# ORBIVFL: SYSTEM-ALGORITHM CO-DESIGN FOR ASYNCHRONOUS VERTICAL FEDERATED LEARNING

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

## ABSTRACT

Vertical Federated Learning (VFL) enables multiple parties with vertically partitioned features to collaboratively train models without directly sharing private data. However, existing VFL systems often suffer from substantial communication and synchronization overhead under heterogeneous computation and wide-area networks, while existing asynchronous approaches insufficiently exploit worker-level parallelism within each party. We present OrbiVFL, a **system–algorithm co-design framework** for practical asynchronous VFL. OrbiVFL builds on Ray and the parameter-server architecture to decouple computation and communication across workers and parties, while explicitly aligning embeddings and gradients through unique identifiers. To reduce communication-induced stalls, we derive inter-party local update intervals and design a semi-asynchronous synchronization mechanism for heterogeneous workers within each party. We further provide convergence guarantees for the proposed training scheme. We evaluate OrbiVFL on four representative datasets, including UCIHAR, Mini-ImageNet, VisionTouch, and MuJoCo. Compared with state-of-the-art VFL approaches, OrbiVFL reduces end-to-end training time by up to 76.43% while maintaining comparable or higher model accuracy, and improves computational resource utilization by 76.35 percentage points. Extensive ablation studies further demonstrate the effectiveness and robustness of our design. Finally, we deploy OrbiVFL in a production-oriented embodied-intelligence scenario, where it achieves 26.30% training-time reduction without sacrificing accuracy under realistic computation.

## 1 INTRODUCTION

In the field of machine learning, preserving data privacy has become a major challenge. In particular, the strict implementation of regulations such as the General Data Protection Regulation (GDPR) (European Parliament and Council, 2016) has imposed rigorous restrictions on data aggregation and usage, leading to the emergence of "data silos." To address these challenges, Federated Learning (FL) (McMahan et al., 2016; Liu et al., 2024) has emerged as a promising distributed machine learning paradigm. More recently, a new paradigm, Vertical Federated Learning (VFL) (Yang et al., 2019; Qi et al., 2022; Liu et al., 2022; Wang et al., 2023), has become a more pragmatic approach to cross-silo collaboration. It facilitates joint model training among parties possessing disparate features of a shared user base (Huang et al., 2023; Xing et al., 2024). VFL has been widely applied in domains such as e-commerce, healthcare, and smart home services. For example, e-commerce companies can collaborate with banks to build user credit prediction models by integrating transaction histories with banking data; similarly, hospitals can leverage diverse clinical datasets to obtain a more comprehensive understanding of patients' health conditions (Sun et al., 2019); VFL can also facilitate the orchestration of IoT devices to better support smart home services while strictly preserving data privacy (Peng et al., 2024).

In current VFL research (Shen et al., 2025), researchers have effectively leveraged Parameter Server (PS)-based parallelism for global aggregation, leading to the development of prominent industrial frameworks (e.g., FATE (Liu et al., 2021) and PaddlePaddle (PaddlePaddle, 2026)). However, in practical wide-area network (WAN) settings, such as the integration of VFL with embodied intelligence (Zhou & Wang, 2022), synchronous training can suffer from substantial communication overhead. Compared to model computation, communication delay has a greater impact on arrival time of intermediate results, while worker imbalance, queuing, and synchronization introduce additional de-



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

lays, causing communication to become a major bottleneck. To alleviate this issue, existing studies have explored asynchronous VFL. For example, VAFL relaxes inter-party synchronization (Chen et al., 2020); C-VFL and CELU-VFL reduce communication overhead through compression and cached computation (Castiglia et al., 2022b; Fu et al., 2022); BS-VFL combines bounded-staleness pipelining with local-step selection (Wang et al., 2025); and PubSub-VFL combines parameter exchanges for intra-party communication (Liu et al., 2025).

However, existing solutions still face the following challenges. (i) System-level coupling remains prevalent. In particular, VFL ID alignment (He et al., 2022) introduces significant coordination challenges, causing existing parameter-server based VFL architectures to suffer from inter-party waiting bottlenecks. (ii) Beyond resource and data heterogeneity in VFL, existing systems do not sufficiently coordinate computational resources across multiple parties and multiple workers within each party. As a result, available compute resources are often underutilized, leading to substantial disparities in execution time across parties and further exacerbating communication-induced waiting. Therefore, building a practical asynchronous VFL system requires not only carefully balancing communication efficiency and model accuracy, but also addressing the above challenges.

To address all the aforementioned challenges, we propose OrbiVFL, a system and algorithm co-design framework that enables asynchronous VFL to be more effectively deployed in production-level systems. At the **system level**, we leverage the distributed framework Ray (Moritz et al., 2017) to decouple system components while integrating it closely with the PS architecture (Li et al., 2014). Each embedding and gradient is associated with an independent identifier, enabling data and model training to be conveniently aligned at the worker level across multiple parties. At the **algorithmic level**, for inter-party coordination, we derive the number of local update steps required to hide system delays caused by communication; within each party, we design a semi-asynchronous mechanism to coordinate the synchronization intervals among multiple workers. Through this co-design, OrbiVFL enables highly parallel computation, improves computational resource utilization, and achieves a balance between communication efficiency and convergence stability. In addition, OrbiVFL can be seamlessly integrated with security protocols such as differential privacy to preserve the security and privacy of VFL. Our main contributions are as follows:

- We propose OrbiVFL, the first to systematically model worker-level execution across multiple parties. We build system with Ray to achieve system-level decoupling while maintaining explicit correspondence for computation and data.
- At the algorithmic level, we analyze asynchronous execution across parties to reduce communication bottlenecks and design a worker-level semi-asynchronization mechanism within parties, thereby effectively balancing model accuracy and communication overhead. We further provide a convergence analysis for our method.
- We conduct extensive experiments on four popular datasets, including UCIHAR, Mini-ImageNet, VisionTouch, and MuJoCo, to evaluate the effectiveness of OrbiVFL. Compared with state-of-the-arts, OrbiVFL achieves comparable accuracy while reducing training time up to 76.43% on Mini-ImageNet. Additional experiments further demonstrate the effectiveness of our design and the robustness of OrbiVFL.
- Finally, we extend OrbiVFL to a practical setting that integrates VFL with embodied intelligence. We collect a dedicated dataset and conduct production-level experiments, which further validate the effectiveness of our approach and demonstrate its potential for privacy-preserving embodied intelligence applications.

## 2 Related Work

**VFL and communication.** To address communication overhead in VFL, C-VFL compresses exchanged representations (Castiglia et al., 2022b), CELU-VFL reuses cached embeddings (Fu et al., 2022), and LESS-VFL (Castiglia et al., 2023) reduces communication through feature selection. Flex-VFL accommodates heterogeneous local optimization (Castiglia et al., 2022a), whereas FedVS reconstructs embeddings in the presence of stragglers (Li et al., 2023). LASER-VFL addresses missing feature blocks during both training and inference (Valdeira et al., 2024). Several solutions have also demonstrated the applicability of the Parameter Server (PS) architecture to VFL, improving resource utilization and the management of dynamically participating parties (Castiglia et al., 2022a; Wu & Xing, 2023). Among them, FATE (Liu et al., 2021) and Paddle (PaddlePaddle, 2026) are two

of the most widely used VFL frameworks based on the PS architecture. However, these approaches primarily rely on synchronous VFL, and the tight coupling between communication and system execution remains a major bottleneck limiting VFL efficiency.

**Asynchronous Vertical Federated Learning.** Asynchronous VFL training (Zhang et al., 2024) offers a promising way to alleviate communication bottlenecks caused by straggler effects and the inefficiency of strict synchronization. Prior work (Zhang et al., 2021) explores an asynchronous quasi-Newton framework for VFL to improve computational resource utilization; however, the approach is implemented in a single-machine setting rather than a practical multi-machine cluster. VAFL and AFSGD-VP relax synchronization across feature holders (Chen et al., 2020; Gu et al., 2021). BS-VFL combines bounded-staleness pipelining with local-step selection (Wang et al., 2025), but it does not address system-level decoupling or support multiple workers within each party. In multi-worker systems, PubSub-VFL combines publish/subscribe communication with local parameter servers (Liu et al., 2025), but it is limited to two parties and does not investigate inter-party staleness. Asynchronous training introduces additional system and optimization complexity. In contrast, our OrbiVFL explicitly addresses the trade-off between model accuracy and communication efficiency, provides convergence guarantees, and better exploits computational resources through joint consideration of both party-level and worker-level execution.

**VFL application.** VFL has been increasingly adopted in real-world applications where multiple parties observe different features or modalities of the same entities (Shen et al., 2025). Representative applications span human activity recognition, robotics, healthcare, emotion analysis, and multimedia analysis. In human activity recognition, heterogeneous sensor signals, such as accelerometer and gyroscope measurements, can be distributed across different parties for collaborative activity prediction (Anguita et al., 2013; Sikder & Nahid, 2021). In robotics and embodied intelligence, multimodal sensors, including cameras, force sensors, and proprioceptive sensors, naturally form vertically partitioned views of the same interaction or environment (Lee et al., 2019; 2020). Healthcare applications similarly integrate complementary patient information from demographics, medical images, and physiological signals while preserving data privacy (Johnson et al., 2016; Wagner et al., 2020). VFL has also been applied to multimodal emotion (Zadeh et al., 2016; 2018; Hasan et al., 2019; Castro et al., 2019) and multimedia analysis (Chua et al., 2009; Arevalo et al., 2017), where text, audio, video, and image features are held by different parties but correspond to the same samples. These applications demonstrate that feature- and modality-level collaboration is common in practical machine learning workloads, making VFL particularly suitable for privacy-preserving learning across heterogeneous data sources.

## 3 BACKGROUND

Here, we consider a practical setting in which multiple parties collaboratively train a machine learning model for regression or classification tasks, where a VFL framework is employed to maximize resource utilization and reduce communication overhead. Vertical federated learning (VFL) trains a joint model on different feature partitions of the same aligned examples. A *passive party* $P_k$, $k \in \mathcal{K} = \{1, \ldots, K\}$, holds features $x_{k,i}$ and a *bottom model* $h_k$. The *active party* holds targets $y_i$ and a *top model* $f_0$ that combines the parties' embeddings. Its Parameter Server $S$ coordinates training. With $N$ aligned examples and $\Theta = (\theta_0, \theta_1, \ldots, \theta_K)$ as the model parameters, multiple parties can use Private Set Intersection (PSI) method (Chen et al., 2017) to securely obtain their shared dataset without revealing any private information. Thus, the objective is

$$\min_{\Theta} \ \mathcal{F}(\Theta) = \frac{1}{N} \sum_{i=1}^{N} \ell(f_0(h_1(x_{1,i}; \theta_1), \ldots, h_K(x_{K,i}; \theta_K); \theta_0), y_i). \tag{1}$$

Raw features remain with their owners to preserve privacy. For batch $b$, parties send embeddings $Z_{k,b} = h_k(X_{k,b}; \theta_k)$; $S$ evaluates the mean batch loss $\mathcal{L}_b$ and returns $G_{k,b} = \partial \mathcal{L}_b / \partial Z_{k,b}$ to train the bottom models. The embeddings can be securely transmitted using mechanisms such as differential privacy (DP) protocols (Dwork et al., 2006), which provides a formal privacy guarantee.

Here, a *party* is a data-owning participant; a *worker* is an execution process within that participant. Party $P_k$ can run $W_k$ workers with separate replicas $\theta_{k,w}$ of its bottom model. The active party similarly runs $W_0$ top-model workers with replicas $\theta_{0,v}$. Workers of one party process different batches of that party's features; workers from different parties process complementary features of

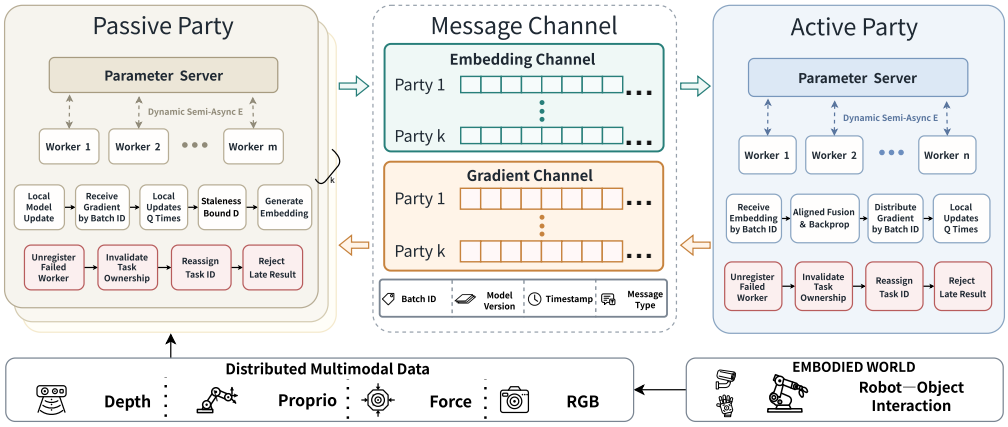



Figure 1: System architecture and workflow of our OrbiVFL. OrbiVFL incorporates a decoupled system design, where message channels and parameter servers provide corresponding batch IDs and gradient IDs to facilitate asynchronous computation and communication. At the algorithmic level, OrbiVFL employs party-level $Q$-step local updates and a worker-level Dynamic Semi-Asynchronous synchronization design.

each shared batch. Accordingly, cross-party embedding exchange and within-party replica synchronization are distinct operations.

