FEDERATED FINE-TUNING OF CODE LANGUAGE MODELS: A CONTROLLED COMPARISON OF FEDAVG, FEDPROX, AND FEDADAM UNDER LANGUAGE-HETEROGENEOUS CLIENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language models specialized for source code excel at program synthesis, but further domain gains require fine-tuning on codebases that are often proprietary and cannot be pooled centrally. Federated Learning (FL) offers a path to collaboratively fine-tune a shared model without exchanging raw data, yet the comparative behavior of FL aggregation algorithms on code generation—particularly under realistic, language-based client heterogeneity—has not been measured systematically. We present a controlled empirical study that fine-tunes Qwen2.5-Coder-1.5B with Quantized Low-Rank Adaptation (QLoRA) inside the Flower framework, comparing three aggregation algorithms (FedAvg, FedProx, FedAdam) across three data-partitioning strategies (random IID, Dirichlet α=0.5, and a realistic non-IID split where each client specializes in one programming language). The resulting global models are benchmarked against a centralized upper bound on HumanEval+, MBPP+, and MultiPL-E using greedy pass@1, with three seeds per condition and paired significance testing. Two findings emerge. First, on algorithm choice, FedAvg consistently matches or exceeds FedProx and substantially outperforms FedAdam in every partition (HumanEval+ pass@1 of 40.0, 36.4, 29.1 for FedAvg across IID, Dirichlet, and by-language), whereas FedAdam is unstable and collapses to 11.6 ± 5.8 under quantity skew; contrary to the expectation that server-adaptive optimization best handles heterogeneity, the simplest aggregator is the most accurate and robust. Second, on the cost of federation, the best federated model is statistically indistinguishable from the centralized reference (38.2) under IID and Dirichlet but lags by ≈ 9 points under the by-language split. A per-language MultiPL-E analysis reveals a stable competence gradient from Python and JavaScript down to Go, without collapse toward a dominant language. We release adapters and configurations for reproducibility.

1 INTRODUCTION

Software engineering increasingly relies on large language models (LLMs) for code completion, function synthesis, bug fixing, and documentation. Open models such as Qwen2.5-Coder (Hui et al., 2024), StarCoder 2 (Lozhkov et al., 2024), and CodeGen (Nijkamp et al., 2023) show that decoder-only transformers trained on large corpora of permissively licensed source code solve a non-trivial fraction of programming problems. A persistent obstacle, however, is that further domain-specific gains require fine-tuning on data that many organizations cannot share: source code frequently contains proprietary algorithms, security-critical configuration, or intellectual property that legal and regulatory constraints—such as the GDPR (European Parliament and Council, 2016)—keep on premises.

Federated Learning (FL; McMahan et al., 2017) addresses this tension by letting multiple parties collaboratively train a shared model while raw data never leaves its owner: a coordinator aggregates model *updates* rather than data. Recent work has begun to extend FL to LLMs (Ye et al.,

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.

2024; Kuang et al., 2024; Gao et al., 2025), but the intersection of FL and code generation remains early. Existing work establishes feasibility and general-purpose frameworks, yet there is no systematic comparison of *aggregation algorithms* specifically on code generation, under realistic non-IID partitioning that mirrors how code is distributed across organizations. Because organizations naturally specialize in different programming languages, a particularly relevant but under-studied form of heterogeneity is a non-IID split *by language*. For example, a fintech firm may write overwhelmingly Java, a data-science consultancy almost exclusively Python, and a systems vendor mainly C++ and Go: each client holds code in a different language, no client sees the full distribution, and a global model must reconcile updates pulled in different linguistic directions. This heterogeneity is *semantic*—different syntax, idioms, and libraries—rather than merely statistical, which makes it more scientifically interesting than a generic quantity skew.

We fill this gap through a carefully controlled empirical study. We federated-fine-tune Qwen2.5-Coder-1.5B, a strong small code LLM, using Quantized Low-Rank Adaptation (QLoRA; Dettmers et al., 2023) on a single consumer GPU, and compare FedAvg, FedProx, and FedAdam under three partitioning strategies against a centralized upper bound on three standard benchmarks. We do *not* propose a new algorithm or pretrain a model; the contribution is a rigorous, reproducible comparison of existing tools under realistic and controlled conditions. The study is organized around four research questions:

- **RQ1 (Accuracy gap).** How far does the best federated configuration fall short of a centralized model trained on the union of client data, measured on HumanEval+ pass@1?
- **RQ2 (Algorithm choice).** With base model, partition, and benchmark held constant, which of FedAvg, FedProx, FedAdam achieves the highest pass@1, and does the ranking change as heterogeneity grows?
- **RQ3 (Partitioning by language).** Under a non-IID split where each client specializes in one language, how does the global model perform on each language individually (via MultiPL-E)?
- **RQ4 (Computational cost).** What is the per-round cost—wall-clock, peak memory, communicated parameters—of federated QLoRA on one consumer GPU, and what training budget does it imply for a fair comparison?

Contributions. (i) A complete, reproducible pipeline for federated fine-tuning of a code LLM on a single GPU, integrating QLoRA, the Flower simulation engine, and reference implementations of FedAvg, FedProx, and FedAdam. (ii) A realistic non-IID-by-language partitioning scheme that mirrors enterprise specialization, with per-client data volume held constant so that distributional heterogeneity is isolated from data volume. (iii) A measured characterization of per-round cost on a consumer GPU (RQ4) and a calibrated experimental budget derived from it. (iv) A complete comparison of the three algorithms across all three partitions against a centralized reference, reported with standard deviations and paired significance tests over three seeds, showing that the simplest aggregator (FedAvg) is most accurate and most robust while server-adaptive FedAdam is significantly weaker and markedly unstable, and quantifying the cost of federation as negligible under mild heterogeneity but ≈ 9 points under language-based heterogeneity.

