, Jin Li, Shaozheng Huang, and Yuyang Chen. 2025. Shuffle model of differential privacy: Numerical composition for federated learning. *Applied Sciences*, 15(3):1595.

Lanye Wei and Zhao Liu. 2025. Light weight blockchain with iot devices to secure smart non-fungible tokens using hybrid secure functions. *Scientific Reports*, 15(1):31633.

Renyou Xie, Chaojie Li, Zhaohui Yang, Zhao Xu, Jian Huang, and ZhaoYang Dong. 2025. Differential privacy enabled robust asynchronous federated multitask learning: A multigradient descent approach. *IEEE Transactions on Cybernetics*.

Zhibo Xing, Zijian Zhang, Ziang Zhang, Zhen Li, Meng Li, Jiamou Liu, Zongyang Zhang, Yi Zhao, Qi Sun, Liehuang Zhu, and 1 others. 2025. Zero-knowledge proof-based verifiable decentralized machine learning in communication network: A comprehensive survey. *IEEE Communications Surveys & Tutorials*.

Xiaopeng Zhao, Xiao Bai, Guohao Sun, and Zhe Yan. 2025. Asynchronous federated learning with local differential privacy for privacy-enhanced recommender systems. *IEEE Internet of Things Journal*.

A Experimental Details

A.1 Hyperparameters and Reproducibility

Shared settings (fair comparison). Unless stated otherwise, all methods share the same backbone, LoRA configuration, client sampling, local training steps, optimizer, and DP mechanism/accountant. The *only* difference across methods is the *server-side ingestion/order policy* (arrival order vs. reconstructed-order release), and whether uplink projection is enabled.

Randomness control. We report mean±std over three seeds. Seeds are fixed for (i) client sampling, (ii) non-IID partitioning, and (iii) the asynchrony simulator (delay draws). For each seed, we run all baselines with the *same* partitions and delay realizations to isolate method effects.

A.2 Datasets, Prompts, and Client Partitioning

Datasets and splits. We use PubMedQA (biomedical QA; Yes/No/Maybe) and FiQA (financial sentiment; Positive/Negative/Neutral) with standard train/test splits. Metrics are **Accuracy** on both tasks (discrete label prediction).

Client partitioning (non-IID). Training data are partitioned to $K{=}200$ clients using Dirichlet non-IID with concentration $\alpha{=}0.5$. We keep per-client sample counts fixed across methods within the same seed.

Prompt templates.

```
PubMedQA:
Context: {context}
Question: {question}
Answer (Yes/No/Maybe):

FiQA:
Headline: {sentence}
Sentiment (Positive/Negative/Neutral):
```

Decoding / label mapping. We use greedy decoding with a constrained label set and map the first generated label token to the target class. Outputs outside the label set are counted as incorrect (this is rare under instruction tuning).

B Compute, Timing, and Communication Accounting

Hardware/software. Experiments run on a private cluster with 8×NVIDIA A100 (80GB) and AMD EPYC 7763 CPUs, implemented in PyTorch with `transformers+peft`. All wall-clock numbers include client compute, communication, server buffering/release, and DP accounting, and are measured under the *same cluster* and the *same number of released server steps*.

Timing protocol (apples-to-apples). To avoid unfair comparisons (e.g., some methods terminating early), we fix $T{=}500$ *released/applied* server steps for all methods. For buffered methods, updates that are not released (dropped/expired) do not count toward T. Reported wall-clock time is the elapsed time to complete T released steps under the same asynchrony realization.

Communication metric (explicit contract). Unless otherwise specified, "communication" reports **uplink only** (client→server), because downlink (server→clients) is identical across methods under the same logical steps in our implementation. For step t, let $\mathcal{R}_t$ be the set of *released* updates applied by the server (typically $|\mathcal{R}_t|{=}m$). We define:

$$\text{Comm/step} = \sum_{i \in \mathcal{R}_t} \Big(\text{bytes}(u_i) + \text{bytes}(\text{meta}_i) \Big) \tag{8}$$

$$\text{Total Comm} := \sum_{t=1}^{T} \text{Comm/step}. \tag{9}$$

Here u_i is the uploaded payload (raw DP update or projected sketch), and $\text{meta}_i = (\texttt{cid}, s, \texttt{ctr}, \tau, \texttt{seed})$ contains authenticated provenance and projection seed; metadata is included but is <1KB/update and negligible at the reported scale. Dropped/expired updates are excluded from both model application and communication totals (but are logged for replay; see below).

