SKILLGIT: FEDERATED SKILL EVOLUTION VIA VERSION CONTROL

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language model agents can continuously improve their capabilities by evolving reusable natural-language skills from execution experience. In real-world deployments, however, such experience is often distributed across multiple data owners and cannot be directly pooled due to privacy constraints, which naturally motivates federated skill evolution. A straightforward solution is to let each client independently evolve the shared skill and upload the resulting document for server-side aggregation. However, this document-level update abstraction is poorly suited to skill evolution. Specifically, i) an evolved skill may still encode client-specific information distilled from private traces, ii) repeated transmission of the entire document incurs unnecessary communication when only a small portion is modified, and iii) the actual knowledge contributed by a client is buried within a largely unchanged skill document, making it difficult to identify and evaluate the useful revisions. To address this issue, we propose **SkillGit**, a federated skill evolution framework that reformulates cross-client collaboration from full-document sharing to semantic revision sharing. Instead of uploading complete evolved skills, each client submits compact semantic commits that explicitly describe the intended changes to a shared skill version. SkillGit further introduces a commit-aware aggregation strategy that evaluates and integrates client revisions according to their contribution to the shared skill, allowing complementary knowledge to accumulate while filtering redundant or harmful updates. To stabilize the evolution process, we design a version validation mechanism that evaluates newly merged skill versions with distributed client feedback and rolls back regressions when necessary. Experimental results across three datasets and three LLM families demonstrate that SkillGit outperforms state-of-the-art baselines by up to 22% and approaches centralized evolution without sharing raw client data.

1 INTRODUCTION

Large language model agents increasingly acquire reusable skills from their execution experience rather than repeatedly solving every task from scratch. Recent studies have shown that successful trajectories, failure feedback, and task-specific experience can be distilled into natural-language skills that guide subsequent reasoning and execution (Yang et al., 2026c; Xiong et al., 2026; Yang et al., 2026b). Compared with updating model parameters, evolving such skills is lightweight, interpretable, and compatible with frozen or API-based language models. More importantly, skills provide a persistent interface through which an agent can accumulate knowledge as it encounters new tasks and environments. Effective skill evolution therefore becomes essential for agents that are expected to continuously improve from experience.

However, existing skill evolution methods (Yang et al., 2026c; Xiong et al., 2026; Yang et al., 2026b) largely assume that the experience used for evolution is centrally available. This assumption rarely holds in real-world deployments. Execution traces may be distributed across organizations, users, devices, or environments, each of which observes only a local portion of the task distribution. Pooling these traces could provide substantially richer experience for evolving generalizable skills, but doing so is often prohibited because traces may contain sensitive inputs, private environment states, proprietary workflows, or user-specific information. As a result, independently evolving skills from isolated local experience limits the knowledge available to each agent, while centralized evolution

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.

conflicts with the privacy requirements of distributed data owners. This creates a natural need for collaborative skill evolution without collecting raw execution traces.

Federated Learning (FL) (McMahan et al., 2017; Kairouz et al., 2021; Hu et al., 2024) offers a promising paradigm for such collaboration because it allows multiple clients to jointly improve a shared model while keeping their raw data local. However, directly applying this paradigm to skill evolution introduces a fundamentally different form of client update. Conventional federated learning exchanges compact parameter updates that explicitly encode what has changed during local training. In contrast, a straightforward federated skill evolution approach would let each client evolve an entire skill document and upload the resulting version for aggregation. Such full-document updates suffer from three limitations. First, keeping raw traces local does not necessarily preserve privacy because an evolved skill can still encode client-specific operations, entities, or environmental information distilled from those traces. Second, local evolution often modifies only a small fraction of the shared skill, while repeatedly transmitting the entire document causes communication to grow with the accumulated skill rather than with the newly acquired knowledge. Third, the actual knowledge contributed by a client is buried in a largely unchanged document, making it difficult for the server to identify, evaluate, and combine useful revisions across clients. These limitations reveal a mismatch between the update abstraction of conventional federated learning and the structured natural-language artifacts optimized by skill evolution. Therefore, *how to represent and aggregate client knowledge in a privacy-preserving, communication-efficient, and fine-grained manner is an important challenge for federated skill evolution.*

Our motivation comes from collaborative software development. Git enables multiple developers to improve a shared codebase without repeatedly exchanging complete repository snapshots. Each developer records an intended modification as a commit that captures the change relative to a shared version, while the repository evolves by evaluating and integrating these commits. This suggests a different abstraction for federated skill evolution. Rather than treating a complete evolved skill as the unit of collaboration, a client can communicate only the semantic change discovered from its local experience. Such a representation exposes the newly acquired knowledge directly, reduces the amount of information exchanged, and allows each revision to be independently examined before it becomes part of the shared skill. In other words, federated skill evolution can operate over semantic revisions instead of complete skill versions. The recent FederatedSkill framework (Yang et al., 2026a) takes a related step by exchanging semantic skill patches, but it uses them to maintain a separate personalized library for each client. SkillGit instead evolves a single document shared by the whole population, which is what allows every merge to be validated against the population's own validation union and rolled back when it degrades.

Inspired by this observation, we propose **SkillGit**, a federated skill evolution framework based on version control. SkillGit maintains a shared skill version at the server and lets participating clients evolve it using their private execution experience. Instead of returning the complete evolved document, each client extracts compact semantic commits that describe the revisions it intends to introduce. To effectively combine knowledge contributed by heterogeneous clients, we design a commit-aware aggregation strategy that evaluates individual commits and selectively integrates complementary revisions while filtering redundant or harmful ones. We further introduce a version validation mechanism that evaluates each newly aggregated skill using distributed client feedback and rolls back updates that degrade the shared skill. This design changes the basic unit of federated skill evolution from complete documents to semantic revisions, allowing the shared skill to accumulate knowledge across clients without directly exposing their raw traces.