2 Background and Related Work

2.1 Federated Learning Fundamentals

In the canonical FL setup (McMahan et al., 2017), a server holds a global model $w^{(t)}$ at round t; each client k receives it, trains locally on private data $\mathcal{D}_k$ for a number of local epochs, and returns its update $w_k^{(t+1)}$. FedAvg aggregates by a sample-weighted average,

$$w^{(t+1)} = \sum_{k=1}^{N} \frac{|\mathcal{D}_k|}{|\mathcal{D}|} w_k^{(t+1)}. \tag{1}$$

Surveys (Kairouz et al., 2021; Liu et al., 2023a) identify three challenges: *data heterogeneity* (non-IID data causes local drift, so a naive average may be a poor global model), *system heterogeneity*

Table 1: Related work at the intersection of federated learning and code intelligence, and how each relates to this study.

Work (venue)	Contribution	Relation to this study
OpenFedLLM (KDD'24)	Seven FL algorithms; first FL-on-code-generation evaluation.	Closest prior art; we isolate algorithm choice on code tasks and add a by-language split, centralized reference, and significance tests.
FedBiOT (KDD'24)	Bi-level compression; evaluates non-IID-by-language code generation.	Similar by-language setting, but its goal is compression; we compare aggregators head-to-head.
F-CodeLLM (ICSE-C'24)	Short poster: federated LoRA framework for code LLMs.	Similar multi-language framing; we provide the full evaluation it lacks.
FlowerTune (NeurIPS'25)	Cross-domain benchmark; Qwen2.5-Coder coding template.	Practical starting template; we add the algorithm comparison, by-language split, and significance testing.

(stragglers and dropouts), and *privacy leakage* (updates can, under some threat models, be inverted). We focus on the first, most directly relevant to aggregation behavior on code tasks. To counter drift, FedProx (Li et al., 2020) augments each client objective with a proximal term $\frac{\mu}{2}\|w - w^{(t)}\|^2$; the FedOpt family (Reddi et al., 2021) instead replaces the average in Eq. 1 with server-side adaptive optimizers (FedAdagrad, FedYogi, FedAdam) that treat aggregated updates as pseudo-gradients. FedAvg, FedProx, and FedAdam form the core of our comparison because they represent three qualitatively different families—simple averaging, client-side drift correction, and server-side adaptive optimization.

2.2 Federated Fine-Tuning of LLMs

Full fine-tuning of even a 1.5B model on a single consumer GPU is impractical, and communicating billions of parameters per round is prohibitive. The community has converged on parameter-efficient fine-tuning (PEFT), notably LoRA (Hu et al., 2022), which freezes pretrained weights and injects trainable matrices $A \in \mathbb{R}^{d\times r}$, $B \in \mathbb{R}^{r\times d}$ with $\Delta W = BA$, $r \ll d$. QLoRA (Dettmers et al., 2023) additionally quantizes the frozen base to 4-bit NormalFloat (NF4), enabling billion-parameter fine-tuning on consumer hardware. Several codebases combine FL with PEFT: OpenFedLLM (Ye et al., 2024) implements seven FL algorithms and evaluates them across datasets including CodeAlpaca (Chaudhary, 2023), reporting that federated fine-tuning of a smaller model can match or exceed the zero-shot performance of much larger models; FederatedScope-LLM (Kuang et al., 2024) provides a production-oriented pipeline; and FlowerTune (Gao et al., 2025) introduces a coding challenge with a Qwen2.5-Coder baseline, a direct template for our setup.

2.3 Federated Learning for Code Generation

Table 1 summarizes the narrower intersection of FL and code. OpenFedLLM (Ye et al., 2024) is the closest prior art: it benchmarks multiple FL algorithms on code generation but uses a single partitioning scheme and does not isolate algorithm choice on code-specific tasks. FedBiOT (Wu et al., 2024) evaluates non-IID-by-language code generation, but its contribution is model compression rather than algorithm comparison. F-CodeLLM (Cai et al., 2024) is a short poster introducing a federated code-LLM framework without a full evaluation. Adjacent work studies FL for program repair (Luo et al., 2025) and code summarization (Kumar et al., 2024). FlowerTune (Gao et al., 2025) provides the cross-domain benchmark and Qwen2.5-Coder template we build on.

Research gap. No prior work systematically answers, in one controlled setting: how the choice of FL aggregation algorithm affects code-generation performance with base model, PEFT method, and benchmark held constant; how a realistic by-language non-IID split affects per-language perfor-

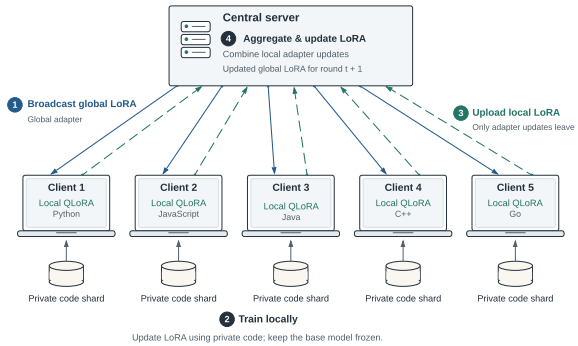

Figure 1: One federated round. The server broadcasts the global LoRA adapter; each of five clients trains locally on its private code shard atop a frozen 4-bit base model and returns only its LoRA update for aggregation. Raw code never leaves the client.

mance; and how the federated model compares to a centralized upper bound. This study answers all three.