# 4 METHOD

## 4.1 SYSTEM OVERVIEW

Figure 1 illustrates the overall workflow of our OrbiVFL framework. At the system level, OrbiVFL decouples data IDs, gradients, parties, and workers. Message channels implemented with Ray, including embedding and gradient channels, together with parameter servers, enable asynchronous communication and computation in VFL with simplified coordination. In this way, OrbiVFL reduces waiting latency, improves system concurrency, and preserves seamless data-ID alignment without imposing additional coordination overhead on workers. At the algorithmic level, for inter-party coordination, active and passive parties perform $Q$ local updates to better match heterogeneous computation capabilities and data-processing speeds across parties, thereby alleviating communication bottlenecks. For intra-party coordination, we introduce a Dynamic Semi-Asynchronous mechanism to accommodate heterogeneous worker performance and further improve resource utilization. As a result, OrbiVFL can be applied to production scenarios involving multimodal data, such as human activity recognition and embodied intelligence, substantially reducing training time and energy consumption and meeting the practical requirements.

As shown in Figure 1, notably, when a worker within a party experiences a network failure or crash, OrbiVFL can detect the failure, mark the corresponding task as invalid, and reassign it to another valid worker. Late results are rejected to preserve convergence and improve system robustness. In the following, we focus on the failure-free setting.

To protect exchanged embeddings, we incorporate Gaussian perturbations under an $(\epsilon, \delta)$-differential privacy formulation (Dwork, 2011). The guarantee is conditional on bounded sensitivity, appropriate noise calibration, and accounting for repeated releases, as specified in Appendix G.4.

Our OrbiVFL workflow is as follows. Initialize models, communication channels, and model-state tags; reset staleness counters and select the local update count $Q$. Assign aligned batches to workers through shared batch identifiers, allowing party procedures to execute concurrently without fixed worker pairing. Passive workers generate and publish embeddings when their staleness counters remain below the bound $D$; otherwise, they pause generation. An active worker collects embeddings from all passive parties for the same batch, verifies alignment, and fuses them. The active worker computes embedding gradients and immediately publishes them to the corresponding channels for workers. Active and passive workers perform local updates using cached embeddings and returned gradients, respectively, allowing computation to overlap communication. During these updates, the

**Algorithm 1** OrbiVFL workflow with algorithm-level design

**Input:** learning rate $\eta$, staleness bound $D$, intervals $E$.
Active Party and $K$ passive parties initialize model parameters and set staleness $d_{k,w} \leftarrow 0$;
Initialize channels and model-state tags; choose $Q = Q_{\min}$ by Eq. 2.
**for** epoch $e = 1, \ldots, T$ **do**
    *PassiveP_Update*(), *ActiveP_Update*();

**PassiveParty** $k = 1, \ldots, K$
**procedure** *PassiveP_Update*($k$)
    **if** receive gradient $G_{k,b}$ **then**
        Set $\theta_{k,w}^{b,0} = \theta_{k,w}$;
        **for** $j = 0, \ldots, Q-1$ **do**
            Local Update $\theta_{k,w}^{b,j+1} = \theta_{k,w}^{b,j} - \eta G_{k,w}^{b,j}$,
            **if** Dynamic Semi-Async requests aggregation at step $j$ **then**
                bottom model aggregation at $S$
        Set $\theta_{k,w} = \theta_{k,w}^{b,Q}$; $d_{k,w} \leftarrow d_{k,w} - 1$;
    **if** $d_{k,w} < D$ **then**
        Generate embedding $Z_{k,b} = h_k(X_{k,b}; \theta_{k,w})$;
        Send $Z_{k,b}$ to Channel $\mathcal{C}_{k,b}^{\text{emb}}$; $d_{k,w} \leftarrow d_{k,w} + 1$;
    **else**
        Stop embedding generation;

**Active Party**
**procedure** *ActiveP_Update*($\{Z_{1,b}, \ldots, Z_{K,b}\}$)
    Receive all embedding vectors $\{Z_{1,b}, \ldots, Z_{K,b}\}$;
    Fuse $\{Z_{1,b}, \ldots, Z_{K,b}\}$; set $\theta_{0,v}^{b,0} = \theta_{0,v}$;
    Derive $G_{k,b} = \partial \mathcal{L}_b / \partial Z_{k,b}$; send $G_{k,b}$ to Channel $\mathcal{C}_{k,b}^{\text{grad}}$, $k \in \mathcal{K}$;
    **for** $j = 0, \ldots, Q-1$ **do**
        Local Update $\theta_{0,v}^{b,j+1} = \theta_{0,v}^{b,j} - \eta \nabla_{\theta_0} \mathcal{L}_b(\theta_{0,v}^{b,j}; Z_{1,b}, \ldots, Z_{K,b})$,
        **if** Dynamic Semi-Async requests aggregation at step $j$ **then**
            top model aggregation at $S$
    Set $\theta_{0,v} = \theta_{0,v}^{b,Q}$;

Dynamic Semi-Async mechanism helps aggregate within each model group with bounded waiting. Workers save updated parameters. Passive workers decrease their staleness counters, recheck generation eligibility, and continue until training completes. The overall procedure of OrbiVFL framework is shown in Algorithm 1.

### 4.2 System Level Design

**Ray channels and identifiers.** Our OrbiVFL provides a systematic implementation of VFL; we build the execution layer with Ray actors and asynchronous message buffers (Moritz et al., 2017). Following the communication separation, logical channels $\mathcal{C}_{k,b}^{\text{emb}}$ and $\mathcal{C}_{k,b}^{\text{grad}}$ carry embeddings and their returned gradients. Ray supplies execution and object transport; our routing layer supplies these application-level channel names. Publication and consumption are independently scheduled, and received messages remain buffered until their consumer is ready.

For batch size $B$, one dataset pass contains $M = \lceil N/B \rceil$ batches, with a possibly smaller last batch. The shared ID $b$ identifies the same ordered example list $\mathcal{I}_b$ at every party. Exactly one worker per party owns $(k, b)$, while one active worker claims the matched batch. Thus $W_k$ changes parallel capacity, not $M$ or batch size. Channels remove fixed worker pairing, while preserving sample alignment and gradient ownership. OrbiVFL could also distinguish retries and reject duplicate or obsolete messages; Appendix B specifies their lifecycle.

### 4.3 Asynchronous Design

**Inter-party Semi-asynchronous Design.** We use $t^{sc}$ and $t_k^{pc}$ to denote the top- and bottom-model computation times per update, respectively; $t_k^e$ denotes the embedding generation time, and $t_k^{ec}$ denotes the one-way transmission time. Typically, $t^{sc} > t_k^{pc}$ for every party, since the active party needs to update more parameters and incurs additional computation overhead from the top model. Therefore, to prevent the active party from idling and to better utilize its computational resources, which constitute the system bottleneck, under symmetric transmission, each party must complete $Qt_k^{pc} + t_k^e + 2t_k^{ec}$ within $Qt^{sc}$. We therefore choose

$$Q(t^{sc} - t_k^{pc}) > t_k^e + 2t_k^{ec}, \qquad Q := Q_{\min} = \left\lfloor \max_{k \in \mathcal{K}} \frac{t_k^e + 2t_k^{ec}}{t^{sc} - t_k^{pc}} \right\rfloor + 1. \tag{2}$$

The floor-plus-one preserves the strict inequality. Among feasible integers, $Q_{\min}$ uses the fewest repeated updates to cover this modeled communication gap; larger $Q$ introduces additional computation and increases the reuse of stale information. For passive worker $(k, w)$, we define staleness $d_{k,w}$ as the number of embeddings sent minus the number of corresponding batches with all $Q$ local updates completed. A worker generates embeddings only when $d_{k,w} < D$; otherwise, it pauses generation and prioritizes returned gradients to complete pending updates. Appendix C proves why the selected $Q_{\min}$ is the optimal choice, denoted by $Q^*$, and further derives the corresponding condition and its extension to the multi-worker setting.

**Intra-party Semi-asynchronous Design.**

To eliminate worker-side waiting delays caused by inter-party asynchronous communication, we propose a Dynamic Semi-Asynchronous mechanism to further improve the computational efficiency of workers within each party. Our mechanism adaptively adjusts the synchronization interval during training. Specifically, as the number of training epochs increases and the model accuracy gradually approaches the target accuracy, the worker-level synchronization interval is enlarged, thereby achieving a better balance between computational efficiency and convergence stability. Specifically, the interval $E$ is defined as

$$E_e = \max\left\{1, \left\lceil \frac{E_0}{2}\left[1 - \cos\left(\pi \frac{e}{T}\right)\right]\right\rceil\right\}, \qquad E_0 \in \mathbb{N}_+, \tag{3}$$

where $E_0$ denotes the maximum worker-level synchronization interval, $e$ is the current training epoch, and $T$ is the total number of training epochs. In the early stage of training, $E$ remains small, resulting in more frequent synchronization to maintain stable learning. As training progresses, $E$ gradually increases according to the cosine schedule, reducing the synchronization frequency and allowing workers to perform more local computation between consecutive synchronizations. Toward the end of training, $E$ gradually approaches $E_0$, thereby further reducing synchronization overhead while maintaining convergence stability.

### 4.4 Convergence Analysis

Let $\mathcal{F}$ be the training objective and $\mathcal{F}^\star = \inf_\Theta \mathcal{F}(\Theta) > -\infty$. Starting from $\Theta_0$, $\Theta_R$ denotes the reference parameters after $R$ update groups, each containing $Q$ local updates. Assume $L$-smoothness and the PL condition $\|\nabla\mathcal{F}(\Theta)\|^2 \geq 2\mu[\mathcal{F}(\Theta) - \mathcal{F}^\star]$, with $0 < \mu \leq L$. Let $L$ also bound noiseless update sensitivity, $\sigma_g^2$ bound batch-gradient variance, $G$ bound parameter-change magnitudes relative to $\eta$, and $\tau$ bound initial staleness and replica differences, including aggregation effects, in local-update units. For Gaussian embedding noise, $\sigma_{\mathrm{dp}}$ is its per-coordinate standard deviation, $p$ counts perturbed coordinates, and $\kappa$ bounds update sensitivity to noise. Under the appendix assumptions, for $R \geq 1$ and $0 < \eta \leq 1/(4LQ)$,

$$\begin{aligned}\mathbb{E}[\mathcal{F}(\Theta_R) - \mathcal{F}^\star] \leq (1 - \mu\eta Q)^R[\mathcal{F}(\Theta_0) - \mathcal{F}^\star] + \frac{L\eta Q\sigma_g^2}{\mu} \\ + \frac{5}{2\mu}\left[L^2\eta^2G^2(\tau + Q - 1)^2 + \kappa^2 p\sigma_{\mathrm{dp}}^2\right].\end{aligned} \tag{4}$$

This bounds the expected loss gap in terms of $Q$ and noise scale; see Appendix D.

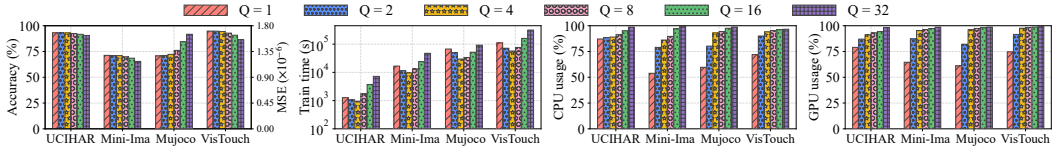
Table 1: Performance comparison on four datasets, where for Acc/MSE, bold entries identify the best predictive performance among four asynchronous approaches; for training time and CPU/GPU usage, bold denotes the best result.

| Acc / MSE ($10^{-6}$) | OrbiVFL | BS_VFL | PubSub_VFL | async_VFL | sync_VFL |
|---|---|---|---|---|---|
| UCIHAR | **93.08** | 92.86 | 92.81 | 92.19 | 93.83 |
| Mini-ImageNet | 70.96 | **71.02** | 70.83 | 70.49 | 71.54 |
| Mujoco | 1.3006 | 1.3089 | **1.2562** | 1.4226 | 1.2264 |
| VisionTouch | 94.30 | **94.57** | 94.43 | 93.89 | 95.12 |

| Train Time (s) | OrbiVFL | BS_VFL | PubSub_VFL | async_VFL | sync_VFL |
|---|---|---|---|---|---|
| UCIHAR | **945.70** | 1545.11 | 1332.27 | 1950.03 | 2077.58 |
| Mini-ImageNet | **9550.73** | 20617.18 | 17706.81 | 34118.56 | 40521.35 |
| Mujoco | **29632.27** | 41857.43 | 39402.03 | 50039.73 | 53857.46 |
| VisionTouch | **57183.09** | 98440.50 | 89443.35 | 133221.32 | 142943.47 |

| CPU usage | OrbiVFL | BS_VFL | PubSub_VFL | async_VFL | sync_VFL |
|---|---|---|---|---|---|
| UCIHAR | **89.03** | 56.71 | 68.15 | 26.70 | 19.80 |
| Mini-ImageNet | **86.11** | 67.42 | 64.67 | 20.65 | 18.41 |
| Mujoco | **93.03** | 67.61 | 63.85 | 20.09 | 19.05 |
| VisionTouch | **94.03** | 70.87 | 72.74 | 18.93 | 17.68 |

| GPU usage | OrbiVFL | BS_VFL | PubSub_VFL | async_VFL | sync_VFL |
|---|---|---|---|---|---|
| UCIHAR | **91.03** | 79.74 | 81.12 | 29.43 | 27.26 |
| Mini-ImageNet | **95.37** | 79.53 | 80.42 | 26.56 | 20.79 |
| Mujoco | **96.13** | 74.23 | 79.78 | 21.67 | 20.76 |
| VisionTouch | **97.56** | 74.17 | 79.16 | 27.81 | 24.01 |




(a) Accuracy / MSE (b) Training time (c) CPU usage (d) GPU usage

Figure 2: Effect of local steps.

# 5 Experiments

## 5.1 Experimental Setup

To evaluate the effectiveness of our OrbiVFL framework, experiments are conducted with 25 logical cores of an Intel Xeon Platinum 8470Q CPU and one RTX 5090 GPU, with four workers per party. We emulate WAN communication conditions to test our OrbiVFL by controlling the network latency between parties, the default is 0.5s among all parties. We set the maximum worker-level synchronization interval to $E_0 = 10$ by default.