We evaluate SkillGit across three datasets and three LLM families and compare it with state-of-the-art skill evolution and federated aggregation baselines. Experimental results show that SkillGit improves over the strongest baseline by up to 22% while approaching the performance of centralized skill evolution without sharing raw client data. We further investigate the effectiveness of commit-level aggregation, version validation, privacy protection, communication efficiency, and different federated configurations. Our main contributions are summarized as follows.

- We propose **SkillGit**, which introduces semantic commits as the unit of federated skill evolution. By communicating the intended revisions rather than complete skill documents, SkillGit enables fine-grained knowledge sharing while reducing unnecessary information exchange and limiting the exposure of client-specific content.

- We design a commit-aware aggregation strategy to evaluate and integrate semantic revisions contributed by heterogeneous clients, together with a version validation mechanism that detects performance regressions and restores previous skill versions when necessary.
- Extensive experiments across three datasets and three LLM families demonstrate that SkillGit consistently outperforms state-of-the-art baselines.

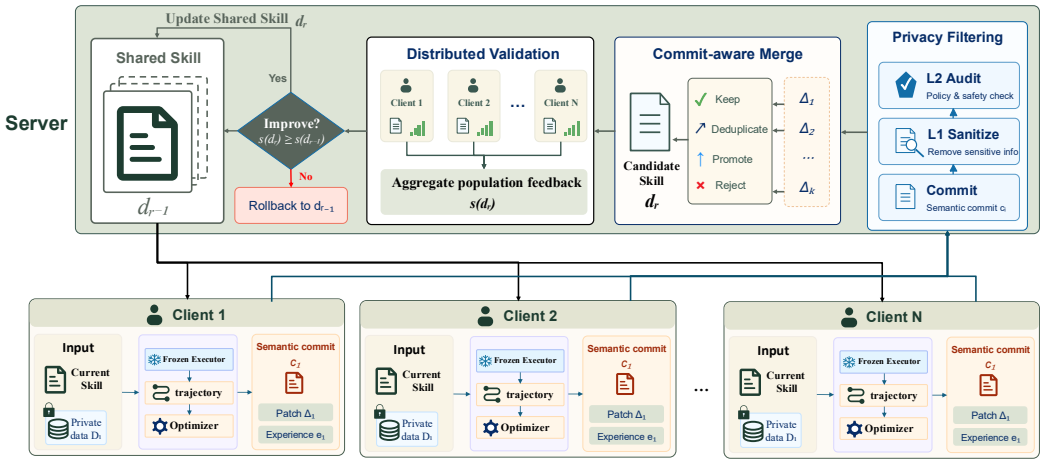


Figure 1: Framework of SkillGit. A scheduled subset of clients receives the published document and evolves it on its own private slice; each participant uploads a single *commit* that packages a *patch* over named rules with an *experience* summary that is routed only to the optimizer. Commits pass the two-stage privacy treatment, first L1 sanitization and then an offline L2 audit against a private-value set extracted from the client's own slice, before reaching the server, which merges them *by disposition*. The candidate document is deployed and scored on the *validation union* of the population; it is published only if it does not degrade that union, otherwise the previous document is restored. Training stops after two consecutive rollbacks, and the two rollback rounds are excluded from the reported convergence round count.

2 Related Work

2.1 Federated Learning

Federated learning (FL) trains a shared model from decentralized data. FedAvg (McMahan et al., 2017) averages locally trained weights, and later work addresses its degradation under non-IID data (Zhao et al., 2018) by constraining local drift (Li et al., 2020b; Karimireddy et al., 2020), correcting objective inconsistency (Wang et al., 2020), adding dynamic regularization (Acar et al., 2021), or personalizing per client (Fallah et al., 2020; Dinh et al., 2020); with heterogeneous models, aggregation is replaced by ensemble distillation (Lin et al., 2020). Surveys and open problems are collected in (Li et al., 2020a; Kairouz et al., 2021), and client selection under resource heterogeneity is studied in (Nishio & Yonetani, 2019). In all of them the shared object is a parameter vector and an aggregation operator, typically a weighted mean, is assumed to exist.

That assumption is what this paper has to give up. A weighted mean is meaningful for a parameter vector because the coordinates of two models are commensurable, but two skill documents do not have that property: their coordinates are rules written in natural language, two documents may disagree about which rules should exist at all, and the arithmetic mean of two rule sets is not a rule set. Federated learning can therefore be carried over to text only if the aggregation operator is replaced by something that selects among whole proposals rather than blending them, which is the role disposition-based merging plays here.

2.2 SKILL EVOLUTION AND SELF-EVOLVING AGENTS

LLM agents increasingly rely on a skill library or skill document, and recent work has automated its evolution. Trajectories are distilled into reusable skills (Yang et al., 2026c; Ju et al., 2026; Zhou et al., 2026; Jiang et al., 2026); skills are consolidated hierarchically (Xiong et al., 2026; Liang et al., 2026) or compressed (Gao et al., 2026; Tu et al., 2026); verified by co-evolution (Zhang et al., 2026) or produced by a parallel fleet and consolidated into a conflict-free directory (Ni et al., 2026); curated, connected and forged at scale (Wang et al., 2026; Ouyang et al., 2026; Shen et al., 2026; Liu et al., 2026; Tian et al., 2026); evolved with reinforcement learning (Shi et al., 2026; Xia et al., 2026; Wu et al., 2026); or treated as a collective process driven by an agentic evolver (Ma et al., 2026). A neighbouring question is what language models learn from context (Si et al., 2026). Two properties of this literature matter here. First, it is almost entirely single-owner, so one agent or task stream evolves one library and the data bottleneck a federation would remove is left in place. Second, the unit of improvement is a text artifact, such as rules, procedures or file-level patches, which is what makes the problem amenable to semantic rather than numeric aggregation.