3 METHODOLOGY

3.1 OVERVIEW

We simulate a federation of five clients that jointly fine-tune a shared code model without exchanging raw code (Figure 1). In each round, the server broadcasts the current global LoRA adapter; each client performs local QLoRA training on its private shard and returns only its updated adapter; the server aggregates the five adapters with one of the three algorithms and produces the next global adapter. Because the base model is frozen and quantized, only the low-rank adapters—$\approx 1.18\%$ of parameters (Appendix B)—are ever trained or transmitted.

3.2 TECHNICAL STACK

Base model. Qwen2.5-Coder-1.5B (Hui et al., 2024): Apache-2.0 licensed, pretrained on 5.5T code tokens, small enough to fit with adapters and multiple simulated clients on one consumer GPU, and the FlowerTune coding baseline, enabling comparable public baselines. **Fine-tuning.** QLoRA (Dettmers et al., 2023; Hu et al., 2022): a 4-bit NF4 frozen base plus `bfloat16` LoRA at rank r=16, α=32, dropout 0.05, on seven projections (q, k, v, o, gate, up, down), giving 18,464,768 trainable parameters of 1,562,179,072 total (1.18%). **Framework.** The Flower simulation engine (Beutel et al., 2020) runs the five clients sequentially on one GPU; FedProx and FedAdam were ported from OpenFedLLM (Ye et al., 2024). **Algorithms.** FedAvg (McMahan et al., 2017) (sample-weighted average), FedProx (Li et al., 2020) (proximal μ=0.01), and FedAdam (Reddi et al., 2021) (server-side Adam, $\eta=10^{-2}$). SCAFFOLD-style methods are excluded because they require additional client state and roughly double communication, exceeding our budget. **Data.** CodeAlpaca-20k (Chaudhary, 2023) for IID/Dirichlet (as in OpenFedLLM); for the by-language split we draw per-language slices for Python, JavaScript, Java, C++, and Go from *github-code-clean* (CodeParrot, 2022), a content-bearing, permissively licensed corpus (substituted for The Stack v2, whose accessible distribution ships only content pointers). This substitution preserves the by-language intent while keeping data acquisition tractable.

3.3 SYSTEM IMPLEMENTATION

Three engineering measures made federated QLoRA tractable and correct on a single GPU. A *per-actor model cache* builds the quantized base once per simulation worker and reuses it across clients and rounds, swapping only the lightweight adapter when the active client changes; this removed a $\approx 300\times$ redundant-reload overhead that otherwise dominated wall-clock time. *Per-round check-*

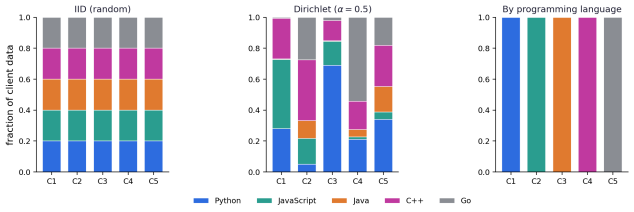

Figure 2: The three data-partitioning strategies, showing each client's language composition. IID gives every client the same balanced mixture; Dirichlet (α=0.5) induces controlled quantity skew; by-language assigns each client a single language.

pointing with automatic resume means an interruption on the shared workstation costs at most one round. A *correctness fix to the FedProx proximal term* snapshots the model's own trainable parameters at the start of local training (when they equal the broadcast global parameters), after which the three algorithms provably produce distinct adapters—a check that specifically caught an earlier proximal-term ordering bug.

3.4 Partitioning Strategies

Three regimes are studied (Figure 2). *IID* shuffles the dataset and splits it uniformly across clients (sanity baseline). *Dirichlet* with concentration α=0.5 produces a controlled quantity-skew split, standard in FL and avoiding the extreme imbalance of small α. *Non-IID by language* assigns each client exactly one language—the study's primary design contribution, reflecting the enterprise reality that organizations specialize by language.

Constant per-client volume. Natural code corpora are highly imbalanced (far more Python than Go). If clients differed in both language and volume, a sample-weighted aggregator would be pulled toward the majority language, and language heterogeneity would be confounded with volume. To isolate the effect of interest, we hold per-client volume *constant*: every client, in every partition, is subsampled to the same 200 examples per round. The by-language comparison therefore varies only the *language* each client sees, so reported differences are attributable to semantic heterogeneity rather than volume. This is a deliberate idealization; the realistic volume-imbalanced regime—where language and quantity skew act together—is left to future work and bounds the external validity of the by-language results.

Fairness and shared hyperparameters. Every federated and centralized run shares one fixed configuration (Table 2); we vary only the aggregation algorithm and the partition. Per-algorithm tuning would confound the comparison—a win could reflect a better-tuned learning rate rather than a better aggregation rule—so algorithm-specific constants (FedProx μ=0.01, FedAdam η=10^{-2}) are set to standard literature values and held fixed. A consequence, examined in Section 5.3, is that FedAdam may operate away from its optimal η; we treat the resulting instability as a finding about the method's sensitivity rather than an unfair disadvantage, since all methods meet the same fixed-configuration standard.

3.5 Experimental Design

The design crosses three partitions with four training methods (a centralized reference plus three federated aggregators); every cell is run with three seeds. Centralized training uses the union of client data and is partition-independent, so the core comparison comprises 27 federated runs (three partitions $\times$ three methods $\times$ three seeds) plus the centralized reference. Each research question is answered by reading the matrix in a different direction: **RQ1** compares the centralized column to the best federated column per partition; **RQ2** compares the three federated columns within each row and across rows; **RQ3** evaluates the by-language global model per language via MultiPL-E; and **RQ4** is answered by the timing probe of Section 4.2. Full participation (all five clients each round) removes client-sampling variance as a confound and is appropriate at this small client count.