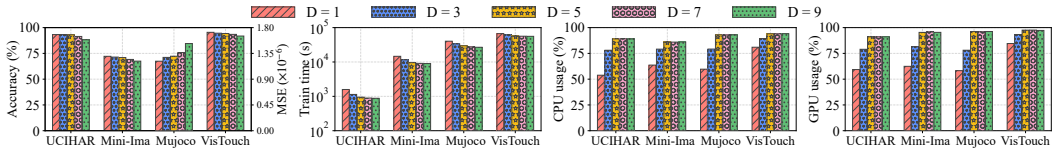
**Datasets, Models and Parameters.** We evaluate our method on four popular datasets: UCIHAR (Anguita et al., 2013), Mini-ImageNet (Vinyals et al., 2016), MuJoCo (Lee et al., 2020) and VisionTouch (Lee et al., 2019). The details of each dataset, including the number of parties and workers, model configurations, default batch size, learning rate, optimizer, and feature partitioning scheme for experiments on each dataset, are in Appendix F. The default staleness bound $D$ is 5. The default channel capacity is 15. All the results reported are averaged over five random seeds.

**Evaluation Metrics.** For effectiveness, we report classification accuracy on UCIHAR, Mini-ImageNet, and VisionTouch, and Mean Squared Error (MSE) on MuJoCo. In addition, we report training time, CPU utilization, and GPU utilization to evaluate the effectiveness of our method in mitigating communication bottlenecks and improving computational resource utilization. Detailed definitions of these metrics are provided in Appendix E.

**Baselines.** In this paper, we adopt four representative baselines: Sync-VFL (Wu et al., 2025), Async-VFL (VAFL (Chen et al., 2020)), BS-VFL (Wang et al., 2025), and PubSub-VFL (Liu et al., 2025). Sync-VFL serves as an upper bound performance on model accuracy/MSE, while BS-VFL and PubSub-VFL represent state-of-the-art approaches that specifically target communication bottlenecks in asynchronous VFL. Together, these baselines enable a comprehensive evaluation of the performance and robustness of our OrbiVFL framework and its design.

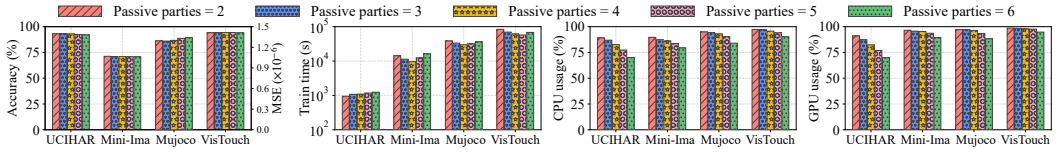
## 5.2 Overall Results

Table 1 compares OrbiVFL with four baselines across four datasets. In terms of accuracy/MSE, OrbiVFL achieves the best or near-best performance among the asynchronous methods. Even compared with Sync-VFL, the performance degradation remains small; for example, the accuracy on UCIHAR decreases by only 0.75 percentage points. In terms of training time, OrbiVFL reduces training time by up to 76.43% compared with Sync-VFL and by 46.06% compared with the strongest async baseline, PubSub-VFL, on Mini-ImageNet. These results demonstrate the effectiveness of our worker-level and party-level system–algorithm co-design in alleviating communication bottlenecks, which is particularly important for training workloads that may otherwise take several days to complete. OrbiVFL also substantially improves computational resource utilization. Compared with



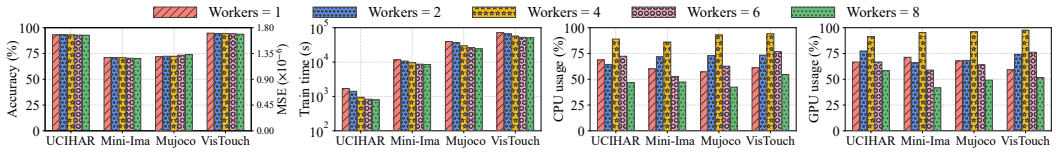

(a) Accuracy / MSE (b) Training time (c) CPU usage (d) GPU usage

Figure 3: Effect of staleness bound.




(a) Accuracy / MSE (b) Training time (c) CPU usage (d) GPU usage

Figure 4: Effect of the number of parties.




(a) Accuracy / MSE (b) Training time (c) CPU usage (d) GPU usage

Figure 5: Effect of the number of workers.

Sync-VFL, CPU and GPU utilization increase by up to 76.35 and 75.37 percentage points, respectively; compared with the strongest async baselines, the improvements reach up to 25.42 and 18.40 percentage points, respectively. These improvements remain consistent across different datasets, demonstrating that OrbiVFL can effectively exploit available computational resources while maintaining stable model performance.

### 5.3 Parameter Sensitivity Evaluation

**Local steps.** Figure 2 shows the effect of the number of local steps $Q$. As $Q$ increases, model accuracy decreases and MSE increases, but remain within an acceptable range, which is consistent with our convergence analysis. Training time first decreases and then increases, indicating that excessively large $Q$ can make model convergence more difficult and require additional training epochs. Although CPU and GPU utilization continue to increase with larger $Q$, an appropriate value of $Q$ is still necessary to balance convergence and system efficiency. In our experiments, $Q = 4$ achieves the best overall performance, which is consistent with the derived choice of $Q_{\min}$.

**Staleness bound.** Figure 3 shows the performance of OrbiVFL under different staleness bounds $D$. As $D$ increases, model accuracy decreases and MSE increases, which is consistent with our convergence analysis. Meanwhile, training time is reduced and computational resources are utilized more efficiently. OrbiVFL maintains strong performance across all evaluated values of $D$, demonstrating the robustness of our design. Nevertheless, an appropriate choice of $D$ remains important for balancing model accuracy and system efficiency.

**Number of Parties and Workers.** Figure 4 and Figure 5 show the performance of OrbiVFL under different numbers of parties and workers. As the number of passive parties increases, model accuracy decreases and MSE increases, mainly because the features become more fragmented across parties and additional synchronization is required. Training time generally first decreases and then increases. This is because moderate parallelism improves execution efficiency, whereas excessive data partitioning makes it more difficult to fully utilize CPU and GPU resources. Nevertheless, OrbiVFL continues to outperform the strongest asynchronous baseline reported in Table 1, suggesting its robustness to varying numbers of parties. As the number of workers increases, accuracy/MSE remains nearly unchanged, while training time decreases. CPU and GPU utilization reach their highest levels when the number of workers is four. Compared with the results obtained by varying the number of parties, these observations further demonstrate the effectiveness of our Dynamic Semi-Asynchronous Mechanism and the robustness of OrbiVFL under different numbers of workers. Together, these results demonstrate the practicality of OrbiVFL and its ability to effectively utilize computational resources to achieve high concurrency across diverse deployment scenarios.

Table 2: Ablation results on UCIHAR. Bold indicates the best results.

| Metric | w/o Ray Channel | w/o $E$ | w/o staleness | w/o Q | w/o multi-worker | OrbiVFL |
|---|---|---|---|---|---|---|
| Acc | 92.98 | **93.17** | 93.14 | 93.12 | 93.09 | 93.08 |
| Train Time (s) | 1457.27 | 1532.35 | 1583.43 | 1265.70 | 1702.55 | **945.70** |
| CPU usage | 57.19 | 66.13 | 53.84 | 87.12 | 68.76 | **89.03** |
| GPU usage | 78.65 | 79.83 | 59.17 | 79.01 | 66.71 | **91.03** |

Table 3: Production experiment results.

| Metric | OrbiVFL | BS_VFL | PubSub_VFL | async VFL | sync_VFL |
|---|---|---|---|---|---|
| MSE ($\times 10^{-6}$) | **2.5116** | 2.5589 | 2.5627 | 2.6120 | 2.4831 |
| Train Time (s) | **43851.76** | 62367.51 | 59497.08 | 76060.93 | 162713.59 |
| CPU usage | **91.43** | 68.16 | 64.58 | 19.89 | 18.91 |
| GPU usage | **89.92** | 70.23 | 69.71 | 22.71 | 19.64 |

**More Experiments** with performance over training time, network latency, differential privacy, larger models, batch sizes, $E_0$ and data heterogeneity are in Appendix G.

### 5.4 Ablation Study

Table 2 presents the ablation study of OrbiVFL. We observe that accuracy remains largely stable across most ablation settings, ranging from 92.98% to 93.17%, compared with 93.08% for the full system. In all other metrics, however, the complete OrbiVFL achieves the best performance, reducing training time by up to 44.45% and improving CPU and GPU utilization by up to 35.19 and 31.86 percentage points, respectively. Among the communication-related ablations, the staleness-free synchronous control has the largest impact on communication efficiency. Similarly, removing Ray-based system, disabling party-level asynchrony by setting $Q = 1$, or constraining worker-level synchronization all lead to performance degradation. These results collectively validate the effectiveness of both the system-level and algorithmic designs in OrbiVFL.

### 5.5 Production Experiments

In this paper, we deploy OrbiVFL in a production-level embodied AI system, specifically for an embodied intelligence application in automotive manufacturing. However, the embodied intelligence company cannot disclose detailed model parameters or information about its robotic arms, while the automotive manufacturer is unwilling to reveal specific production scenarios because they constitute commercially sensitive information. Therefore, we deploy both the bottom model and the top model at the embodied intelligence company, while the automotive manufacturer hosts its corresponding bottom model. The two parties collaboratively train the model to enable robotic arms to perform vehicle assembly tasks. Under OrbiVFL, both model privacy and data privacy can be preserved throughout the collaborative training process. The details of the settings are in Appendix G.2.

Table 3 shows that OrbiVFL achieves the best overall performance across all evaluated metrics compared with the asynchronous baselines. Particularly, OrbiVFL reduces training time by 73.05% for Sync_VFL while achieving substantially higher CPU and GPU utilization. Compared with Sync-VFL, the MSE increases by only 0.03, indicating a minor loss in model quality for the efficiency gains. These results demonstrate the practicality of OrbiVFL in real-world settings and its potential to support future privacy-preserving applications that integrate embodied intelligence with VFL.

## 6 Conclusion

We present OrbiVFL, a system–algorithm co-design framework for practical asynchronous Vertical Federated Learning. OrbiVFL leverages Ray and PS to decouple computation and communication across multiple parties and workers, derives inter-party local update intervals to hide communication-induced waiting, and introduces an intra-party semi-asynchronous mechanism to coordinate heterogeneous workers while bounding model staleness. Extensive experiments show OrbiVFL reduces training time by up to 76.43% and improves resource utilization while maintaining accuracy comparable to existing approaches. Our production-oriented embodied-intelligence deployment further demonstrates the practicality of OrbiVFL. Overall, our jointly optimizing system execution and training synchronization can improve the efficiency of practical VFL systems.

## AI USE STATEMENT

In this work, we used generative AI tools for [ help develop theoretical models or conceptual frameworks, formulate mathematical claims, provide critical ingredients for proving mathematical claims, assist in the writing of proofs, propose or refine hypotheses, design or provide feedback on research methodology or experiments, implement methods, assist with translation,, support qualitative and thematic data analysis, interpret results]. We have not used generative AI tools for [Generate synthetic data sets, clean and reformat dataset], and [] are not applicable to this work. Additionally, we used generative AI tools for [create or modify scientific figures or images, suggest experimental parameters, create or edit software code, creation of artifacts, draft parts of a research paper, summarize or analyse existing literature, discover research topics or identify gaps, brainstorming, sourcing/searching for information, edit a research paper to improve readability, identify relevant literature, format references, suggest a structure for a research paper, propose a title or keywords for a research paper]. We have reviewed all AI-assisted work. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

## ETHICS STATEMENT

This paper does not raise any ethical issues.

## REPRODUCIBILITY STATEMENT

We provide all source code in the supplementary material, together with the detailed configurations and hyperparameter settings required to reproduce our experimental results.

## REFERENCES

D. Anguita, Alessandro Ghio, L. Oneto, Xavier Parra, and Jorge Luis Reyes-Ortiz. A public domain dataset for human activity recognition using smartphones. In *The European Symposium on Artificial Neural Networks*, 2013. URL https://api.semanticscholar.org/CorpusID:6975432.

John Arevalo, Thamar Solorio, Manuel Montes y Gómez, and Fabio A. González. Gated multimodal units for information fusion. *ArXiv*, abs/1702.01992, 2017. URL https://api.semanticscholar.org/CorpusID:9401721.

Timothy Castiglia, Shiqiang Wang, and Stacy Patterson. Flexible vertical federated learning with heterogeneous parties. *IEEE Transactions on Neural Networks and Learning Systems*, 35:17878–17892, 2022a. URL https://api.semanticscholar.org/CorpusID:251881434.

Timothy Castiglia, Yi Zhou, Shiqiang Wang, Swanand Kadhe, Nathalie Baracaldo, and Stacy Patterson. Less-vfl: Communication-efficient feature selection for vertical federated learning. *ArXiv*, abs/2305.02219, 2023. URL https://api.semanticscholar.org/CorpusID:258461290.

Timothy J Castiglia, Anirban Das, Shiqiang Wang, and Stacy Patterson. Compressed-vfl: Communication-efficient learning with vertically partitioned data. In *International conference on machine learning*, pp. 2738–2766. PMLR, 2022b.

Santiago Castro, Devamanyu Hazarika, Verónica Pérez-Rosas, Roger Zimmermann, Rada Mihalcea, and Soujanya Poria. Towards multimodal sarcasm detection (an _obviously_ perfect paper). In *Annual Meeting of the Association for Computational Linguistics*, 2019. URL https://api.semanticscholar.org/CorpusID:174799301.

Hao Chen, Kim Laine, and Peter Rindal. Fast private set intersection from homomorphic encryption. *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017. URL https://api.semanticscholar.org/CorpusID:4448210.