2.3 FEDERATED SKILL EVOLUTION

SkillGit sits where these two lines meet, and it differs from both along four axes. The shared object is one skill document for a frozen executor rather than a parameter vector, so federating it removes the single-owner bottleneck of the previous subsection without pooling raw data. Contributions are discrete semantic edits merged by disposition, namely kept, deduplicated, promoted or rejected, under a lossless-first, full-coverage rule; there is no averaging operator, and the server never rewrites client text. Publication has to be earned, because every merged document is scored on the validation union of the population and a candidate that degrades it is rejected and the previous document restored (§3.3). Convergence is therefore read off the population's own rollbacks rather than off a fixed budget. Each client's upload is one commit that packages a patch with an experience digest routed only to the optimizer (§3.2 and §3.4), and the privacy claim is checked on the transmitted artifact by an L1 sanitizer paired with an L2 audit against a private-value set built from the client's own slice (§3.5).

The closest prior work is FederatedSkill (Yang et al., 2026a), which also replaces the full-document update with a semantic patch and therefore shares our diagnosis of the document-level abstraction. The two frameworks then part on three decisions. First, FederatedSkill keeps a separate library per client and returns personalized updates, whereas SkillGit evolves one document that the whole population shares and is scored on. That choice is what makes an acceptance test possible at all: a personalization objective has no single artifact whose quality the population can jointly certify. Second, because the shared object is common, SkillGit can gate each merge on the validation union and roll back a regression, which yields an artifact that is monotone by construction and a stopping rule read off the population's own failures. FederatedSkill assumes a trusted environment and reports no such acceptance criterion. Third, patch abstraction alone does not make the upload safe, since a patch can still carry client-specific values distilled from private traces; SkillGit therefore treats privacy as an explicit sanitize-then-audit procedure over the payload that actually crosses the network.

3 METHOD

3.1 PROBLEM SETTING

We consider clients $C = \{1, \ldots, N\}$ that share one task but hold disjoint, frozen private slices: client i owns a training slice D_i and a validation set V_i, and no party may disclose raw data to another party or to the server. A seed skill document d_0 is broadcast to every client (the corpus's own where one exists, otherwise an empty skill with only the section headers). A frozen executor f_{exec} conditions on a document and produces an answer; the same model acts as the optimizer, $f_{opt} = f_{exec}$, and reads trajectories rather than labelled answers.

Evolution proceeds in rounds r = 1, 2, ... At round r a scheduler selects a participating subset $S_r \subseteq$ C; each participant receives the currently published document d_{r-1} and its own slice. For every method, the reported artifact is the document published at termination; this rule is fixed before any

run. All arms use the same per-round budget envelope and are therefore comparable in the amount of private data exposed per round.

Three further conventions keep the setting honest. The scheduler is fixed in advance rather than adapting to the observed scores, so the cost of a run is decided before its outcome is known and no arm can buy accuracy with a better selection policy. A client that is not scheduled in a round contributes nothing to that round, which is what makes the participation ratio a real variable rather than a bookkeeping choice. And the seed document is held fixed across arms, so every reported difference comes from how the document was evolved rather than from where it started.

3.2 Semantic commits and commit-aware aggregation

At round r, each participant p $\in S_r$ evolves d_{r-1} on its own slice and submits a bounded semantic edit: a few operations on *named* rules, namely *append*, *insert after*, or *replace*. Submissions are label-free: they are justified by the executor's behaviour rather than by gold answers. The submission size is capped by a per-round budget held constant across the study, so what a participant uploads scales with what it learned that round, not with the accumulated skill.

The commit. A participant's upload is a single structured unit, a commit, packaging (i) a patch, which is the structured edit set over named rules (add / edit / delete), and (ii) an experience digest, a locally distilled summary of what the round's trajectories taught the participant. The two travel together because they arise from the same local episode but are consumed differently (§3.4): the patch is a candidate change to the shared artifact, the digest is optimizer-only context.

The in-client evolution follows a SkillOpt-style arrangement (Yang et al., 2026b): the target (our executor) runs the document on the client's slice and produces trajectories, and an optimizer reads them and distils the patch, so proposals are grounded in observed behaviour; in the experiments a single model per family plays both roles (§4.1).

A commit is content-addressed: its identity derives from the parent document's hash, the seed lineage, the slice hash, the round and attempt identity, and the evidence hashes of its edits. Identical contributions therefore collapse onto one commit, and a payload edited after the fact no longer matches its recorded hash. The addressing scheme is what lets the merge step treat a submission as a proposal with a history rather than as anonymous text, and it is also what makes duplicated effort visible: two participants who independently propose the same correction produce the same identity and can be merged without either of them being penalised for the overlap.

The server aggregates commits by disposition rather than by averaging: every edit is classified as *kept*, *deduplicated*, *promoted to a shared rule*, or *rejected as invalid*, under a lossless-first, full-coverage rule. This commit-aware aggregation evaluates each proposal against its contribution to the shared skill, so complementary revisions accumulate while redundant or harmful ones are filtered out. An edit is discarded only when it duplicates an already-kept edit or is structurally invalid. Because the shared object is a document rather than a vector, merging is semantics-preserving: the server never averages text, and the published document is one that some party proposed. The disposition record is what keeps the merge accountable, since every submitted edit is accounted for and a reader can see which proposals were promoted and which were dropped.