Table 2: Fixed configuration shared by all federated and centralized runs.

Component	Setting
Base model	Qwen2.5-Coder-1.5B (4-bit NF4)
LoRA rank / α / dropout	16 / 32 / 0.05
LoRA target modules	q, k, v, o, gate, up, down (7)
Trainable parameters	18,464,768 (1.18% of 1.56B)
Optimizer	AdamW, learning rate 2×10^{-4}, cosine schedule
Batch size	2 per device $\times$ 8 grad. accum. (effective 16)
Sequence length	1024
Precision	`bfloat16` compute
Clients	5, simulated sequentially on one GPU
FedProx μ / FedAdam η	0.01 / 10^{-2}
Federated rounds / local epochs	20 / 5
Examples per client per round	200
Seeds	42, 123, 2026 (three per condition)

Ethical considerations. The study uses only publicly available, permissively licensed datasets and models; no human-subjects data is collected. FL is privacy-motivated but not a privacy panacea: we do not incorporate differential privacy or secure aggregation, so our accuracy findings are *not* privacy guarantees. Model-generated code is executed only in the sandboxed, timeout-guarded EvalPlus subprocesses (Liu et al., 2023b).

4 Evaluation Protocol

4.1 Benchmarks and Metric

Three complementary benchmarks measure functional correctness by executing generated code against unit tests, not textual similarity. *HumanEval+* (Liu et al., 2023b; Chen et al., 2021) extends HumanEval with $\approx 80\times$ more tests (164 Python problems) and is the headline metric, since every recent code-LLM paper reports it. *MBPP+* (Austin et al., 2021; Liu et al., 2023b) (378 problems) corroborates on a broader distribution. *MultiPL-E* (Cassano et al., 2023) translates the problems into multiple languages with matched tests and is essential for RQ3.

The metric is pass@k (Chen et al., 2021): with $n \geq k$ samples per problem and c passing all tests, the unbiased estimator is

$$\text{pass@}k = \mathbb{E}_{\text{problems}}\left[1 - \frac{\binom{n-c}{k}}{\binom{n}{k}}\right]. \tag{2}$$

We report greedy pass@1 (temperature 0, one sample), the strictest single-shot setting and the one a user receives from a single query. Temperature-sampled pass@10, which would reveal whether federated models lose output diversity, is deferred to future work for compute reasons.

4.2 Timing Probe and Compute Calibration (RQ4)

Runs used shared workstations with NVIDIA RTX 3090 (24 GB) GPUs, each run pinned to one GPU; because the EvalPlus sandbox is Unix-only, completions are generated on the GPU workstation and scored on a separate Linux host. Before committing to the full matrix, we measured the true per-round cost of a full-data federated round: five clients over the complete CodeAlpaca partition ($\approx$ 4,000 examples each) with one local epoch completed in 2,500 s, i.e. $\approx$ 0.125 s per example per local epoch. At that rate, the originally proposed budget (100–200 rounds, 10 local epochs, full data) would cost $\approx$ 29 days *per run*—infeasible for $\approx$ 39 runs on a shared machine. We therefore calibrated to 20 rounds, 5 local epochs, and 200 examples/client/round, bringing a run to $\approx$ 3.5 h (Figure 4, Appendix A). The reduction is applied *uniformly* to every condition, so the *relative* comparison—the object of study—remains fair; only absolute scores are conservative. On the communication axis, only the adapter is exchanged: 18.46M parameters ($\approx$ 37 MB/client/round in `bfloat16`) versus $\approx$ 3.1 GB for the full model, an $\approx 80\times$ reduction. Because all three aggrega-

Table 3: HumanEval+ pass@1 (%), mean ± std over three seeds. Base model: 33.5. The centralized reference (union of client data) is partition-independent and repeated per row. Best federated method per row in **bold**.

Partition	Centralized	FedAvg	FedProx	FedAdam
IID (random)	38.2 ± 0.3	$\mathbf{40.0 \pm 2.0}$	37.8 ± 3.2	25.8 ± 4.3
Dirichlet (α=0.5)	38.2 ± 0.3	$\mathbf{36.4 \pm 2.3}$	34.0 ± 2.3	11.6 ± 5.8
Non-IID by language	38.2 ± 0.3	$\mathbf{29.1 \pm 3.1}$	25.0 ± 1.1	18.3 ± 1.6

Table 4: MBPP+ pass@1 (%), mean ± std over three seeds (corroboration benchmark). Best per row in **bold**.

Partition	FedAvg	FedProx	FedAdam
IID (random)	$\mathbf{50.2 \pm 2.4}$	49.5 ± 2.1	36.0 ± 6.6
Dirichlet (α=0.5)	$\mathbf{49.4 \pm 1.6}$	48.8 ± 4.4	14.4 ± 4.5
Non-IID by language	$\mathbf{28.0 \pm 0.2}$	25.3 ± 0.6	24.2 ± 2.6

tors exchange an identically sized adapter and run the same local optimization, their per-round time and communication costs are effectively equal.

5 Results and Discussion

5.1 Main Results

All numbers are greedy pass@1, mean ± standard deviation over seeds $\{42, 123, 2026\}$. The unfine-tuned base model scores 33.5 on HumanEval+; centralized QLoRA reaches 38.2. Table 3 reports HumanEval+ with the centralized reference, Table 4 the corroborating MBPP+, and Figure 3 visualizes HumanEval+; per-seed values are in Appendix A.