Payload sizes (closed-form and measured). For our LoRA configuration (rank 16 on `q_proj,v_proj`), the trainable adapter has ≈13.6M parameters. We transmit DP-noised updates in FP32 for audit conservatism. Thus, the *raw* uplink per client update is:

$$S_{\text{raw}} \approx 13.6\text{M} \times 4 \text{ bytes} \approx 54.4\text{MB}.$$

With projection, each client transmits a length-d_{proj} FP32 sketch, so:

$$S_{\text{proj}} \approx d_{\text{proj}} \times 4 \text{ bytes}.$$

For $d_{\text{proj}}{=}4096$, $S_{\text{proj}} \approx 16$KB per client update (plus negligible metadata/seed). Therefore, with $m{=}10$ released updates per step:

- **Raw uplink (no projection):** $\text{Comm/step} \approx m \cdot S_{\text{raw}} \approx 10 \times 54.4\text{MB} \approx 544\text{MB}$.

Table 6: **Hyperparameters (FedSAN-LLM and baselines).**

Item	Value
Federated training	
Clients K / sampled per step m	200 / 10 (Poisson subsampling rate q=0.05)
Released server steps T	500 released (applied) steps
Local epochs E / batch size B	1 / 4
Max sequence length	512 (truncate longer inputs)
Backbone & PEFT	
Backbone	Llama-3-8B-Instruct (frozen)
LoRA rank / alpha / dropout	r=16 / α=32 / 0.05
Target modules	`q_proj`, `v_proj`
Precision	BF16 forward; FP32 for DP noise injection and transmitted update tensors
Optimization	
Client optimizer / lr	AdamW / 3×10^{-4}
Server update	asynchronous FedAvg-style step (η=1.0)
Weight decay	0.0
Differential privacy	
Unit of privacy	client-level (one participating client update = one DP event)
Clip / (ϵ, δ)	C=1.0 / $(2.0, 10^{-5})$
Noise multiplier σ	calibrated via RDP to match ϵ at T=500
RDP orders	$\lambda \in \{2, 3, \ldots, 64\}$ (min over λ)
FedSAN	
Re-order buffer W / disorder ρ	15 / 0.5 (main)
Projection dim d_{proj}	4096 (if enabled)
Tie-break for equal snapshot id	deterministic by $(\texttt{cid}, \texttt{ctr})$

- **Projected uplink (d_{proj}=4096):** $\text{Comm/step} \approx 10 \times$ 16KB $\approx$ 160KB.

All reported communication numbers are also verified by measuring serialized tensor bytes transmitted on the wire; the measured values match the above closed-form estimates within small overhead (serialization + metadata).

C Asynchrony Model and Disorder Control

Event-time simulation. We emulate asynchronous arrivals via per-client timestamps $\tau_{\text{arr}} = \tau_{\text{issue}} + t_{\text{comp}} + t_{\text{comm}} + \delta_{\text{net}}$, where t_{comp} captures compute heterogeneity, t_{comm} depends on payload size, and δ_{net} models heavy-tailed network delay. The simulator is calibrated to realistic straggler statistics (median/tails) and uses identical delay realizations across methods within each seed.

Disorder ratio. We report a disorder ratio $\rho \in [0, 1]$ defined as the fraction of updates that would be *stale* under pure async ingestion (W=0), i.e., updates that arrive after the server would have already advanced by at least one additional released step. We set ρ=0.5 for main experiments (severe disorder).

Drop and audit semantics (replayable execution). FEDSAN uses a bounded buffer (W) and a deterministic, snapshot-consistent release rule. On overflow, extreme stragglers are dropped by a fixed policy; dropped updates are neither applied nor charged, and each drop (with provenance) is logged. An auditor can replay the release/drop trace from logs to recompute ϵ_{replay} and verify that the ledger matches the realized execution.

Table 7: **Projection dimension vs. uplink and utility (PubMedQA).** Uplink is reported *per client update* (FP32): $4 \cdot d_{\text{proj}}$ bytes for sketches, and ≈ 54.4MB for raw LoRA updates under our configuration.

d_{proj}	Uplink / client	Acc. (%)
None	54.4MB	73.8
16384	64KB	73.7
4096	16KB	73.6
1024	4KB	72.1

D Additional Ablation: Projection Dimension

We study the impact of the projection dimension on the uplink–utility trade-off under the same DP budget and disorder setting. We report uplink per client update as $4 \cdot d_{\text{proj}}$ bytes (FP32 sketches).