3.3 Version validation: the rollback gate and convergence

A merged document is not guaranteed to be an improvement: an edit that helps one participant's slice can degrade the shared document on another's. SkillGit therefore treats publication as a deployment decision that must be earned, through version validation: a validation-union rollback gate.

The client population maintains, for the whole run, the union of the clients' validation sets, $V_\cup = \bigcup_{i \in C} V_i$. Each round, the matured candidate document d_r (the merge of that round's commits) is deployed to the clients and scored on $V_\cup$. Let s$(\cdot)$ denote the resulting success rate on $V_\cup$. Then

- if s$(d_r) \geq$ s(d_{r-1}), the candidate is accepted and d_r is published;
- otherwise the candidate is rejected: publication is rolled back and $d_r := d_{r-1}$.

Because improvement is not monotone, the protocol does not fix the number of rounds. Training terminates as soon as two consecutive rollbacks occur, which we read as evidence that the shared document has stopped improving on the population's validation union. The convergence round count reported by all methods is the number of rounds up to and including the last accepted round *before* that pair: if the rollbacks occur at rounds k and k+1, it is $r^* = k - 1$, i.e. the two rollback rounds are excluded; Figures 2 and 3 are read on this axis.

The gate is population-level and label-free with respect to the *task*: it consumes validation behaviour only, so no client reveals labels or validation sets to the server. Because it can always roll back to the last accepted document, the published artifact is monotone by construction.

3.4 The training-side experience channel

The protocol optionally maintains an experience channel on the training side. The digests carried by the commits (§3.2) are aggregated deterministically into a round memory, attached to the optimizer's context in the next round and at merge time. The channel is deliberately one-sided: it changes what the optimizer knows but is never written into the document the executor reads. Its composition is reported as typed entries, so "the channel is empty" is an observable property of the artifact. §4.3 ablates this channel.

Building a digest does not require a model. A digest is *derived* from the round's recorded step records by fixed rules, so the same evidence always yields the same digest, and no model call is spent on it. It records the round, client and epoch identity, the step count and the local gate decision, per-step outcomes with the validation score before and after, the failure and success patterns with their evidence hashes, the patterns that rejected edits failed to address, and a `privacy` block asserting that no raw task or trajectory is carried. The round memory holds a fixed budget of typed entries, so the channel's size is a constant of the protocol.

3.5 Privacy: a sanitizer and a private-value audit

Privacy is enforced on the outbound commit, in two stages. *L1, sanitization:* Every free-text field of a commit is rewritten by a language model instructed to strip private content such as entities, references and other identifying material, and replace it with typed placeholders; rule identifiers and edit operations survive unchanged, so the patch still applies after sanitization.

L2, audit against a private-value set: For each client, a language model extracts a private-value set from that client's own training slice. These are the literal values whose appearance in a transmitted payload would constitute a leak (file paths, table identifiers and comparable artifacts). The sanitized commit is then audited offline against this set; a payload whose rule text contains an element of the set is judged to be a leak and returned to L1 for a rewrite, and only payloads that pass are transmitted.

The two stages are deliberately asymmetric: the sanitizer is a *rewriter* that sees the payload, the private-value set is built from the client's own data and never leaves the client, and the audit consumes the frozen set and the sanitized text, so a verdict is reproducible. §4.4 reports the audit verdicts of real payloads, with and without L1. Regarding scope, the audit tests whether the transmitted text contains values drawn from the client's own private set. That is evidence about what the artifact carries, not a guarantee against inference from aggregate statistics; where formal privacy mechanisms are required, it should complement rather than replace them.

4 Experiments

4.1 Setup

Models: We evaluate three model families, one per row group in Table 1: GPT-5.4-mini, Qwen3.5-flash and Gemini3-flash; within each family, the same model plays both roles, so differences cannot be attributed to an asymmetric executor/optimizer split.

Datasets: SearchQA (open-domain question answering), LiveMath (research-level mathematics multiple choice) and DocVQA (document visual question answering). All three corpora ship their own seed skill document, so every run starts from the corpus's own seed.

Protocol: The main comparison uses 10 clients with a 30 % participation ratio (three participants per round), under the same per-round budget envelope and gating as the other arms below, except *Independent*, which is gated by each client's own validation set. Data splits: each dataset is partitioned so that no two clients hold the same examples. Each client's slice is drawn from a different part of the task distribution, producing a deliberately non-IID split across question type, difficulty band and answer format; the held-out evaluation set is disjoint from every client's slice.

Metric and pairing: The reported quantity is the task success rate (TSR) on a sealed held-out split, graded per dataset; the convergence round count (§3.3) is reported only in Figures 2 and 3. Every paired comparison below uses the same items within the same run.

Arms: All arms use the same executor, the same per-round exposure, the same merge machinery wherever an aggregation step exists, and the same rollback gate; they differ only in *how* the document is produced.

- No skill. The frozen executor is queried with no skill document, which is the floor separating what the model already knows from what the evolved document adds.
- Independent. Each client evolves its own copy of the seed, gated by its own validation set V_i under the same two-consecutive-rollback rule; the reported value is the mean over clients. This is the "no federation" reference.
- Centralized. One process trained on the pooled per-round exposure (the same per-round data as the federated arm), gated by $V_\cup$. This is the "no privacy constraint" reference.
- LLMFed. Each client uploads exactly one mature document, and a strong optimizer merges the set in one call, gated by $V_\cup$. This isolates *iterative* scheduling against a single-shot merge.