5.2 Algorithm Comparison (RQ2)

The clearest signal is a consistent ordering, **FedAvg** $\geq$ **FedProx** $\gg$ **FedAdam**, that holds in *all three* partitions. Under IID, both averaging methods meet or exceed the centralized reference (FedAvg 40.0, FedProx 37.8) while FedAdam trails at 25.8. Under Dirichlet, FedAvg (36.4) and FedProx (34.0) remain close while FedAdam collapses to 11.6. Under the hardest by-language split the ordering is strict and the gaps widest: $29.1 > 25.0 > 18.3$. MBPP+ corroborates the ordering in every partition. Crucially, FedAvg outperforms FedAdam on *every* one of the nine (partition $\times$ seed) HumanEval+ comparisons, so the ordering is not an artifact of seed variance. Contrary to our initial hypothesis that drift-correcting and server-adaptive methods would best absorb heterogeneity, the simplest aggregator is both the most accurate and the most robust—consistent with the recurring finding that a well-configured FedAvg is a strong baseline that more elaborate aggregators do not reliably surpass at small client counts and full participation (Kairouz et al., 2021). The gap between FedAvg and FedProx is small and, as shown below, not statistically significant in any partition: at this scale the proximal term neither helps nor hurts appreciably.

5.3 Instability of Server-Adaptive Optimization

FedAdam is the weakest method in every partition, and under the Dirichlet quantity-skew split it collapses to 11.6 ± 5.8 on HumanEval+ pass@1—over twenty points below the base model, with a per-seed range of 5.5 to 17.1. Even under the benign IID split it recovers only 25.8, below the base model, and across all conditions its standard deviation is roughly two to three times that of FedAvg and FedProx. This behavior is characteristic of server-side adaptivity, which treats aggregated client updates as pseudo-gradients and is known to be sensitive to the server learning rate η and to second-moment estimates when those pseudo-gradients are noisy or drift-dominated (Reddi et al., 2021).

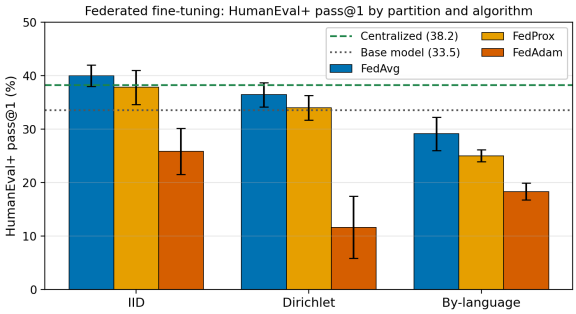

Figure 3: HumanEval+ pass@1 by partition and aggregation algorithm (mean ± std over three seeds). Dashed line: centralized reference (38.2); dotted line: un-fine-tuned base model (33.5). FedAvg is strongest in every partition; FedProx tracks it closely; FedAdam is consistently weakest and collapses under the Dirichlet quantity-skew split. The federated–centralized gap is negligible under IID and Dirichlet but opens markedly under by-language.

Table 5: Paired t-tests on HumanEval+ pass@1 across three seeds. Δ is the mean pairwise difference (A−B) in percentage points; p is the two-sided paired test (n=3). * marks $p < 0.05$.

Partition	**Comparison (A vs. B)**	**Δ (A−B)**	p
IID	FedAvg vs. FedProx	+2.2	0.529
IID	FedAvg vs. FedAdam	+14.2	0.027*
IID	FedProx vs. FedAdam	+12.0	0.060
Dirichlet	FedAvg vs. FedProx	+2.4	0.402
Dirichlet	FedAvg vs. FedAdam	+24.8	0.021*
Dirichlet	FedProx vs. FedAdam	+22.4	0.010*
By-language	FedAvg vs. FedProx	+4.1	0.184
By-language	FedAvg vs. FedAdam	+10.8	0.007*
By-language	FedProx vs. FedAdam	+6.7	0.034*
IID	Centralized vs. FedAvg	−1.8	0.271
Dirichlet	Centralized vs. FedAvg	+1.8	0.288
By-language	Centralized vs. FedAvg	+9.1	0.043*

Held to a fixed η for fairness, FedAdam would require per-condition tuning to be competitive—a practically important negative result for a method frequently proposed as a drop-in improvement over FedAvg: at this model scale and client count, adaptivity buys instability rather than robustness.

5.4 Statistical Significance

We test each headline comparison with a paired two-sided t-test across the three seeds, pairing runs by seed so that seed-induced variance is differenced out (Table 5). FedAvg's and FedProx's advantage over FedAdam is significant ($p < 0.05$) in every partition except the borderline IID FedProx–FedAdam pair (p=0.06). The difference between FedAvg and FedProx is *not* significant anywhere ($p \geq 0.18$). The centralized reference is not significantly different from FedAvg under IID and Dirichlet, but the by-language gap reaches significance at the conventional threshold (p=0.043). With only three seeds these tests are low-powered, so we treat the effect sizes (mean differences) as primary evidence and the p-values as corroboration.

5.5 The Cost of Federation (RQ1)

Reading Table 3 down the FedAvg column against the centralized reference (38.2): under IID, federation matches—indeed marginally exceeds—centralized (40.0 vs. 38.2, Δ=−1.8, p=0.271); under Dirichlet the gap stays small and non-significant (36.4 vs. 38.2, p=0.288); but under by-language a real cost appears (29.1 vs. 38.2, $\approx$ 9 points, p=0.043). The cost of federation is therefore not a fixed

Table 6: Per-language pass@1 (%) under the by-language partition (MultiPL-E; Python from HumanEval+), mean ± std over three seeds. Best method per language in **bold**.