Tianyi Chen, Xiao Jin, Yuejiao Sun, and Wotao Yin. Vafl: a method of vertical asynchronous federated learning. *arXiv preprint arXiv:2007.06081*, 2020.

Tat-Seng Chua, Jinhui Tang, Richang Hong, Haojie Li, Zhiping Luo, and Yantao Zheng. Nus-wide: a real-world web image database from national university of singapore. In *ACM International Conference on Image and Video Retrieval*, 2009. URL https://api.semanticscholar.org/CorpusID:6483070.

Cynthia Dwork. Differential privacy. *Encyclopedia of Cryptography and Security*, pp. 338–340, 2011.

Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam D. Smith. Calibrating noise to sensitivity in private data analysis. *J. Priv. Confidentiality*, 7:17–51, 2006. URL https://api.semanticscholar.org/CorpusID:2468323.

European Parliament and Council. Regulation (EU) 2016/679 of the european parliament and of the council of 27 april 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data (General Data Protection Regulation). Official Journal of the European Union, 2016. URL https://eur-lex.europa.eu/eli/reg/2016/679/oj.

Fangcheng Fu, Xupeng Miao, Jiawei Jiang, Huanran Xue, and Bin Cui. Towards communication-efficient vertical federated learning training via cache-enabled local updates. *arXiv preprint arXiv:2207.14628*, 2022.

Bin Gu, An Xu, Zhouyuan Huo, Cheng Deng, and Heng Huang. Privacy-preserving asynchronous vertical federated learning algorithms for multiparty collaborative learning. *IEEE Transactions on Neural Networks and Learning Systems*, 33:6103–6115, 2021. URL https://api.semanticscholar.org/CorpusID:235627816.

M. Hasan, Wasifur Rahman, Amir Zadeh, Jianyuan Zhong, Md. Iftekhar Tanveer, Louis philippe Morency, and Ehsan Hoque. Ur-funny: A multimodal language dataset for understanding humor. In *Conference on Empirical Methods in Natural Language Processing*, 2019. URL https://api.semanticscholar.org/CorpusID:119309472.

Yuanyuan He, Xin-Yu Tan, Jianbing Ni, Laurence Tianruo Yang, and Xianjun Deng. Differentially private set intersection for asymmetrical id alignment. *IEEE Transactions on Information Forensics and Security*, 17:3479–3494, 2022. URL https://api.semanticscholar.org/CorpusID:252396111.

Sepp Hochreiter and J. Urgen Schmidhuber. Long short-term memory. *Neural Computation*, 9: 1735–1780, 1997. URL https://api.semanticscholar.org/CorpusID:1915014.

Chung-Ju Huang, Leye Wang, and Xiao Han. Vertical federated knowledge transfer via representation distillation for healthcare collaboration networks. *Proceedings of the ACM Web Conference 2023*, 2023. URL https://api.semanticscholar.org/CorpusID:256826960.

Alistair E. W. Johnson, Tom J. Pollard, Lu Shen, Li wei H. Lehman, Mengling Feng, Mohammad M. Ghassemi, Benjamin Moody, Peter Szolovits, Leo Anthony Celi, and Roger G. Mark. Mimic-iii, a freely accessible critical care database. *Scientific Data*, 3, 2016. URL https://api.semanticscholar.org/CorpusID:33285731.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. URL https://api.semanticscholar.org/CorpusID:6628106.

Michelle A. Lee, Yuke Zhu, Peter Zachares, Matthew Tan, Krishna Parasuram Srinivasan, Silvio Savarese, Fei-Fei Li, Animesh Garg, and Jeannette Bohg. Making sense of vision and touch: Learning multimodal representations for contact-rich tasks. *IEEE Transactions on Robotics*, 36:582–596, 2019. URL https://api.semanticscholar.org/CorpusID:198985891.

Michelle A Lee, Brent Yi, Roberto Martín-Martín, Silvio Savarese, and Jeannette Bohg. Multimodal sensor fusion with differentiable filters. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 10444–10451. IEEE, 2020.

Mu Li, David G. Andersen, Jun Woo Park, Alex Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J. Shekita, and Bor-Yiing Su. Scaling distributed machine learning with the parameter server. In *USENIX Symposium on Operating Systems Design and Implementation*, 2014. URL https://api.semanticscholar.org/CorpusID:4614646.

Songze Li, Duanyi Yao, and Jin Liu. Fedvs: Straggler-resilient and privacy-preserving vertical federated learning for split models. *IACR Cryptol. ePrint Arch.*, 2023:597, 2023. URL https://api.semanticscholar.org/CorpusID:258332102.

Xiang Liu, Liangxi Liu, Feiyang Ye, Yunheng Shen, Xia Li, Linshan Jiang, and Jialin Li. Fedlpa: One-shot federated learning with layer-wise posterior aggregation. *Advances in Neural Information Processing Systems 37*, 2024.

Yang Liu, Tao Fan, Tianjian Chen, Qian Xu, and Qiang Yang. Fate: An industrial grade platform for collaborative learning with data protection. *Journal of machine learning research*, 22:226:1–226:6, 2021. URL https://api.semanticscholar.org/CorpusID:246432494.

Yang Liu, Yan Kang, Tianyuan Zou, Yanhong Pu, Yuanqin He, Xiaozhou Ye, Ye Ouyang, Yaqin Zhang, and Qian Yang. Vertical federated learning: Concepts, advances, and challenges. *IEEE Transactions on Knowledge and Data Engineering*, 36:3615–3634, 2022. URL https://api.semanticscholar.org/CorpusID:253801700.

Yi Liu, Yang Liu, Leqian Zheng, Jue Hong, Junjie Shi, Qingyou Yang, Ye Wu, and Cong Wang. Pubsub-vfl: Towards efficient two-party split learning in heterogeneous environments via publisher/subscriber architecture. *NeurIPS*, 2025.

H. B. McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera Y. Arcas. Communication-efficient learning of deep networks from decentralized data. In *International Conference on Artificial Intelligence and Statistics*, 2016. URL https://api.semanticscholar.org/CorpusID:14955348.

Philipp Moritz, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, William Paul, Michael I. Jordan, and Ion Stoica. Ray: A distributed framework for emerging ai applications. *ArXiv*, abs/1712.05889, 2017. URL https://api.semanticscholar.org/CorpusID:34552495.

PaddlePaddle. PaddleFL: A Federated Learning Framework Based on PaddlePaddle. https://github.com/PaddlePaddle/PaddleFL, 2026. Accessed: 2026-05-01.

Yuanzhe Peng, Zhuo Lu, and Jie Xu. Joint horizontal and vertical federated learning for multimodal iot. *Proceedings of the 30th Annual International Conference on Mobile Computing and Networking*, 2024. URL https://api.semanticscholar.org/CorpusID:274521501.

Tao Qi, Fangzhao Wu, Chuhan Wu, Lingjuan Lyu, Tongye Xu, Zhongliang Yang, Yongfeng Huang, and Xing Xie. Fairvfl: A fair vertical federated learning framework with contrastive adversarial learning. *Advances in Neural Information Processing Systems*, 2022. URL https://api.semanticscholar.org/CorpusID:249431495.

Mark Sandler, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 4510–4520, 2018. URL https://api.semanticscholar.org/CorpusID:4555207.

Wei Shen, Weiqi Liu, Mingde Chen, Wenke Huang, and Mang Ye. Mars-vfl: A unified benchmark for vertical federated learning with realistic evaluation. *Annual Conference on Neural Information Processing Systems*, 2025.

Niloy Sikder and Abdullah Al Nahid. Ku-har: An open dataset for heterogeneous human activity recognition. *Pattern Recognit. Lett.*, 146:46–54, 2021. URL https://api.semanticscholar.org/CorpusID:233706984.

Congzheng Song and Ananth Raghunathan. Information leakage in embedding models. *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020. URL https://api.semanticscholar.org/CorpusID:214743021.

Chang Sun, Lianne Ippel, Johan Van Soest, Birgit Wouters, Alexander Malic, Onaopepo Adekunle, Bob van den Berg, Ole Mussmann, Annemarie Koster, Carla van der Kallen, et al. A privacy-preserving infrastructure for analyzing personal health data in a vertically partitioned scenario. *MedInfo*, 264:373–377, 2019.

Pedro Valdeira, Shiqiang Wang, and Yuejie Chi. Vertical federated learning with missing features during training and inference. *ArXiv*, abs/2410.22564, 2024. URL https://api.semanticscholar.org/CorpusID:273696060.

Oriol Vinyals, Charles Blundell, Timothy P. Lillicrap, Koray Kavukcuoglu, and Daan Wierstra. Matching networks for one shot learning. In *Neural Information Processing Systems*, 2016. URL https://api.semanticscholar.org/CorpusID:8909022.

Patrick Wagner, Nils Strodthoff, Ralf Bousseljot, Dieter Kreiseler, Fatima I. Lunze, Wojciech Samek, and Tobias Schaeffter. Ptb-xl, a large publicly available electrocardiography dataset. *Scientific Data*, 7, 2020. URL https://api.semanticscholar.org/CorpusID:218865062.

Ganyu Wang, Bin Gu, Qingsong Zhang, Xiang Li, Boyu Wang, and Charles X. Ling. A unified solution for privacy and communication efficiency in vertical federated learning. *Advances in Neural Information Processing Systems*, 2023. URL https://api.semanticscholar.org/CorpusID:268030783.

Xiong Wang, Yi Zhang, Yuxin Chen, Yuqing Li, Chuanhu Ma, Bo Li, and Hai Jin. Runtime-aware pipeline for vertical federated learning with bounded model staleness. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 1*, pp. 1539–1550, 2025.

Yuncheng Wu and Nai-Li Xing. Falcon: A privacy-preserving and interpretable vertical federated learning system. *Proc. VLDB Endow.*, 16:2471–2484, 2023. URL https://api.semanticscholar.org/CorpusID:260085150.

Zhaomin Wu, Zheng Qin, Junyi Hou, Haodong Zhao, Qinbin Li, Bingsheng He, and Lixin Fan. Vertical federated learning in practice: The good, the bad, and the ugly. *ArXiv*, abs/2502.08160, 2025. URL https://api.semanticscholar.org/CorpusID:276287532.

Yidan Xing, Zhenzhe Zheng, and Fan Wu. Preventing strategic behaviors in collaborative inference for vertical federated learning. *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2024. URL https://api.semanticscholar.org/CorpusID:271961681.

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.

Amir Zadeh, Rowan Zellers, Eli Pincus, and Louis philippe Morency. Mosi: Multimodal corpus of sentiment intensity and subjectivity analysis in online opinion videos. *ArXiv*, abs/1606.06259, 2016. URL https://api.semanticscholar.org/CorpusID:13978043.

Amir Zadeh, Paul Pu Liang, Soujanya Poria, E. Cambria, and Louis philippe Morency. Multimodal language analysis in the wild: Cmu-mosei dataset and interpretable dynamic fusion graph. In *Annual Meeting of the Association for Computational Linguistics*, 2018. URL https://api.semanticscholar.org/CorpusID:51868869.

Ke Zhang, Ganyu Wang, Han Li, Yulong Wang, Hong Chen, and Bin Gu. Asynchronous vertical federated learning for kernelized auc maximization. *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2024. URL https://api.semanticscholar.org/CorpusID:271961695.

Qingsong Zhang, Bin Gu, Cheng Deng, Songxiang Gu, Liefeng Bo, Jian Pei, and Heng Huang. Asysqn: Faster vertical federated learning algorithms with better computation resource utilization. *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, 2021. URL https://api.semanticscholar.org/CorpusID:236980153.

Kaiwen Zhou and Xin Eric Wang. Fedvln: Privacy-preserving federated vision-and-language navigation. *ArXiv*, abs/2203.14936, 2022.

# A MATH NOTATION

Table 4 summarizes the math notation of this paper.

Table 4: Mathematical notation.

| Symbol | Meaning and context |
|---|---|
| **Participants, workers, data, and models** | |
| $P_k$ | Passive party holding the $k$th feature partition. |
| $K$, $\mathcal{K}$ | Number of passive parties and their index set, $\mathcal{K} = \{1, \ldots, K\}$. The active party is not included in $K$. |
| $k$ | Passive-party index, $k \in \mathcal{K}$. |
| $S$ | Logical parameter-server service coordinating the top model and maintaining separate aggregation state for each model group. |
| $a$ | Model-group index: $a = 0$ for the top model and $a = k$ for passive party $P_k$'s bottom model. Also reused as a temporary operand in a proof inequality. |
| $W_k$, $W_0$ | Numbers of bottom-model workers at party $k$ and top-model workers at the active party, respectively. |
| $w$, $v$ | Bottom-worker and top-worker indices, respectively. In unified group notation, $w$ can also index a top-model worker. |
| $N$, $i$ | Number of aligned training examples and training-example index. In the proof, $i$ is also a summation index for intermediate parameter changes. |
| $x_{k,i}$, $y_i$ | Features of example $i$ held by party $k$, and the target of that example. |
| $B$ | Configured training batch size. |
| $M$ | Number of batches in one dataset pass: $M = \lceil N/B \rceil$. |
| $b$ | Shared training-batch identifier. In the MSE formula, it instead indexes test sliding windows; in a generic proof inequality it is also a temporary operand. |
| $\mathcal{I}_b$ | Ordered list of example indices for batch $b$, shared across parties. |
| $B_b$ | Actual size of batch $b$: $B_b = \lvert\mathcal{I}_b\rvert \leq B$. |
| $X_{k,b}$ | Features of batch $b$ held by party $k$, in the order prescribed by $\mathcal{I}_b$. |
| $(k, b)$, $KM$ | Party–batch routing key, and number of embedding/gradient channel pairs in one dataset pass, respectively. |
| $h_k$, $f_0$ | Bottom model at passive party $k$, and active-party top model combining all parties' embeddings. |
| $\theta_k$, $\theta_0$ | Parameters of bottom model $h_k$ and top model $f_0$. |
| $\Theta$ | Collection of all model parameters: $(\theta_0, \theta_1, \ldots, \theta_K)$. |
| $\theta_{k,w}$, $\theta_{0,v}$ | Private bottom-model and top-model parameter replicas held by workers $w$ and $v$. |
| $\theta_{k,w}^{b,j}$, $\theta_{0,v}^{b,j}$ | Worker parameters at local step $j$ for batch $b$. Superscripts $b, 0$ and $b, Q$ denote the initial and final states of that batch's local-update block. |
| $\ell$ | Per-example prediction loss. |
| $\mathcal{F}(\Theta)$ | Mean training objective over all $N$ aligned examples. |
| $\mathcal{L}_b$ | Mean loss of batch $b$, normalized by its actual size $B_b$. |
| $Z_{k,b}$ | Embedding of party $k$ for batch $b$: $h_k(X_{k,b}; \theta_k)$, or the corresponding worker-replica evaluation. |