4.2 THE MAIN COMPARISON: FEDERATING ONE SHARED DOCUMENT

Table 1: Main results. Task success rate (%) on three datasets, by model family and method. 10 clients, 30 % participation. Each group puts the Centralized ceiling above a rule; the four methods below that rule are the comparison, and the best value in each column among them is set in bold.

Model	Method	SearchQA	LiveMath	DocVQA
GPT-5.4-mini	Centralized	79.4	28.8	86.4
	No skill	75.9	14.7	70.9
	Independent	76.1	27.2	82.8
	LLMFed	77.5	28.0	81.3
	SkillGit	**78.6**	**28.8**	**88.5**
Qwen3.5-flash	Centralized	83.1	32.8	90.4
	No skill	77.7	23.2	77.6
	Independent	80.1	25.6	83.7
	LLMFed	81.2	27.2	80.2
	SkillGit	**82.4**	**29.6**	**85.3**
Gemini3-flash	Centralized	85.9	41.6	91.4
	No skill	79.2	31.2	86.6
	Independent	77.2	31.2	88.4
	LLMFed	83.1	32.8	88.4
	SkillGit	**87.3**	**40.0**	**90.1**

Table 1 reports the main comparison: five arms per model group, SkillGit emphasised; all cells are task success rate (%). The table supports four observations. First, the no-skill floor provides the reference for the gain from an evolved document. Second, federating the shared document beats letting each owner evolve its own copy: SkillGit exceeds Independent in all nine cells, by +1.6 (GPT-5.4-mini, LiveMath) to +10.1 (Gemini3-flash, SearchQA), mean +4.3. Third, SkillGit recovers most of the benefit of centralizing: the median gap against Centralized is -0.8 and SkillGit leads in two of the nine cells; by family averages it leads on GPT-5.4-mini (+0.4), trails on Gemini3-flash (-0.5) and Qwen3.5-flash (-3.0). Fourth, single-shot merging is not enough: LLMFed trails in all

nine cells (+0.8 to +7.2), because a single aggregation must commit to one reconciliation of every submitted edit at once, with no later round in which to correct a bad one.

The margins are also informative. The smallest ones fall where the decisive evidence is narrow: on LiveMath for GPT-5.4-mini and Qwen3.5-flash, SkillGit finishes only 1.6 and 4.0 points clear of Independent, because a rule over-fitted to one client's slice can survive the pooled-union test when the evidence is thin. The margin is widest where the task demands breadth: SkillGit leads on DocVQA for all three families, whose instances are the most heterogeneous across clients, so a single owner's slice is a biased sample. The residual gap to the Centralized ceiling measures what the privacy constraint costs, and it is largest on Gemini3-flash, the family with the highest scores and therefore the least headroom left to recover.

4.3 Ablation I: the training-side experience channel

Table 2: Experience ablation (GPT-5.4-mini + SearchQA). Task success rate (%).

Variant	TSR (%)
Full SkillGit	78.6
w/o experience in next-round training	76.9
w/o experience in the federated merge	77.2

Table 2 ablates the experience channel on GPT-5.4-mini + SearchQA: both ablated arms keep the same commits, merge operator and gate, differing only in whether the experience digest reaches the optimizer at the next round's training and at merge. Both ablations reduce task success relative to the full protocol (76.9 and 77.2 against 78.6): the channel is useful in both roles it plays, and the two effects are of similar size. The gain is not a leak, because the channel never enters the artifact the executor reads, and scanning every published document for memory markers returns zero hits, so the difference must be attributed to what the optimizer knows rather than to what the executor is shown.

At merge time the digest tells the merge step which submitted edits have already been tried and rolled back, so a proposal that looks attractive in isolation is recognised as a repetition and rejected instead of re-published. At training time it tells the optimizer what the previous document failed to fix on that client's slice, so the next proposal starts from the residual error rather than re-deriving it. Recovering one to two points of task success from a digest that costs no model call is what makes the channel cheap enough to keep on by default.

4.4 Ablation II: the L1 sanitizer

Table 3: Privacy ablation (GPT-5.4-mini + SearchQA). Share of payloads that pass the L2 audit on the first attempt, with and without the L1 sanitizer.

Setting	L2 first-pass rate (%)
With L1	81
Without L1	33

Table 3 evaluates the two-stage privacy treatment of §3.5 on the payloads produced by real runs, again on GPT-5.4-mini + SearchQA. Both rows are first-pass rates, so the table isolates the sanitizer: with L1 in place 81 % of payloads clear the audit on the first attempt, against 33 % when it is removed, so the share rejected falls from 67 % to 19 %. Two consequences follow: the check runs on the artifact that actually crosses the network, and the audit consumes the same commit that carries the patch, so privacy needs no second communication path.

4.5 Heterogeneity I: the number of clients

Figure 2 plots the convergence round count of §3.3 against the number of clients. The count rises monotonically from 5 rounds at ten clients to 9 at a hundred, and the growth is back-loaded: ten to thirty clients costs one round ($5 \rightarrow 6$), while fifty to a hundred adds two ($7 \rightarrow 9$). Early rounds

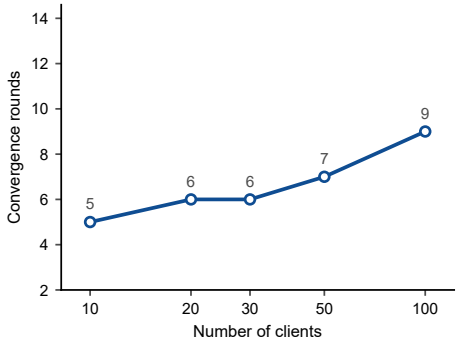


Figure 2: Convergence round count as a function of the number of clients, with the participation ratio fixed at 30 % (GPT-5.4-mini + SearchQA). The count rises monotonically with the population and the growth is back-loaded.