Method	Python	JavaScript	Java	C++	Go
FedAvg	$\mathbf{29.1 \pm 3.1}$	$\mathbf{30.4 \pm 1.1}$	$\mathbf{24.1 \pm 2.2}$	$\mathbf{20.3 \pm 1.6}$	$\mathbf{18.0 \pm 5.3}$
FedProx	25.0 ± 1.1	26.9 ± 2.0	22.2 ± 2.3	18.2 ± 3.8	17.1 ± 2.3
FedAdam	18.3 ± 1.6	21.0 ± 4.9	15.0 ± 2.6	13.8 ± 2.0	11.0 ± 1.9

tax but a function of client heterogeneity: negligible under mild skew, and material—on the order of a quarter of the model's benchmark accuracy—once each client specializes by language. A caveat applies to the *absolute* by-language level: it fine-tunes on github-code-clean source whereas the other conditions use CodeAlpaca instruction pairs closer to the Python-centric benchmarks, so part of the shortfall reflects a data-source shift rather than federation *per se*; the qualitative conclusion rests on the within-condition comparison.

5.6 Per-Language Behavior (RQ3)

Using MultiPL-E, we evaluate every by-language global model on Python, JavaScript, Java, C++, and Go with matched test suites (Table 6; visualized in Figure 5, Appendix A). Two patterns stand out. First, the method ordering is remarkably stable: FedAvg is best on *every* language, FedProx second, FedAdam last, so the RQ2 conclusion hides no per-language reversal. Second, all methods exhibit the same competence gradient—strongest on JavaScript and Python, then Java, then C++, weakest on Go (grand means across methods of 26.1, 24.1, 20.4, 17.4, 15.4 respectively). Because each language is trained on exactly one client's shard, the global model does not collapse toward a single dominant language; it retains graded competence tracking each language's representation in pretraining and the difficulty of its test suite. Higher variance on Go (FedAvg 18.0 ± 5.3) indicates that the lowest-resource language is also the least stable across seeds.

5.7 Limitations

Several caveats bound the interpretation. The reduced, uniformly applied budget makes absolute pass@1 conservative (though relative comparisons remain fair), and the three-seed tests are low-powered, so we lead with effect sizes. The centralized reference was scored on HumanEval only, so RQ1 rests on HumanEval+; and the cross-partition *absolute* comparison is confounded by the CodeAlpaca-versus-github-code-clean data-source difference, so we base conclusions on within-condition comparisons.

6 Conclusion

We presented a controlled study of federated fine-tuning for code generation, comparing FedAvg, FedProx, and FedAdam under IID, Dirichlet, and by-language partitions against a centralized upper bound. On algorithm choice, the aggregators order as FedAvg $\geq$ FedProx $\gg$ FedAdam in every partition and on both benchmarks: FedAvg significantly beats FedAdam and is far more robust, whereas FedAdam collapses under quantity skew—contrary to the expectation that server-adaptivity best handles heterogeneity, the simplest aggregator wins. On the cost of federation, the best federated model is statistically indistinguishable from centralized under IID and Dirichlet but falls short by ≈ 9 points under by-language, so the price of keeping code on premises is negligible when clients are similar but significant once they specialize by language—precisely the enterprise setting that motivates federated code-LLM fine-tuning. A per-language analysis shows graded competence from Python and JavaScript down to Go, without collapse toward a dominant language. Future work includes temperature-sampled pass@10, a local-only lower bound, wider heterogeneity sweeps, privacy-preserving mechanisms, per-condition FedAdam tuning, and scaling to larger models.

Reproducibility Statement

The complete fixed configuration is given in Table 2, the experimental matrix and seed protocol in Section 3.5, the benchmarks and pass@1 procedure in Section 4, and the compute budget in Section 4.2. Every condition is run with three fixed seeds $(42, 123, 2026)$ and reported as mean $\pm$ standard deviation; per-seed values for all conditions are tabulated in Appendix A, and the 1.18% trainable-parameter figure is derived from first principles in Appendix B. Evaluation uses the public EvalPlus (HumanEval+/MBPP+) and MultiPL-E harnesses under greedy decoding, with functional-correctness scoring performed on a Linux host with pinned dependencies. The trained LoRA adapters and the training and evaluation configuration files are released in an anonymized repository as supplementary material so that the full comparison can be reproduced.

Ethics Statement

This study uses only publicly available, permissively licensed datasets (CodeAlpaca-20k, github-code-clean, and the HumanEval/MBPP/MultiPL-E benchmarks) and an Apache-2.0 base model; no human-subjects data is involved. Federated learning is privacy-motivated, but our method incorporates no differential privacy or secure aggregation, and gradient-based attacks can in principle recover information from uploaded updates; our accuracy findings must therefore not be read as privacy guarantees. Code-generation models carry documented risks of producing insecure or biased code; our evaluation measures functional correctness only, and generated code is executed exclusively in sandboxed, timeout-guarded subprocesses. We claim no production readiness for the resulting models.

AI Use Statement

The authors used a generative-AI assistant for language editing, LaTeX formatting, and drafting assistance on portions of the manuscript. All AI-assisted text was reviewed and edited by the authors, and all technical claims, experimental results, statistical analyses, and figures were produced and verified by the authors. Generated code was tested for correctness by the authors. No part of the experimental design, data analysis, or conclusions was delegated to an AI system without author verification. The authors take full responsibility for the final content of this work.

References

Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. GQA: Training generalized multi-query transformer models from multi-head checkpoints. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2023.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

Daniel J. Beutel, Taner Topal, Akhil Mathur, Xinchi Qiu, Javier Fernandez-Marques, Yan Gao, Lorenzo Sani, Kwing Hei Li, Titouan Parcollet, Pedro Porto Buarque de Gusmão, and Nicholas D. Lane. Flower: A friendly federated learning research framework. *arXiv preprint arXiv:2007.14390*, 2020.