Continued on the next page

Table 4 (continued)

| Symbol | Meaning and context |
|---|---|
| $G_{k,b}$ | Gradient of mean batch loss with respect to the embedding: $\partial \mathcal{L}_b / \partial Z_{k,b}$. |
| $\mathcal{C}_{k,b}^{\text{emb}}, \mathcal{C}_{k,b}^{\text{grad}}$ | Logical channels carrying the embedding and its returned gradient for party $k$ and batch $b$. |
| **Local updates, synchronization, and model-state freshness** | |
| $\eta$ | Fixed learning rate for each local parameter update. In the convergence result, $0 < \eta \leq 1/(4LQ)$. |
| $Q$ | Number of local updates per exchanged batch or accepted update block; a positive integer. |
| $j$ | Local-step index, usually $j = 0, \ldots, Q-1$. In the accuracy formula, $j$ instead indexes evaluation examples. |
| $Q_{\min}, Q^*$ | Smallest positive integer satisfying all strict timing constraints. The main text also denotes this selected optimum by $Q^*$. Optimality is with respect to the stated timing constraint. |
| $e, T$ | Current training epoch and total epoch count in the workflow algorithm. |
| $E, E_e$ | Worker synchronization interval and its value at epoch $e$. |
| $E_0$ | Positive-integer parameter controlling the interval schedule. |
| $D$ | Positive-integer upper bound on staleness at each passive worker. |
| $d_{k,w}$ | Staleness of worker $(k, w)$, defined as the number of embeddings sent minus the number of corresponding batches for which all $Q$ local updates have been completed. |
| $C_a, C_a/M$ | The number of completed batch blocks counted for model group $a$, and its epoch-equivalent completed work. |
| **Computation and communication timing** | |
| $t^{sc}, t_k^{pc}$ | Time per top-model update and per bottom-model update at party $k$. |
| $t_k^e, t_k^{ec}$ | Embedding-generation time and one-way transmission time for party $k$. Symmetric links give round-trip time $2t_k^{ec}$. |
| $c_k$ | Non-update time on party $k$'s modeled path: $c_k = t_k^e + 2t_k^{ec} \geq 0$. |
| $\gamma_k$ | Per-update timing difference: $\gamma_k = t^{sc} - t_k^{pc}$. The finite formula for $Q_{\min}$ requires every $\gamma_k > 0$. |
| $R_k(Q)$ | Uncovered time in the modeled overlap window: $\max\{0, c_k - Q\gamma_k\}$. |
| $R$ (timing appendix) | Largest communication-to-computation-gap ratio: $R = \max_{k \in \mathcal{K}} c_k/\gamma_k$. This is distinct from the block count in the convergence analysis. |
| $m$ (timing appendix) | Integer part of the timing ratio: $m = \lfloor R \rfloor$, giving $Q_{\min} = m + 1$. |
| $\mathcal{P}, (k, w, v)$ | Set of profiled passive/top-worker path combinations, and an individual party–bottom-worker–top-worker combination. |
| $t_v^{sc}, t_{k,w}^{pc}$ | Top-update time for worker $v$ and bottom-update time for worker $w$ at party $k$. |
| $t_{k,w}^e$ | Embedding-generation time for passive worker $(k, w)$. |
| $t_k^{\text{emb-up}}, t_k^{\text{grad-down}}$ | Party-level directional embedding-upload and gradient-download transmission times for asymmetric links. |
| $t_{k,w}^{\text{emb-up}}, t_{k,w}^{\text{grad-down}}$ | Worker-specific directional transmission times used in the multi-worker expression for $Q_{\min}$. |
| **Privacy notation** | |

Continued on the next page

Table 4 (continued)

| Symbol | Meaning and context |
|---|---|
| $\epsilon$, $\delta$ | Privacy parameters in the $(\epsilon, \delta)$-differential-privacy guarantee. |
| $\mathcal{M}$ | Randomized mechanism in the privacy definition. |
| $D$, $D'$ (privacy) | Adjacent datasets differing in one private training record; distinct from the worker staleness bound $D$. |
| $O$ | Measurable output event in the inequality $\Pr[\mathcal{M}(D) \in O] \leq e^{\epsilon} \Pr[\mathcal{M}(D') \in O] + \delta$. |
| $\Pr[\cdot]$ | Probability of an event. |
| $q$ | Embedding query whose output is perturbed before release. |
| $\Delta_2$ | $L_2$ sensitivity of $q$: $\sup_{D \sim D'} \|q(D) - q(D')\|_2$. |
| $\epsilon_0$, $\delta_0$ | Per-release differential privacy parameters. |
| $c$ (privacy calibration) | Positive calibration constant satisfying $c^2 > 2\log(1.25/\delta_0)$; distinct from the parameter context $c$. |
| $R_{\mathrm{rel}}$ | Upper bound on the number of accounted releases, including new noisy releases caused by retries. |
| $\widetilde{Z}$, $\xi$ | Released noisy embedding and additive Gaussian noise, with $\widetilde{Z} = q(D) + \xi$. |
| $\sigma_{\mathrm{dp}}$, $\sigma_{\mathrm{dp}}^2$ | Standard deviation and variance of Gaussian noise per perturbed embedding coordinate. |
| $p$ | Total number of noise coordinates in one block, each counted once even if reused in later local steps. |
| $\zeta_r$ | Vector of embedding noise in block $r$, with $\zeta_r \in \mathbb{R}^p$ and $\mathbb{E}\|\zeta_r\|^2 \leq p\sigma_{\mathrm{dp}}^2$. |
| $z$ | Generic embedding-noise argument of the update map $U_b(c, z)$; $z = 0$ means no privacy noise. |
| $\kappa$ | Bound on update-direction sensitivity to embedding perturbations: $\|U_b(c, z) - U_b(c, 0)\| \leq \kappa\|z\|$. |
| **Convergence assumptions and reference quantities** | |
| $R$, $r$ (convergence) | Total number of accepted update blocks and block index $r = 0, \ldots, R-1$. Each block comprises $Q$ local updates. |
| $\Theta_r$, $\Theta_0$, $\Theta_R$ | Reference model parameters at the start of block $r$, at initialization, and after $R$ blocks. $\Theta_{r+1}$ denotes the next block's reference parameters. |
| $\Theta^\star$ | A parameter collection minimizing $\mathcal{F}$. |
| $\mathcal{F}^\star$ | Infimum objective value: $F^\star = \inf_\Theta F(\Theta) > -\infty$. |
| $\mu$ | Positive PL constant satisfying $\|\nabla\mathcal{F}(\Theta)\|^2 \geq 2\mu[\mathcal{F}(\Theta) - \mathcal{F}^\star]$, with $0 < \mu \leq L$. |
| $L$ | Common upper bound for two separately assumed Lipschitz constants: objective-gradient changes and noiseless update-map changes with parameter context. |
| $x$, $y$ (loss assumptions) | Two generic settings of all model parameters; not features and labels in this context. |
| $\sigma_g^2$ | Upper bound on the conditional variance of the fresh reference batch gradient. |
| $G$, $G^2$ | Nonnegative constant controlling parameter-change magnitudes relative to $\eta$, and its square. Distinct from the embedding gradient $G_{k,b}$. |
| $\tau$ | Nonnegative bound on initial staleness and replica disagreement, expressed in equivalent local-update steps. |

Continued on the next page

Table 4 (continued)

| Symbol | Meaning and context |
|---|---|
| $\mathcal{B}_r$ | Batch selected for block $r$. |
| $\mathscr{H}_r$ | Training history available before selecting $\mathcal{B}_r$; used to condition expectations. |
| $\mathcal{L}_{\mathcal{B}_r}$ | Mean loss of the batch selected for block $r$. |
| $g_r$ | Fresh, noiseless reference batch gradient: $\nabla\mathcal{L}_{\mathcal{B}_r}(\Theta_r)$. It is an analytical comparison quantity. |
| $u_{r,j}$ | Actual joint update direction at step $j$ in block $r$, including worker-combination weights. |
| $c_{r,j}$ | Collection of all parameter values actually used to calculate $u_{r,j}$, including older values used by cached computations. |
| $c_r^\circ$ | Reference computation context using $\Theta_r$ consistently for all computations, embeddings, and returned gradients. |
| $c$, $c'$ (update map) | Generic parameter contexts compared in the update-map Lipschitz assumption. |
| $U_b(c, z)$ | Update-direction map for batch $b$, parameter context $c$, and embedding noise $z$. In particular, $U_{\mathcal{B}_r}(c_r^\circ, 0) = g_r$. |
| **Auxiliary symbols used in the convergence proof** | |
| $m$ (displacement proof) | Number of intermediate parameter changes in $c_{r,j} - c_r^\circ = \sum_{i=1}^{m} v_i$, with $m \le \tau + j$. Distinct from $\lfloor R \rfloor$ in the timing proof. |
| $v_i$ | One intermediate parameter-change vector, satisfying $\mathbb{E}\|v_i\|^2 \le \eta^2 G^2$. Distinct from top-worker index $v$. |
| $a$, $b$ (generic inequality) | Temporary operands in $\|a+b\|^2 \le 2\|a\|^2 + 2\|b\|^2$; not model-group or batch indices there. |
| $\gamma$ | Block-level step-size factor $\eta Q$; distinct from the timing difference $\gamma_k$. |
| $e_r$ | Mean update-direction error within a block: $Q^{-1}\sum_{j=0}^{Q-1}(u_{r,j} - g_r)$. Distinct from epoch index $e$. |
| $B_Q$ | Bound on $\mathbb{E}\|e_r\|^2$, defined as $2L^2\eta^2G^2(\tau+Q-1)^2 + 2\kappa^2 p\sigma_{\mathrm{dp}}^2$. Distinct from batch sizes $B$ and $B_b$. |
| $a_r$ | Full objective gradient at the block reference state: $\nabla\mathcal{F}(\Theta_r)$. |
| $\Delta_r$, $\Delta_0$, $\Delta_R$ | Expected objective gap $\Delta_r = \mathbb{E}[\mathcal{F}(\Theta_r) - \mathcal{F}^\star]$, initially and after $R$ blocks. $\Delta_{r+1}$ is the next gap in the recurrence. |
| $\rho$ | Contraction factor $1 - \mu\gamma$, with $3/4 \le \rho < 1$ under the stated step-size assumptions. |
| **Evaluation metrics** | |
| $N_{\mathrm{eval}}$ | Number of classification evaluation samples. |
| $\hat{y}_j$, $y_j$ | Predicted and true class for evaluation sample $j$. The hat marks a prediction. |
| $\mathbf{1}[\cdot]$ | Indicator function: one if the enclosed condition is true, and zero otherwise. |
| $N_{\mathrm{test}}$ | Number of MuJoCo test sliding windows. |
| $b$, $t$, $c$ (MSE) | Test-window index, within-window time-step index $t = 1, \ldots, 16$, and coordinate-component index $c \in \{x, y\}$. |
| $x$, $y$ (coordinates) | Names of the two position-coordinate components in the MSE formula; not the generic parameter vectors used in the convergence assumptions. |

Continued on the next page

Table 4 (continued)

| Symbol | Meaning and context |
|---|---|
| $\hat{y}_{b,t,c}$, $y_{b,t,c}$ | Predicted and true scalar position for window $b$, time step $t$, and coordinate $c$. |
| $d$ | Task index in the training-time comparison; distinct from worker staleness counters. |
| $T_{\mathrm{Our},d}$ | OrbiVFL wall-clock training time on task $d$, measured in seconds. |
| $T^{\min}_{\mathrm{base},d}$ | Shortest training time among the four competing baselines on task $d$. |
| $\mathrm{Reduction}_d$ | Percentage time reduction: $(1 - T_{\mathrm{Our},d}/T^{\min}_{\mathrm{base},d}) \times 100\%$. |
| **Mathematical operators and notation conventions** | |
| $\mathbb{N}_+$, $\mathbb{R}^p$ | Positive integers and the space of real vectors with $p$ coordinates. |
| $\mathbb{E}[\cdot]$, $\mathbb{E}[\cdot \mid \mathscr{H}_r]$ | Expectation and expectation conditional on the training history. |
| $\nabla$, $\nabla_{\theta_0}$, $\partial$ | Gradient, gradient with respect to top-model parameters, and partial derivative. |
| $\langle\cdot,\cdot\rangle$, $\|\cdot\|$ | Euclidean inner product and Euclidean norm, respectively. |
| $\lvert\mathcal{I}_b\rvert$, $\lvert\cdot\rvert$ | Cardinality of the batch-index list and absolute value of a scalar, respectively. |
| $\lfloor\cdot\rfloor$, $\lceil\cdot\rceil$ | Floor (round down to an integer) and ceiling (round up to an integer). |
| $\sum$ | Summation function. |
| $e^{\epsilon}$ | Exponential of the privacy parameter. |
| $\in$, $\{\cdot\}$, $\ldots$ | Set membership, set notation, and continuation of an indexed sequence. |
| $(\cdot)^2$, $(\cdot)^{-1}$, $(\cdot)^R$ | Square, reciprocal, and power. Superscripts such as $\mathrm{emb}$ and $\mathrm{src}$ are descriptive tags rather than powers. |

## B MESSAGE IDENTITY AND MODEL-STATE FRESHNESS

**Party, worker, and batch identities.** The active party hosts $S$ and the top-model workers; $K$ counts the passive feature-holding parties. A party is not assumed to be a single physical device. Workers are Ray execution actors with private model replicas. We use $a = 0$ for the top-model group and $a = k$ for party $P_k$'s bottom-model group. The symbol $w$ can also index top-model replicas when a unified group notation is needed. One logical service $S$ performs top-model coordination and maintains separate aggregation state for every group; these functions need not execute in one serial actor. Only replicas of the same model are combined.