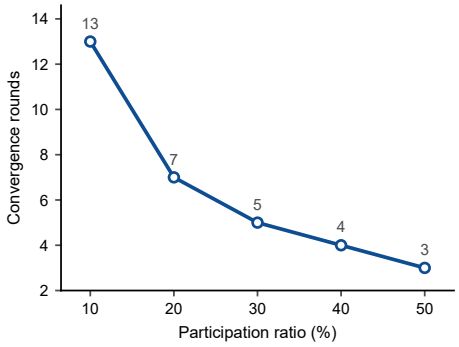


Figure 3: Convergence round count as a function of the participation ratio, with the number of clients fixed at 10 (GPT-5.4-mini + SearchQA). Denser participation shortens the path sharply, with strongly diminishing returns.

have the most headroom, so most submissions merge and each round yields a large accepted patch; as the document improves, the remaining errors are the ones clients disagree about, so submissions collide, fewer edits survive, and the rollback counter takes longer to trip.

4.6 Heterogeneity II: the participation ratio

Figure 3 asks the mirror question. Raising the participation ratio from 10 % to 50 % shortens the path from 13 rounds to 3, because a larger subset brings more evidence per round, while the returns are sharply diminishing: 10 % $\rightarrow$ 20 % removes six rounds (13 $\rightarrow$ 7), 40 % $\rightarrow$ 50 % removes one (4 $\rightarrow$ 3).

5 Conclusion

We asked whether a single shared skill document can be evolved federatively and whether this beats letting each data owner evolve its own copy. SkillGit answers both in the affirmative. Clients submit bounded, label-free commits; a server merges them by disposition; a validation-union rollback gate decides what is published, so the artifact is monotone by construction and convergence is read off the population's own rollbacks. Across three model families and three datasets, federating the shared document improves task success over independent evolution in every cell by +1.6 to +10.1 points, mean +4.3, and lands within about a point of centralized training (median -0.8) while no raw data leaves a client; an iterative, gated schedule also beats a single-shot merge in every cell. Part of the gain comes from a training-side experience channel that never reaches the executor, and the L1 sanitizer lifts first-pass clearance of the L2 audit from 33 % to 81 %. Finally, the convergence cost has a shape an operator can act on: it rises with the number of clients (5 $\rightarrow$ 9 rounds from ten to a hundred) and falls sharply with the participation ratio (13 $\rightarrow$ 3 rounds from 10 % to 50 %), which turns the protocol's operating point into an informed decision rather than a hyperparameter to be tuned blindly.

AI USE STATEMENT

In this work, we used generative AI tools for method design discussions, creating and modifying code, helping interpret experimental results, producing figures, and polishing the manuscript. We have reviewed all AI-assisted work. LLM-generated code was executed and its outputs were checked against the recorded experimental artifacts by the authors; AI-suggested interpretations of results were verified against the raw outputs before being adopted; and every figure was reviewed by the authors for correctness against the underlying data. All other tasks with required disclosure are not applicable to this work. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

§3 specifies the protocol completely: the client/optimizer roles, the structure of a commit (patch plus experience digest), the disposition-based merge with its lossless-first, full-coverage rule, the validation-union rollback gate, and the two-consecutive-rollback stopping rule that defines the convergence round count. §4.1 gives the model families, the datasets, the client and participation settings, and how the non-IID split is constructed. The executor, optimizer and merge prompts open the Appendix. All datasets are public. We plan to release the code and the configuration files of the experiments reported here in a public repository, so that the protocol can be re-run and audited from that repository as well as re-implemented from the description above.

REFERENCES

Durmus Alp Emre Acar, Yue Zhao, Ramon Matas Navarro, Matthew Mattina, Paul N. Whatmough, and Venkatesh Saligrama. Federated learning based on dynamic regularization. In *ICLR 2021*, 2021.

Canh T. Dinh, Nguyen H. Tran, and Tuan Dung Nguyen. Personalized federated learning with moreau envelopes. In *NeurIPS 2020*, 2020.

Alireza Fallah, Aryan Mokhtari, and Asuman Ozdaglar. Personalized federated learning with theoretical guarantees: A model-agnostic meta-learning approach. In *NeurIPS 2020*, 2020.

Yudong Gao, Zongjie Li, Yuanyuan Yuan, Zimo Ji, Pingchuan Ma, and Shuai Wang. Skillreducer: Optimizing llm agent skills for token efficiency. *arXiv preprint arXiv:2603.29919*, 2026.

Ming Hu, Yue Cao, Anran Li, Zhiming Li, Chengwei Liu, Tianlin Li, Mingsong Chen, and Yang Liu. FedMut: Generalized federated learning via stochastic mutation. In *AAAI Conference on Artificial Intelligence*, pp. 12528–12537, 2024. doi: 10.1609/aaai.v38i11.29146.

Guanyu Jiang, Zhaochen Su, Xiaoye Qu, and Yi R. Fung. Xskill: Continual learning from experience and skills in multimodal agents. *arXiv preprint arXiv:2603.12056*, 2026.