Zhi Cai et al. F-CodeLLM: A federated learning framework for adapting large language models to practical software development. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering: Companion (ICSE-Companion)*, pp. 416–417, 2024.

Federico Cassano, John Gouwar, Daniel Nguyen, Sydney Nguyen, Luna Phipps-Costin, Donald Pinckney, Ming-Ho Yee, Yangtian Zi, Carolyn Jane Anderson, Molly Q Feldman, Arjun Guha, Michael Greenberg, and Abhinav Jangda. MultiPL-E: A scalable and polyglot approach to benchmarking neural code generation. *IEEE Transactions on Software Engineering*, 49(7):3675–3691, 2023.

Sahil Chaudhary. Code alpaca: An instruction-following LLaMA model for code generation. `https://github.com/sahil280114/codealpaca`, 2023.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

CodeParrot. github-code-clean dataset. `https://huggingface.co/datasets/codeparrot/github-code-clean`, 2022.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. QLoRA: Efficient finetuning of quantized LLMs. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

European Parliament and Council. Regulation (eu) 2016/679 (general data protection regulation). Official Journal of the European Union, 2016.

Yan Gao, Daniel J. Beutel, et al. FlowerTune: A cross-domain benchmark for federated fine-tuning of large language models. In *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*, 2025.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations (ICLR)*, 2022.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Kai Dang, et al. Qwen2.5-Coder technical report. *arXiv preprint arXiv:2409.12186*, 2024.

Peter Kairouz, H. Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, et al. Advances and open problems in federated learning. *Foundations and Trends in Machine Learning*, 14(1–2):1–210, 2021.

Weirui Kuang, Bingchen Qian, Zitao Li, Daoyuan Chen, Dawei Gao, Xuchen Pan, Yuexiang Xie, Yaliang Li, Bolin Ding, and Jingren Zhou. FederatedScope-LLM: A comprehensive package for fine-tuning large language models in federated learning. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*, 2024.

Ajay Kumar et al. Federated fine-tuning of large language models for code summarization. In *Proceedings of the International Conference on Evaluation and Assessment in Software Engineering (EASE)*, 2024.

Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. Federated optimization in heterogeneous networks. In *Proceedings of Machine Learning and Systems (MLSys)*, volume 2, pp. 429–450, 2020.

Bingyan Liu, Nuoyan Lv, Yuanchun Guo, and Yawen Li. Recent advances on federated learning: A systematic survey. *arXiv preprint arXiv:2301.01299*, 2023a.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. Is your code generated by ChatGPT really correct? rigorous evaluation of large language models for code generation. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023b.

Anton Lozhkov, Raymond Li, Loubna Ben Allal, et al. StarCoder 2 and The Stack v2: The next generation. *arXiv preprint arXiv:2402.19173*, 2024.

Xin Luo et al. Fine-tuning code language models on private data via federated learning for automated program repair. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 2025.

Sourab Mangrulkar, Sylvain Gugger, Lysandre Debut, Younes Belkada, Sayak Paul, and Benjamin Bossan. PEFT: State-of-the-art parameter-efficient fine-tuning methods. `https://github.com/huggingface/peft`, 2022.

Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics (AISTATS)*, pp. 1273–1282, 2017.

Erik Nijkamp, Bo Pang, Hiroaki Hayashi, Lifu Tu, Huan Wang, Yingbo Zhou, Silvio Savarese, and Caiming Xiong. CodeGen: An open large language model for code with multi-turn program synthesis. In *International Conference on Learning Representations (ICLR)*, 2023.

Sashank J. Reddi, Zachary Charles, Manzil Zaheer, Zachary Garrett, Keith Rush, Jakub Konečný, Sanjiv Kumar, and H. Brendan McMahan. Adaptive federated optimization. In *International Conference on Learning Representations (ICLR)*, 2021.

Feijie Wu, Zitao Li, Yaliang Li, Bolin Ding, and Jing Gao. FedBiOT: LLM local fine-tuning in federated learning without full model. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*, pp. 3345–3355, 2024.

Rui Ye, Wenhao Wang, Jingyi Chai, Dihan Li, Zexi Li, Yinda Xu, Yaxin Du, Yanfeng Wang, and Siheng Chen. OpenFedLLM: Training large language models on decentralized private data via federated learning. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*, pp. 6137–6147, 2024.

A Per-Seed Results and Per-Language Figure

Figure 4 shows the compute calibration of Section 4.2, and Figure 5 visualizes the per-language pass@1 of Table 6. Table 7 then lists the per-seed pass@1 scores underlying the aggregated Tables 3 and 4, for all three partitions and the centralized reference. HE and MBPP denote the base benchmarks; HE+ and MBPP+ denote the EvalPlus-extended suites (the headline metrics). The centralized reference was evaluated on the HumanEval suites only.

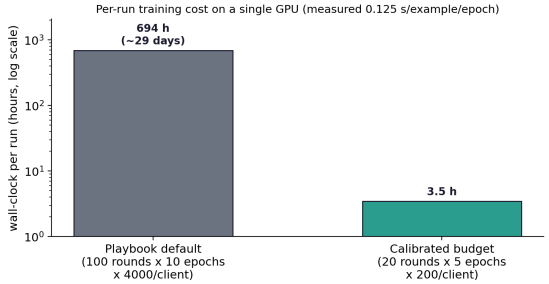

Figure 4: Per-run training cost under the original proposal versus the calibrated budget (log scale), at 0.125 s per example per local epoch. Reducing local epochs, rounds, and per-client volume brings each run from $\approx$ 29 days to $\approx$ 3.5 hours.