A batch identifier refers to an ordered index list, including the order within the batch. With $M = \lceil N/B \rceil$, the final batch uses $B_b \leq B$, and its loss is divided by $B_b$. Thus $\partial \mathcal{L}_b / \partial Z_{k,b}$ already includes the mean-loss normalization. For example, $N = 1000$ and $B = 64$ yield 16 batches, whose last batch has 40 examples, regardless of the number of workers. There are $KM$ party–batch embedding/gradient channel pairs for a single pass. These are logical routing keys that can share a finite set of broker actors and transport connections.

**Ray implementation boundary.** The broker maintains bounded buffers and a registry mapping a run, party, and Batch ID to the producing worker. Ray actors manage mutable ownership and synchronization state; object references can transport embedding and gradient payloads. A pending read releases the communication actor to serve other requests. A training actor must not occupy the same event loop with long synchronous computation that prevents message handling. Ray is the execution substrate selected here, while our application layer implements matching, ownership, deduplication, and freshness checks.

**Matching and ownership.** Each message carries its run identifier, $k$, $b$, an attempt token, its producer identity, and model-state tags. The routing key $(k, b)$ preserves the notation of the main text; the additional fields disambiguate retries and state changes. A top worker claims a batch only after all $K$ required embedding slots are present, have the same ordered examples, and pass validation. Each returned gradient is routed to the worker and retained input context that produced its embedding. Another worker cannot consume it merely because it has the same model architecture. The registry records consumed messages so a duplicate delivery cannot apply the same update twice. A bounded buffer uses backpressure when full; its capacity is separate from $D$ and by default is 15.

**Staleness and its bound.** The staleness $d_{k,w}$ of passive worker $(k, w)$ is defined as the number of embeddings sent minus the number of corresponding batches for which all $Q$ local updates have been completed. Initially, $d_{k,w} = 0$. Sending an embedding increases $d_{k,w}$ by one, while completing all $Q$ corresponding local updates decreases it by one. Receiving a gradient alone does not change $d_{k,w}$. A worker generates new embeddings only when $d_{k,w} < D$. Otherwise, it pauses embedding generation while continuing to process returned gradients and complete pending updates. The eligibility check and increment of $d_{k,w}$ are serialized for each worker. With $D = 1$, each worker completes all $Q$ local updates for its current batch before generating the next embedding. Larger values permit multiple batches to remain pending.

**Refresh, progress, and stopping.** An invalid or abandoned batch attempt receives a fresh run-unique Batch ID on retry, with its original example list retained. Its obsolete messages are marked closed, and the registry creates at most one replacement attempt. Once a worker has checked the required messages for a batch and started its $Q$ local updates, it completes all $Q$ updates before stopping; closure does not interrupt these updates or undo those already completed. A retry does not undo model updates that were already committed; this possible imbalance is part of the accepted-update selection analyzed below. Each group counts a Batch ID once only after its complete $Q$-step. Counts therefore represent completed batch work, not the number of channel deliveries or the number of primitive updates. Under retries, $C_a/M$ measures epoch-equivalent work, rather than an assertion that every example has appeared exactly once. The application should log retries separately. At a finite training stop, workers complete the remaining local updates for each batch whose required messages have passed the checks and whose $Q$ local updates have already started, then initiate a final deadline-bounded group collection. Remaining unstarted work is closed, timely snapshots are aggregated, and the latest $K$ bottom aggregates and top aggregate are exported.

## C Derivation and Scope of the Local Update Count

**The modeled overlap window.** For each profiled party path, define $c_k = t_k^e + 2t_k^{ec} \geq 0$ and $\gamma_k = t^{sc} - t_k^{pc}$. The intended steady-state schedule exposes a top-computation window of length $Qt^{sc}$ during which the party-side path performs $Q$ bottom updates, generates the next embedding, and completes one embedding/gradient round trip. In this idealized window, the uncovered duration is

$$R_k(Q) = \max\{0,\, c_k - Q\gamma_k\}. \tag{5}$$

The requested strict condition $Q\gamma_k > c_k$ makes this modeled residual zero with positive slack. It is conditional on that overlap schedule being available. In particular, the window starts after an embedding gradient has been released.

**Smallest feasible integer.** If every $\gamma_k > 0$, all party inequalities hold exactly when

$$Q > R, \qquad R = \max_{k \in \mathcal{K}} \frac{c_k}{\gamma_k}.$$

Write $m = \lfloor R \rfloor$. Since $m \leq R < m + 1$, $Q = m + 1$ is feasible, whereas every positive integer $Q \leq m$ is infeasible for at least one maximizing path. Hence

$$Q_{\min} = \lfloor R \rfloor + 1 = \operatorname*{argmin}_{Q \in \mathbb{N}_+} \{Q : Q\gamma_k > c_k \text{ for every } k\}.$$

For an integer ratio $R = 4$, this gives $Q_{\min} = 5$, not 4. The original strict condition can therefore be written as the integer requirement $Q \geq Q_{\min}$. If the design instead used a non-strict timing inequality, its answer would be $\max\{1, \lceil R \rceil\}$.

**Why the smallest value is preferred.** All $Q \geq Q_{\min}$ eliminate the modeled residual, so further increases cannot make that residual less than zero. Under the fixed per-step timing model, they increase computation per exchanged batch and the duration of reuse of fixed embeddings or gradients. Selecting $Q_{\min}$ therefore minimizes local work subject to the stated coverage constraint. The convergence analysis shows the similar results.

**Multiple workers and asymmetric links.** Timings must be measured at the deployed batch size and concurrency, rather than dividing single-worker time by $W_k$. Let $\mathcal{P}$ contain the profiled combinations $(k, w, v)$ of a passive worker and a top worker, and let $t_k^{\text{emb-up}}$ and $t_k^{\text{grad-down}}$ denote the two directional transmission times when they differ. The extension is

$$Q_{\min} = \left\lfloor \max_{(k,w,v) \in \mathcal{P}} \frac{t_{k,w}^e + t_{k,w}^{\text{emb-up}} + t_{k,w}^{\text{grad-down}}}{t_v^{sc} - t_{k,w}^{pc}} \right\rfloor + 1. \tag{6}$$

# D CONVERGENCE ANALYSIS

**What the bound measures.** Let $\Theta$ collect the top- and bottom-model parameters and let $\mathcal{F}(\Theta)$ be the training objective. Define its best possible value by

$$\mathcal{F}^\star = \inf_{\Theta} \mathcal{F}(\Theta) > -\infty.$$

This value refers to the training objective, rather than the best test score observed in an experiment. We analyze $R \geq 1$ groups of updates, each containing $Q \geq 1$ local updates. The indices are $r = 0, \ldots, R-1$ for groups and $j = 0, \ldots, Q-1$ for local updates. Let $\Theta_r$ be the reference parameters at the start of group $r$, with fixed initial parameters $\Theta_0$. Our goal is to bound $\mathbb{E}[\mathcal{F}(\Theta_R) - \mathcal{F}^\star]$, where $\mathbb{E}$ averages over the randomness in batch selection, noise, and the analyzed execution.

**Loss conditions.** For parameter vectors $x$ and $y$, assume

$$\|\nabla\mathcal{F}(x) - \nabla\mathcal{F}(y)\| \leq L\|x - y\|, \qquad L > 0. \tag{7}$$

Here $\nabla$ denotes the gradient and $\|\cdot\|$ the Euclidean norm. This condition limits how quickly the gradient can change when the parameters change. It gives the following upper bound on the loss:

$$\mathcal{F}(y) \leq \mathcal{F}(x) + \langle\nabla\mathcal{F}(x), y - x\rangle + \frac{L}{2}\|y - x\|^2, \tag{8}$$

where $\langle\cdot,\cdot\rangle$ is the inner product. The last term accounts for the difference between the actual loss change and its first-order prediction.

We replace strong convexity with the Polyak–Lojasiewicz (PL) condition:

$$\frac{1}{2}\|\nabla\mathcal{F}(\Theta)\|^2 \geq \mu[\mathcal{F}(\Theta) - \mathcal{F}^\star], \qquad 0 < \mu \leq L. \tag{9}$$

The constant $\mu$ relates the gradient size to the loss gap. In plain terms, when the loss is still far above its best possible value, the gradient must remain large enough to indicate progress. This assumption allows a nonconvex objective; it does not require strong convexity. We assume it holds at all reference parameters covered by the analysis.

**The reference update.** Let $\eta > 0$ be the fixed local learning rate and let $u_{r,j}$ be the joint update direction at step $j$ of group $r$. As in the original update model, worker-combination weights are included in $u_{r,j}$. We assume

$$\Theta_{r+1} = \Theta_r - \eta \sum_{j=0}^{Q-1} u_{r,j}. \tag{10}$$

Thus, one group's parameter change is the sum of its local update directions multiplied by the learning rate.

**Batch sampling.** Let $\mathcal{B}_r$ be the batch accepted for group $r$ and let $\mathscr{H}_r$ contain the training history before its selection. For a generic batch index $b$, $\mathcal{L}_b$ is its mean loss. Define the fresh, noiseless reference batch gradient by

$$g_r = \nabla\mathcal{L}_{\mathcal{B}_r}(\Theta_r).$$

This gradient is a comparison quantity: it is recomputed at $\Theta_r$ without using old parameter values or embedding noise. Assume

$$\begin{aligned} \mathbb{E}[g_r \mid \mathscr{H}_r] &= \nabla\mathcal{F}(\Theta_r), \\ \mathbb{E}[\|g_r - \nabla\mathcal{F}(\Theta_r)\|^2 \mid \mathscr{H}_r] &\leq \sigma_g^2. \end{aligned} \tag{11}$$

The first condition says that the batch gradient has the correct direction on average. The second bounds its random variation by $\sigma_g^2$. Both conditions concern batches accepted after scheduling.

**Parameter differences and embedding noise.** Let $c_{r,j}$ collect all parameter values used to compute $u_{r,j}$. Let $c_r^\circ$ use $\Theta_r$ consistently in all corresponding computations, including embeddings and returned gradients. For parameter contexts $c, c'$ and a noise vector $z$, let $U_b(c, z)$ denote the update direction for batch $b$. Let $\zeta_r$ collect the embedding noise used in group $r$. Assume

$$u_{r,j} = U_{\mathcal{B}_r}(c_{r,j}, \zeta_r), \qquad U_{\mathcal{B}_r}(c_r^\circ, 0) = g_r, \tag{12}$$

$$\|U_b(c,0) - U_b(c',0)\| \le L\|c - c'\|, \qquad \|U_b(c,z) - U_b(c,0)\| \le \kappa\|z\|. \tag{13}$$

Here $\kappa \ge 0$ bounds the effect of embedding noise on the update. The parameter-context bound is assumed separately from objective smoothness; $L$ is chosen large enough for both bounds.

We retain the full-context displacement assumption

$$\mathbb{E}\|c_{r,j} - c_r^\circ\|^2 \le \eta^2 G^2 (\tau + j)^2, \tag{14}$$

where $G \ge 0$ controls parameter-change magnitudes relative to $\eta$. The constant $\tau \ge 0$ bounds the parameter differences already present when a group starts, including stale parameters, disagreement between worker replicas, and aggregation-related differences. The index $j$ accounts for further changes during the group. The same constants must cover every group in the theorem.

Let $p$ be the number of perturbed embedding coordinates and let $\sigma_{\mathrm{dp}}$ be the Gaussian noise standard deviation per coordinate. Each coordinate is counted once, even if reused during later local updates. We assume

$$\mathbb{E}\|\zeta_r\|^2 \le p\sigma_{\mathrm{dp}}^2. \tag{15}$$

Independent zero-mean Gaussian coordinates with this standard deviation give equality. The proof only needs the displayed bound.

**Theorem 1.** *Suppose the loss, reference-update, sampling, update-map, parameter-displacement, and noise conditions above hold. For fixed $\Theta_0$, a constant step size $0 < \eta \le 1/(4LQ)$, and any integer $R \ge 1$ covered by these conditions,*

$$\begin{aligned}\mathbb{E}[\mathcal{F}(\Theta_R) - \mathcal{F}^\star] \le (1 - \mu\eta Q)^R [\mathcal{F}(\Theta_0) - \mathcal{F}^\star] + \frac{L\eta Q \sigma_g^2}{\mu} \\ + \frac{5}{2\mu}\left[L^2\eta^2 G^2 (\tau + Q - 1)^2 + \kappa^2 p \sigma_{\mathrm{dp}}^2\right].\end{aligned} \tag{16}$$

*Proof.* **Step 1: separate the effects of parameter differences and noise.** Add and subtract the noiseless update at the actual parameter context:

$$\begin{aligned}u_{r,j} - g_r = U_{\mathcal{B}_r}(c_{r,j}, \zeta_r) - U_{\mathcal{B}_r}(c_{r,j}, 0) \\ + U_{\mathcal{B}_r}(c_{r,j}, 0) - U_{\mathcal{B}_r}(c_r^\circ, 0).\end{aligned}$$

The first difference is due to noise; the second is due to parameter differences. For any vectors $x, y$, $\|x+y\|^2 \le 2\|x\|^2 + 2\|y\|^2$. This follows by expanding the square and using $2\langle x, y\rangle \le \|x\|^2 + \|y\|^2$. Applying the update-map bounds gives

$$\begin{aligned}\mathbb{E}\|u_{r,j} - g_r\|^2 &\le 2L^2\mathbb{E}\|c_{r,j} - c_r^\circ\|^2 + 2\kappa^2\mathbb{E}\|\zeta_r\|^2 \\ &\le 2L^2\eta^2 G^2(\tau + j)^2 + 2\kappa^2 p\sigma_{\mathrm{dp}}^2.\end{aligned} \tag{17}$$

No independence between the two error sources is needed for this step.