Ruofei Ju, Xinrui Wang, Xin Ding, Yifan Yang, Hao Wu, Shiqi Jiang, Qianxi Zhang, Hao Wen, Xiangyu Li, Weijun Wang, Kun Li, Yunxin Liu, Haipeng Dai, Wei Wang, and Ting Cao. Embodiskill: Skill-aware reflection for self-evolving embodied agents. *arXiv preprint arXiv:2605.10332*, 2026.

Peter Kairouz, H. Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, Rafael G. L. D'Oliveira, Hubert Eichner, Salim El Rouayheb, David Evans, Josh Gardner, Zachary Garrett, Adrià Gascón, Badih Ghazi, Phillip B. Gibbons, Marco Gruteser, Zaid Harchaoui, Chaoyang He, Lie He, Zhouyuan Huo, Ben Hutchinson, Justin Hsu, Martin Jaggi, Tara Javidi, Gauri Joshi, Mikhail Khodak, Jakub Konečný, Aleksandra Korolova, Farinaz Koushanfar, Sanmi Koyejo, Tancrède Lepoint, Yang Liu, Prateek Mittal, Mehryar Mohri, Richard Nock, Ayfer Özgür, Rasmus Pagh, Mariana Raykova, Hang Qi, Daniel Ramage, Ramesh Raskar, Dawn Song, Weikang Song, Sebastian U. Stich, Ziteng Sun, Ananda Theertha Suresh, Florian Tramèr, Praneeth Vepakomma, Jianyu Wang, Li Xiong, Zheng Xu, Qiang Yang, Felix X. Yu, Han Yu, and Sen Zhao. Advances and open problems in federated learning. *Foundations and Trends in Machine Learning 14(1–2)*, 2021.

Sai Praneeth Karimireddy, Satyen Kale, Mehryar Mohri, Sashank J. Reddi, Sebastian U. Stich, and Ananda Theertha Suresh. Scaffold: Stochastic controlled averaging for federated learning. In *ICML 2020*, 2020.

Tian Li, Anit Kumar Sahu, Ameet Talwalkar, and Virginia Smith. Federated learning: Challenges, methods, and future directions. *IEEE Signal Processing Magazine 37(3)*, 2020a.

Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. Federated optimization in heterogeneous networks. In *MLSys 2020*, 2020b.

Yuan Liang, Ruobin Zhong, Haoming Xu, Chen Jiang, Yi Zhong, Runnan Fang, Jia-Chen Gu, Shumin Deng, Yunzhi Yao, Mengru Wang, Shuofei Qiao, Yida Xue, Xin Xu, Tongtong Wu, Kun Wang, Yang Liu, Zhen Bi, Jungang Lou, Yuchen Eleanor Jiang, Hangcheng Zhu, Gang Yu, Haiwen Hong, Longtao Huang, Hui Xue, Chenxi Wang, Yijun Wang, Zifei Shan, Xi Chen, Zhaopeng Tu, Feiyu Xiong, Xin Xie, Peng Zhang, Zhengke Gui, Lei Liang, Jun Zhou, Chiyu Wu, Jin Shang, Yu Gong, Junyu Lin, Changliang Xu, Hongjie Deng, Wen Zhang, Keyan Ding, Qiang Zhang, Fei Huang, Ningyu Zhang, Jeff Z. Pan, Guilin Qi, Haofen Wang, and Huajun Chen. Skillnet: Create, evaluate, and connect ai skills. *arXiv preprint arXiv:2603.04448*, 2026.

Tao Lin, Lingjing Kong, Sebastian U. Stich, and Martin Jaggi. Ensemble distillation for robust model fusion in federated learning. In *NeurIPS 2020*, 2020.

Xingyan Liu, Xiyue Luo, Linyu Li, Ganghong Huang, Jianfeng Liu, and Honglin Qiao. Skillforge: Forging domain-specific, self-evolving agent skills in cloud technical support. In *Proceedings of the 49th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2026. doi: 10.1145/3805712.3808466.

Ziyu Ma, Shidong Yang, Yuxiang Ji, Xucong Wang, Yong Wang, Yiming Hu, Tongwen Huang, and Xiangxiang Chu. Skillclaw: Let skills evolve collectively with agentic evolver. *arXiv preprint arXiv:2604.08377*, 2026.

H. Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *AISTATS 2017*, 2017.

Jingwei Ni, Yihao Liu, Xinpeng Liu, Yutao Sun, Mengyu Zhou, Pengyu Cheng, Dexin Wang, Erchao Zhao, Xiaoxi Jiang, and Guanjun Jiang. Trace2skill: Distill trajectory-local lessons into transferable agent skills. *arXiv preprint arXiv:2603.25158*, 2026.

Takayuki Nishio and Ryo Yonetani. Client selection for federated learning with heterogeneous resources in mobile edge. In *IEEE ICC 2019*, 2019.

Siru Ouyang, Jun Yan, Yanfei Chen, Rujun Han, Zifeng Wang, Bhavana Dalvi Mishra, Rui Meng, Chun-Liang Li, Yizhu Jiao, Kaiwen Zha, Maohao Shen, Vishy Tirumalashetty, George Lee, Jiawei Han, Tomas Pfister, and Chen-Yu Lee. Skillos: Learning skill curation for self-evolving agents. *arXiv preprint arXiv:2605.06614*, 2026.

Shuaike Shen, Wenduo Cheng, Mingqian Ma, Alistair Turcan, Martin Jinye Zhang, and Jian Ma. Skillfoundry: Building self-evolving agent skill libraries from heterogeneous scientific resources. *arXiv preprint arXiv:2604.03964*, 2026.