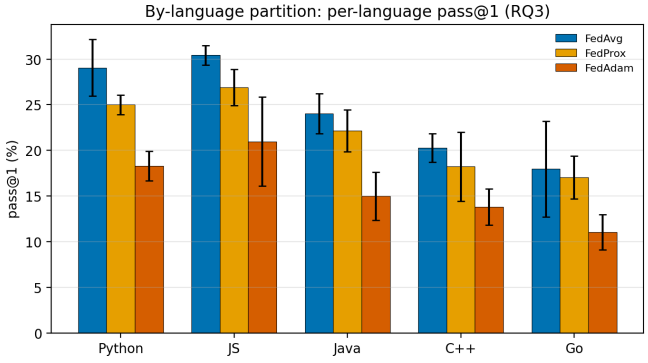

Figure 5: Per-language pass@1 of the by-language global models (mean $\pm$ std over three seeds). A competence gradient runs from Python and JavaScript down to Go, and the FedAvg $\geq$ FedProx $\gg$ FedAdam ordering is preserved within every language.

Table 7: Per-seed pass@1 (%) for all conditions. HE+/MBPP+ are the headline metrics.

Method	Partition	Seed	HE	HE+	MBPP	MBPP+
FedAvg	IID	42	43.29	37.80	61.38	50.53
FedAvg	IID	123	47.56	41.46	58.47	47.62
FedAvg	IID	2026	44.51	40.85	63.49	52.38
FedProx	IID	42	46.95	41.46	62.70	50.26
FedProx	IID	123	40.24	36.59	59.52	47.09
FedProx	IID	2026	40.85	35.37	61.64	51.06
FedAdam	IID	42	29.27	25.00	46.03	33.60
FedAdam	IID	123	35.37	30.49	53.17	43.39
FedAdam	IID	2026	27.44	21.95	40.74	30.95
FedAvg	Dirichlet	42	42.07	34.76	61.64	50.00
FedAvg	Dirichlet	123	42.07	39.02	60.05	50.53
FedAvg	Dirichlet	2026	40.24	35.37	56.08	47.62
FedProx	Dirichlet	42	42.68	36.59	59.26	47.09
FedProx	Dirichlet	123	38.41	32.93	63.76	53.70
FedProx	Dirichlet	2026	37.80	32.32	55.56	45.50
FedAdam	Dirichlet	42	20.12	17.07	25.40	19.31
FedAdam	Dirichlet	123	14.02	12.20	15.87	13.23
FedAdam	Dirichlet	2026	6.71	5.49	13.23	10.58
FedAvg	By-language	42	33.54	29.88	34.13	28.04
FedAvg	By-language	123	29.88	25.61	34.66	27.78
FedAvg	By-language	2026	35.37	31.71	34.66	28.04
FedProx	By-language	42	26.22	23.78	32.01	25.93
FedProx	By-language	123	30.49	25.61	30.95	25.13
FedProx	By-language	2026	29.88	25.61	30.95	24.87
FedAdam	By-language	42	25.00	18.90	28.57	23.54
FedAdam	By-language	123	20.73	16.46	26.72	21.96
FedAdam	By-language	2026	21.34	19.51	32.80	26.98
Centralized	(union)	42	45.70	38.40	—	—
Centralized	(union)	123	43.30	38.40	—	—
Centralized	(union)	2026	42.70	37.80	—	—

B LoRA Parameter Budget

The 1.18% trainable-parameter figure is the ratio of trainable LoRA parameters to the total parameter count of the adapted model, as reported by the PEFT library (Mangrulkar et al., 2022). Each adapted linear layer with a frozen weight $W \in \mathbb{R}^{d_{\text{in}} \times d_{\text{out}}}$ learns a low-rank update $\Delta W = BA$ with $A \in \mathbb{R}^{d_{\text{in}} \times r}$, $B \in \mathbb{R}^{r \times d_{\text{out}}}$, contributing $n_{\text{module}} = r(d_{\text{in}} + d_{\text{out}})$ trainable parameters. At r=16, LoRA is attached to the seven projection modules of each of the L=28 transformer blocks of Qwen2.5-Coder-1.5B (Hui et al., 2024), with hidden size d=1536, key/value projection dimension 256 (grouped-query attention (Ainslie et al., 2023)), and feed-forward dimension 8960. Table 8 applies the formula per module.

Summing the seven modules gives $659{,}456$ trainable parameters per layer and $28 \times 659{,}456 = 18{,}464{,}768$ in total. With a frozen base of $1{,}543{,}714{,}304$ parameters, the adapted model has $1{,}562{,}179{,}072$ parameters, so the trainable fraction is $18{,}464{,}768/1{,}562{,}179{,}072 = 1.18\%$. Only these 18.46M adapter parameters are trained locally and exchanged each round; the 4-bit base is never updated or transmitted. Consistent with PEFT's convention, each 4-bit-quantized base weight counts as one parameter, so the figure is a fraction of *parameters*, not of the model's memory footprint in bytes.

Table 8: Trainable LoRA parameters per module for one transformer layer of Qwen2.5-Coder-1.5B at rank r=16.

Module	$d_{\text{in}} \to d_{\text{out}}$	**Params** $r(d_{\text{in}}+d_{\text{out}})$
q_proj	$1536 \to 1536$	49,152
k_proj	$1536 \to 256$	28,672
v_proj	$1536 \to 256$	28,672
o_proj	$1536 \to 1536$	49,152
gate_proj	$1536 \to 8960$	167,936
up_proj	$1536 \to 8960$	167,936
down_proj	$8960 \to 1536$	167,936
Per layer		**659,456**
All 28 layers		**18,464,768**