**Step 2: combine the $Q$ local updates.** Define the group step-size coefficient and average error by

$$\gamma = \eta Q, \qquad e_r = \frac{1}{Q}\sum_{j=0}^{Q-1}(u_{r,j} - g_r).$$

Then $\sum_{j=0}^{Q-1} u_{r,j} = Q(g_r + e_r)$, so the update is exactly

$$\Theta_{r+1} = \Theta_r - \gamma(g_r + e_r).$$

The squared norm of an average is at most the average squared norm. Using this inequality and $j \le Q - 1$ gives

$$\begin{aligned}\mathbb{E}\|e_r\|^2 &\le \frac{1}{Q}\sum_{j=0}^{Q-1}\mathbb{E}\|u_{r,j} - g_r\|^2 \\ &\le 2L^2\eta^2 G^2(\tau + Q - 1)^2 + 2\kappa^2 p\sigma_{\mathrm{dp}}^2 =: B_Q.\end{aligned} \tag{18}$$

The symbol $B_Q$ denotes this error bound. The same noisy embedding can be reused across local updates; such reuse does not create independent noise samples, so the noise term is not divided by $Q$.

**Step 3: bound the loss after one group.** Let $a_r = \nabla \mathcal{F}(\Theta_r)$. The sampling assumptions imply

$$\mathbb{E}[\langle a_r, g_r \rangle \mid \mathscr{H}_r] = \|a_r\|^2.$$

They also give

$$\begin{aligned}\mathbb{E}[\|g_r\|^2 \mid \mathscr{H}_r] &= \|a_r\|^2 + 2\langle a_r, \mathbb{E}[g_r - a_r \mid \mathscr{H}_r]\rangle + \mathbb{E}[\|g_r - a_r\|^2 \mid \mathscr{H}_r] \\ &\le \|a_r\|^2 + \sigma_g^2.\end{aligned}$$

The middle term is zero because the conditional mean of $g_r - a_r$ is zero.

Substituting the group update into Eq. 8 yields

$$\mathcal{F}(\Theta_{r+1}) \le \mathcal{F}(\Theta_r) - \gamma\langle a_r, g_r + e_r\rangle + \frac{L\gamma^2}{2}\|g_r + e_r\|^2.$$

To control the error terms, expand a nonnegative square:

$$0 \le \left\| \frac{1}{2}a_r + e_r \right\|^2 = \frac{1}{4}\|a_r\|^2 + \langle a_r, e_r\rangle + \|e_r\|^2.$$

Therefore,

$$-\langle a_r, e_r\rangle \le \frac{1}{4}\|a_r\|^2 + \|e_r\|^2, \qquad \|g_r + e_r\|^2 \le 2\|g_r\|^2 + 2\|e_r\|^2.$$

Taking expectations and collecting terms gives

$$\begin{aligned}\mathbb{E}\mathcal{F}(\Theta_{r+1}) \le \mathbb{E}\mathcal{F}(\Theta_r) &- \gamma(3/4 - L\gamma)\mathbb{E}\|a_r\|^2 + L\gamma^2\sigma_g^2 \\ &+ (\gamma + L\gamma^2)\mathbb{E}\|e_r\|^2.\end{aligned} \tag{19}$$

The coefficient $3/4$ comes from $1 - 1/4$: part of the ideal descent is used to control the error inner product.

Since $\eta \le 1/(4LQ)$, we have $L\gamma \le 1/4$ and hence

$$3/4 - L\gamma \ge 1/2, \qquad \gamma + L\gamma^2 \le \frac{5}{4}\gamma.$$

Using $\mathbb{E}\|e_r\|^2 \le B_Q$ gives

$$\begin{aligned}\mathbb{E}\mathcal{F}(\Theta_{r+1}) \le \mathbb{E}\mathcal{F}(\Theta_r) &- \frac{\gamma}{2}\mathbb{E}\|a_r\|^2 \\ &+ L\gamma^2\sigma_g^2 + \frac{5\gamma}{4}B_Q.\end{aligned} \tag{20}$$

This is the current loss minus a descent term, plus the costs of batch variation and update error.

**Step 4: turn the gradient term into a loss-gap term.** Define

$$\Delta_r = \mathbb{E}[\mathcal{F}(\Theta_r) - \mathcal{F}^\star].$$

The PL condition gives $\mathbb{E}\|a_r\|^2 \ge 2\mu\Delta_r$. Multiplication by $-\gamma/2$ reverses the inequality, so

$$-\frac{\gamma}{2}\mathbb{E}\|a_r\|^2 \le -\mu\gamma\Delta_r.$$

Subtracting $\mathcal{F}^\star$ from Eq. 20 therefore yields

$$\Delta_{r+1} \le (1 - \mu\gamma)\Delta_r + L\gamma^2\sigma_g^2 + \frac{5\gamma}{4}B_Q. \tag{21}$$

This is the step that uses the PL condition. The preceding error and descent calculations do not require strong convexity.

**Step 5: apply the recurrence over all groups.** Define $\rho = 1 - \mu\gamma$. Because $0 < \mu \leq L$ and $0 < \gamma \leq 1/(4L)$, we have $3/4 \leq \rho < 1$. For example, the first two applications give

$$\Delta_1 \leq \rho\Delta_0 + L\gamma^2\sigma_g^2 + \frac{5\gamma}{4}B_Q,$$
$$\Delta_2 \leq \rho^2\Delta_0 + (1+\rho)\left(L\gamma^2\sigma_g^2 + \frac{5\gamma}{4}B_Q\right).$$

Here $\Delta_1$ and $\Delta_2$ are the previously defined loss gap at indices 1 and 2. Repeating the substitution gives

$$\begin{aligned}\Delta_R &\leq \rho^R\Delta_0 + \left(L\gamma^2\sigma_g^2 + \frac{5\gamma}{4}B_Q\right)\sum_{r=0}^{R-1}\rho^r \\ &= \rho^R\Delta_0 + \left(L\gamma^2\sigma_g^2 + \frac{5\gamma}{4}B_Q\right)\frac{1-\rho^R}{\mu\gamma} \\ &= \rho^R\Delta_0 + (1-\rho^R)\left(\frac{L\gamma\sigma_g^2}{\mu} + \frac{5B_Q}{4\mu}\right) \\ &\leq \rho^R\Delta_0 + \frac{L\gamma\sigma_g^2}{\mu} + \frac{5B_Q}{4\mu}.\end{aligned}$$

The second line uses the geometric-sum formula and $1-\rho = \mu\gamma$. The last line uses $0 \leq 1-\rho^R \leq 1$ and nonnegative error terms. Finally, substitute $\gamma = \eta Q$, $\Delta_0 = \mathcal{F}(\Theta_0) - \mathcal{F}^\star$, and $B_Q$ from Eq. 18. The factor $5/(2\mu)$ in the theorem is $5/(4\mu)$ multiplied by the factor 2 in $B_Q$. This proves Eq. 16. □

**Reading the bound.** The first term describes how much of the initial loss gap remains after $R$ groups. The second comes from batch-gradient variation. The parameter-difference term depends on $(\tau + Q - 1)^2$, while the embedding-noise term depends on $p\sigma_{\mathrm{dp}}^2$. For fixed values of the other constants, doubling the noise standard deviation multiplies the noise term by four. Increasing $Q$ changes both the contraction factor and the error terms, so the bound does not imply that more local updates always help. With a fixed step size and nonzero errors, the result gives a residual loss-gap bound rather than exact convergence to the minimum.

# E EVALUATION METRICS

UCIHAR (Anguita et al., 2013), Mini-ImageNet (Vinyals et al., 2016), and VisionTouch (Lee et al., 2019) use classification Accuracy, for which higher is better. MuJoCo uses mean squared error (MSE), for which lower is better. Classification results are displayed as percentages, and accuracy differences are reported in percentage points. MuJoCo (Lee et al., 2020) tables consistently use units of $10^{-6}$.

All methods are trained for at most 150 epochs, with early stopping if the monitored performance does not improve for five consecutive epochs.
Training time is measured as wall-clock time from the start of the training procedure until the final output model is available, including training-related warm-up, validation for early stopping, completion of in-progress updates, and final model aggregation. Final test-set evaluation is excluded.

For a classification evaluation set with $N_{\text{eval}}$ samples,

$$\text{Accuracy} = \frac{1}{N_{\text{eval}}} \sum_{j=1}^{N_{\text{eval}}} \mathbf{1}[\hat{y}_j = y_j], \tag{22}$$

where $\hat{y}_j$ is the predicted class, $y_j$ the true class, and $\mathbf{1}[\cdot]$ the indicator function. A difference between two accuracy fractions is multiplied by $100$ to express percentage points.

MuJoCo MSE is computed over all predicted scalars in the entire test set. Let $N_{\text{test}}$ be the number of test sliding windows, each containing 16 time steps with two-dimensional `Cylinder0_pos[:, 0:2]` targets. Then

$$\text{MSE} = \frac{1}{N_{\text{test}} \times 16 \times 2} \sum_{b=1}^{N_{\text{test}}} \sum_{t=1}^{16} \sum_{c \in \{x,y\}} \left(\hat{y}_{b,t,c} - y_{b,t,c}\right)^2 . \tag{23}$$

Here $b$ indexes test windows, $t$ indexes steps within a window, and $c$ is a coordinate component. Predictions and labels both have shape `[batch, 16, 2]`. Squared errors are summed across all windows, time steps, and coordinates before normalization. Predictions in overlapping windows are counted at their respective window positions, without deduplication by original trajectory time.

System metrics are wall-clock training time in seconds and average CPU utilization in percent, reported together with predictive quality. For task $d$, let $T_{\text{Our},d}$ be OrbiVFL's training time and $T^{\min}_{\text{base},d}$ the shortest time among the four competing baselines. The time reduction relative to that baseline is

$$\text{Reduction}_d = \left(1 - \frac{T_{\text{Our},d}}{T^{\min}_{\text{base},d}}\right) \times 100\%. \tag{24}$$

CPU utilization measures the proportion of time that computing resources are busy. Differences are reported in percentage points relative to the highest-utilization baseline on the same task.

GPU utilization measures the proportion of time that GPU compute engines are actively executing kernels. Differences are reported in percentage points relative to the highest-utilization baseline on the same task.

# F DATASETS, MODELS, AND TRAINING CONFIGURATIONS

For all four tasks, the active party holds supervision labels and performs representation fusion and loss computation; passive parties hold their corresponding features or modalities. The standard configuration assigns four workers to the active party and to each passive party. The following tables give task-specific configurations; evaluation batch sizes apply to testing or validation by default.

## F.1 UCIHAR

See Table 5.

Table 5: UCIHAR task, model, and training configuration.

| Item | Configuration |
|---|---|
| Task and target | Human activity classification with six classes |
| Input features | 561 inertial-sensor features |
| Feature partition | One passive party holds 348 acceleration- and gravity-related features; the other holds 213 gyroscope-related features |
| Bottom models | Each of the two passive parties uses an MLP to encode its local features |
| Top model | Concatenates both embeddings and uses an MLP to predict six classes |
| Parties | One active party and two passive parties |
| Workers | Four workers at the active party and at each passive party |
| Training/evaluation batch size | 256/256 |
| Optimizer | Adam (Kingma & Ba, 2014) |
| Learning rate | 0.001 |
| Metric | Accuracy; higher is better |

## F.2 MINI-IMAGENET

See Table 6.

Table 6: Mini-ImageNet task, model, and training configuration.

| Item | Configuration |
|---|---|
| Task and target | Image classification with 100 classes |
| Input images | $3 \times 84 \times 84$ per image |
| Feature partition | Four width-wise patches of $3 \times 84 \times 21$, held by four passive parties |
| Bottom models | Lightweight two-dimensional convolution (Conv2d) encodes each local patch into feature maps |
| Inter-party transmission | Conv2d output feature maps, rather than raw image patches |
| Top model | Concatenates branch feature maps and uses pretrained MobileNetV2 (Sandler et al., 2018) for 100-class prediction |
| Parties | One active party and four passive parties |
| Workers | Four workers at the active party and at each passive party |
| Training/evaluation batch size | 128/256 |
| Optimizer | SGD with momentum 0.9 |
| Learning rate | 0.01 |
| Metric | Accuracy; higher is better |

### F.3 MuJoCo

**Input–target alignment and data splitting.** The position input describes the robot end-effector, whereas the target is the manipulated object's two-dimensional position. Inputs and targets share the same time indices within each 16-step window, without a future-target shift. The target field is excluded from the input features. Training, validation, and test sets are separated at the trajectory level before window extraction. Overlapping windows are generated only within their respective splits, so windows from the same trajectory never appear in different splits. Each window contains 16 consecutive time steps, with a stride of 8; incomplete trailing windows are discarded.

See Table 7.

Table 7: MuJoCo task, model, and training configuration.

| Item | Configuration |
|---|---|
| Task and target | Time-aligned estimation of the manipulated object's 2D position, using Cylinder0_pos[:, 0:2] as the target |
| Input modalities | Robot end-effector position (eef_pos), force/contact measurements, images, and control features |
| Modality partition | One modality group per passive party, with four passive parties |
| Bottom models | All four bottom models include long short-term memory (LSTM) networks (Hochreiter & Schmidhuber, 1997) for modality-specific sequence representations |
| Top model | Concatenates branch sequence representations and applies a linear regression layer at each time step to predict two-dimensional position |
| Output and label shape | [batch, 16, 2]: 16 time steps per window and two coordinate components per step |
| Parties | One active party and four passive parties |
| Workers | Four workers at the active party and at each passive party |
| Training/evaluation batch size | 128/128 |
| Optimizer | Adam (Kingma & Ba, 2014) |
| Learning rate | 0.001 |
| Metric | MSE over all predicted scalars in the test set; lower is better |

### F.4 VisionTouch

See Table 8.

Table 8: VisionTouch task, model, and training configuration.

| Item | Configuration |
|---|---|
| Task and target | Binary prediction of whether contact occurs at the next time step |
| Input modalities | Image, force, proprioception (proprio), depth, and action |
| Modality partition | Five modality groups held by five passive parties |
| Bottom models | CNN, causal one-dimensional convolution (Conv1D), and MLP components extract local representations |
| Top model | Concatenates local representations and uses an MLP to produce binary-classification logits |
| Parties | One active party and five passive parties |
| Workers | Four workers at the active party and at each passive party |
| Training/evaluation batch size | 32/32 |
| Optimizer | Adam (Kingma & Ba, 2014) |
| Learning rate | Default 0.001 |
| Metric | Accuracy; higher is better |

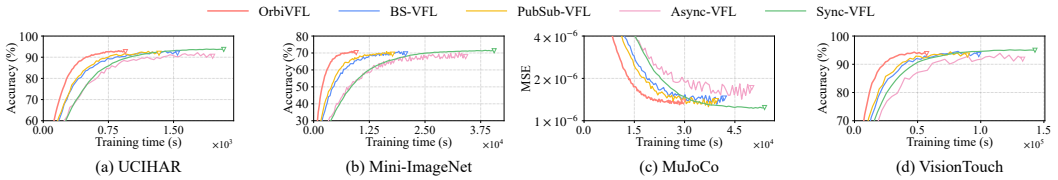



Figure 6: Predictive performance versus elapsed training time.

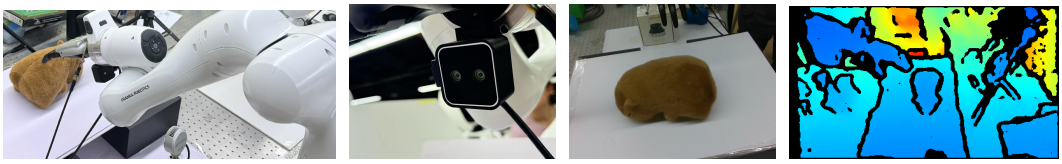


Figure 7: Production-level embodied AI experimental setup example. From left to right, the figures show a robotic arm grasping a toy, the camera setup, the toy object, and the corresponding depth image. In our actual deployment, the dataset is collected from an industrial production scenario in which robotic arms collaborate with an automotive manufacturer for vehicle assembly.

# G Complete Numerical Results

## G.1 Predictive Performance over Training Time

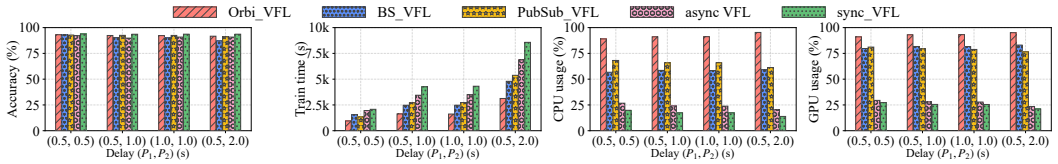
Figure 6 presents accuracy and MSE against elapsed training time on the four datasets, complementing the summary results in Table 1, which shows that OrbiVFL achieves the best performance among the baselines.

## G.2 Production-Level Experiment Setting

In our production-level experiments, we use a Franka Research 3 robotic arm equipped with a Dexterous Hand Series RY-H2 robotic hand. An Intel RealSense Depth Camera D405 is mounted on the robotic arm, and an additional Intel RealSense D435 camera is used to provide a third-person view. We collect trajectory data from both the robotic arm and the robotic hand, together with visual data from the two cameras. In total, we collect 400 trajectories (300 for training and 100 for testing), each lasting approximately 30 seconds. To avoid revealing confidential details of the automotive manufacturing environment, we present only a representative illustration of the deployment scenario, as shown in Figure 7.

In the real-world deployment, the automotive manufacturing company uses an Intel RealSense D435 camera to collect environmental information. The embodied intelligence company uses an Intel RealSense Depth Camera D405 to capture local depth information of manipulated objects, and operates a Franka Research 3 robotic arm equipped with a Dexterous Hand Series RY-H2 robotic hand. Based on our measurements, the average network latency between the two companies is 1.54 s.

At each time step, we synchronize the visual observations with the proprioceptive states of the robotic arm and robotic hand to construct a multimodal trajectory. The Franka Research 3 provides seven actuated joints, while the RY-H2 hand contains six actively controlled degrees of freedom and five passive joints. The robot state is represented by a 26-dimensional proprioceptive feature vector, including the joint positions and joint velocities of the seven arm joints and the six actively controlled hand joints. The visual observation consists of synchronized RGB-D frames captured by the wrist-mounted D405 camera and the third-person D435 camera, which provide complementary close-range and global views of the manipulation process. The input at each time step therefore contains the visual observations from both cameras together with the 26-dimensional robot state. The corresponding control output is a 13-dimensional action vector containing the target commands for the seven arm joints and six actively controlled hand joints. The five passive joints of the RY-H2 are mechanically coupled and are therefore not independently represented in the action space. Each




(a) Accuracy (b) Training time (c) CPU usage (d) GPU usage

Figure 8: Effect of different network latencies.

Table 9: Effect of Gaussian embedding perturbation on UCIHAR.

| Noise level | 0 | 0.1 | 0.5 | 1 | 5 | 10 |
|---|---|---|---|---|---|---|
| Acc | **93.08** | 93.01 | 92.51 | 91.72 | 90.30 | 85.17 |

collected trajectory is organized as a temporally aligned sequence of observations and corresponding control actions and is directly used for policy training and evaluation.

### G.3 Experiment with Network Latency

Figure 8 shows the experimental results under different network latencies. In this experiment, we employ two passive parties, and their respective network latencies to the active party are indicated in the figure. On UCIHAR dataset, we observe that as the network latency increases, the reductions in training time and the improvements in CPU and GPU utilization become more pronounced. In particular, when the network latency reaches 2s, $Q = 15$, which is larger than the default setting as $Q$ increases with network latency. Under this setting, OrbiVFL reduces the training time by up to 63.51%, while improving CPU and GPU utilization by 81.46% and 73.98%, respectively. Compared with the best-performing asynchronous VFL baseline, OrbiVFL further improves training time, CPU utilization, and GPU utilization by 34.75%, 34.05%, and 11.99%, respectively. Across all evaluated network latency settings, OrbiVFL consistently achieves the shortest training time and the highest CPU and GPU utilization among all methods. Although the model accuracy slightly decreases as the network latency increases, the degradation remains within an acceptable range compared with Sync_VFL. This demonstrates that our method can substantially reduce training time under high network latency and effectively alleviate the communication bottleneck.

### G.4 Differential Privacy Experiments

**Privacy definition and noise calibration.** We consider neighboring datasets $D$ and $D'$ that differ in one private training record. A randomized mechanism $\mathcal{M}$ satisfies $(\epsilon, \delta)$-differential privacy if, for every measurable event $O$,

$$\Pr[\mathcal{M}(D) \in O] \leq e^{\epsilon} \Pr[\mathcal{M}(D') \in O] + \delta.$$

For a released embedding query $q$ with bounded $L_2$ sensitivity

$$\Delta_2 = \sup_{D \sim D'} \|q(D) - q(D')\|_2,$$

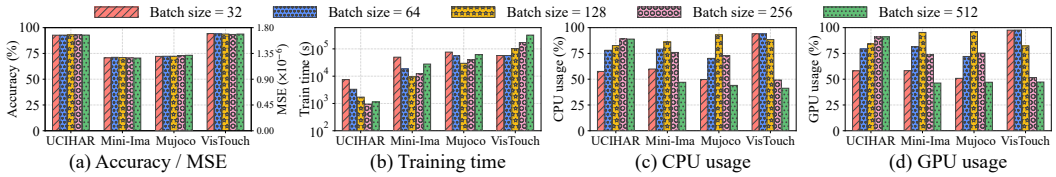
the Gaussian mechanism releases $\widetilde{Z} = q(D) + \xi$, where $\xi \sim \mathcal{N}(0, \sigma_{\mathrm{dp}}^2 I)$. For $0 < \epsilon_0 < 1$, a sufficient condition for $(\epsilon_0, \delta_0)$-DP is

$$\sigma_{\mathrm{dp}} \geq \frac{c\Delta_2}{\epsilon_0}, \qquad c^2 > 2\log(1.25/\delta_0).$$

**Repeated releases and scope.** If each release satisfies the above guarantee conditional on the previous transcript, at most $R_{\mathrm{rel}}$ releases satisfy $(R_{\mathrm{rel}}\epsilon_0, R_{\mathrm{rel}}\delta_0)$-DP by basic composition. New noisy releases generated by retries are included in $R_{\mathrm{rel}}$. This guarantee requires the sensitivity bound to hold throughout training and covers the accounted releases and their post-processing. Other observable data-dependent outputs require separate accounting. The convergence analysis in Appendix D bounds the optimization effect of noise rather than the cumulative privacy loss.

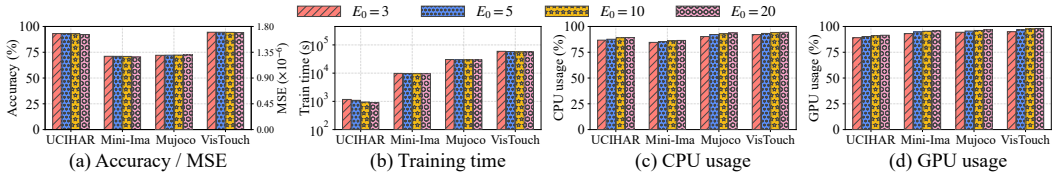
Table 10: Effect of model size on UCIHAR.

| Model | Acc | Train Time (s) | CPU usage | GPU usage |
|---|---|---|---|---|
| OrbiVFL | 93.08 | **945.70** | 89.03 | 91.03 |
| Larger_Model_OrbiVFL | **93.87** | 1539.12 | **92.37** | **94.17** |




(a) Accuracy / MSE (b) Training time (c) CPU usage (d) GPU usage

Figure 9: Effect of batch size.




(a) Accuracy / MSE (b) Training time (c) CPU usage (d) GPU usage

Figure 10: Effect of $E_0$.

Table 9 reports UCIHAR accuracy under different Gaussian noise levels. The implementation parameter $\sigma_{\mathrm{dp}}$ controls perturbation strength and is distinct from the privacy parameters $\epsilon$ and $\delta$. Larger values correspond to stronger perturbations, while $\sigma_{\mathrm{dp}}$=0 is the unprotected baseline. For $\sigma_{\mathrm{dp}} \leq 5$, accuracy decreases by at most 2.78 percentage points relative to the unperturbed setting.

We evaluate embedding inversion using the attack (EIA) of (Song & Raghunathan, 2020). The adversary observes the released embeddings and trains an inverse model using an auxiliary dataset drawn from a similar distribution to the target data. At $\sigma_{\mathrm{dp}} = 5$, the observed attack success rate is 2.71%.

### G.5 Experiment with Larger Models

Table 10 reports the experimental results with larger models on UCIHAR, where we use Four-Layer MLP as the bottom model for each passive party and Three-Layer MLP as the top model.

The results show that OrbiVFL can effectively support larger model configurations. As the model size increases, the model achieves higher accuracy, while training time and both CPU and GPU utilization increase accordingly.

### G.6 Experiments with Batch Size

Figure 9 shows the effects of batch size. As the batch size increases, the accuracy/MSE remains nearly unchanged, while the training time first decreases and then increases. CPU and GPU utilization first increase and then decrease, reflecting the trade-off between computation efficiency and workload granularity. Nevertheless, both communication and computational resources are effectively utilized across different batch sizes, demonstrating the robustness of our method.

### G.7 Experiment with $E_0$

Figure 10 shows the effect of the maximum worker-level synchronization interval $E_0$. As $E_0$ increases, training time decreases and CPU and GPU utilization increase, while predictive performance changes modestly. We choose $E_0 = 10$ as the default to balance training efficiency and predictive performance. These results support the robustness of OrbiVFL to the choice of $E_0$ within the evaluated range.

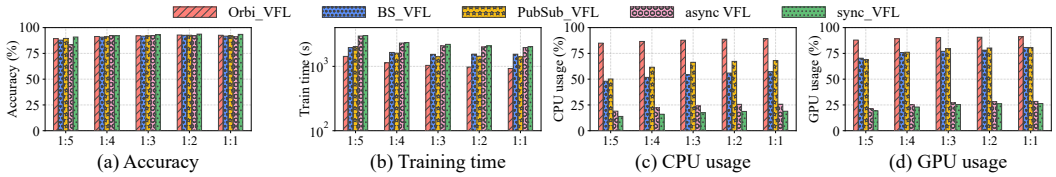



Figure 11: Effect of data heterogeneity.

### G.8 Experiment with Data Heterogeneity

On the UCIHAR dataset (experiment on two parties), we vary the feature-partition ratio between two parties as 1:5, 1:4, 1:3, 1:2, and 1:1. Figure 11 shows the corresponding results. OrbiVFL achieves consistent performance across different feature-partition ratios. Compared with Sync-VFL and the other baselines, OrbiVFL maintains competitive accuracy while reducing training time and improving computational resource utilization. As the feature distribution becomes more balanced, performance improves, yet OrbiVFL remains robust even under substantial data heterogeneity.