Yaorui Shi, Yuxin Chen, Zhengxi Lu, Yuchun Miao, Shugui Liu, Qi GU, Xunliang Cai, Xiang Wang, and An Zhang. Skill1: Unified evolution of skill-augmented agents via reinforcement learning. *arXiv preprint arXiv:2605.06130*, 2026.

Shuzheng Si, Haozhe Zhao, Yu Lei, Qingyi Wang, Dingwei Chen, Zhitong Wang, Zhenhailong Wang, Kangyang Luo, Zheng Wang, Gang Chen, Fanchao Qi, Minjia Zhang, and Maosong Sun. From context to skills: Can language models learn from context skillfully? *arXiv preprint arXiv:2604.27660*, 2026.

Yu Tian, Jiawei Chen, Lifan Zheng, Mingxiang Tao, Xinyi Zeng, Zhaoxia Yin, Hang Su, and Xian Sun. Skills-coach: A self-evolving skill optimizer via training-free grpo. *arXiv preprint arXiv:2604.27488*, 2026.

Songjun Tu, Chengdong Xu, Qichao Zhang, Yaocheng Zhang, Xiangyuan Lan, Linjing Li, Dong Li, and Dongbin Zhao. Dynamic dual-granularity skill bank for agentic rl. *arXiv preprint arXiv:2603.28716*, 2026.

Chenxi Wang, Zhuoyun Yu, Xin Xie, Wuguannan Yao, Runnan Fang, Shuofei Qiao, Kexin Cao, Guozhou Zheng, Xiang Qi, Peng Zhang, and Shumin Deng. Skillx: Automatically constructing skill knowledge bases for agents. *arXiv preprint arXiv:2604.04804*, 2026.

Jianyu Wang, Qinghua Liu, Hao Liang, Gauri Joshi, and H. Vincent Poor. Tackling the objective inconsistency problem in heterogeneous federated optimization. In *NeurIPS 2020*, 2020.

Jinyang Wu, Shuo Yang, Changpeng Yang, Yuhao Shen, Shuai Zhang, Zhengqi Wen, and Jianhua Tao. Spark: Strategic policy-aware exploration via dynamic branching for long-horizon agentic learning. *arXiv preprint arXiv:2601.20209*, 2026.

Peng Xia, Jianwen Chen, Hanyang Wang, Jiaqi Liu, Kaide Zeng, Yu Wang, Siwei Han, Yiyang Zhou, Xujiang Zhao, Haifeng Chen, Zeyu Zheng, Cihang Xie, and Huaxiu Yao. Skillrl: Evolving agents via recursive skill-augmented reinforcement learning. *arXiv preprint arXiv:2602.08234*, 2026.

Yuan Xiong, Ziqi Miao, Qian Chen, Lijun Li, Yequan Wang, Shizhu He, Jun Zhao, and Kang Liu. Skillpyramid: A hierarchical skill consolidation framework for self-evolving agents. *arXiv preprint arXiv:2606.03692*, 2026.

Jingbo Yang, Guanyu Yao, Yang Zhang, Ramana Rao Kompella, Gaowen Liu, and Shiyu Chang. FederatedSkill: Federated learning for agentic skill evolution. *arXiv preprint arXiv:2606.03143*, 2026a.

Yifan Yang, Ziyang Gong, Weiquan Huang, Qihao Yang, Ziwei Zhou, Zisu Huang, Yan Li, Xuemei Gao, Qi Dai, Bei Liu, Kai Qiu, Yuqing Yang, Dongdong Chen, Xue Yang, and Chong Luo. Skillopt: Executive strategy for self-evolving agent skills. *arXiv preprint arXiv:2605.23904*, 2026b.

Yutao Yang, Junsong Li, Qianjun Pan, Bihao Zhan, Yuxuan Cai, Lin Du, Jie Zhou, Kai Chen, Qin Chen, Xin Li, Bo Zhang, and Liang He. Autoskill: Experience-driven lifelong learning via skill self-evolution. *arXiv preprint arXiv:2603.01145*, 2026c.

Hanrong Zhang, Shicheng Fan, Henry Peng Zou, Yankai Chen, Zhenting Wang, Jiayu Zhou, Chengze Li, Wei-Chieh Huang, Yifei Yao, Kening Zheng, Xue Liu, Xiaoxiao Li, and Philip S. Yu. Coevoskills: Self-evolving agent skills via co-evolutionary verification. *arXiv preprint arXiv:2604.01687*, 2026.

Yue Zhao, Meng Li, Liangzhen Lai, Naveen Suda, Damon Civin, and Vikas Chandra. Federated learning with non-iid data. *arXiv preprint arXiv:1806.00582*, 2018.

Huichi Zhou, Siyuan Guo, Anjie Liu, Zhongwei Yu, Ziqin Gong, Bowen Zhao, Zhixun Chen, Menglong Zhang, Yihang Chen, Jinsong Li, Runyu Yang, Qiangbin Liu, Xinlei Yu, Jianmin Zhou, Na Wang, Chunyang Sun, and Jun Wang. Memento-skills: Let agents design agents. *arXiv preprint arXiv:2603.18743*, 2026.

APPENDIX

The appendix gives the prompts of the protocol, that is the executor, the optimizer that proposes edits, the merge step, and the two privacy stages (L1 sanitization and L2 private-value extraction, §3.5). Each is shown as its opening lines followed by a summary of the body, then an ellipsis. The experience digest and the commit packaging need no prompt at all: both are derived by fixed rules from recorded step records (§3.4). The full text of each template will accompany the planned framework release (see the Reproducibility Statement).

A.1 Executor prompt (adapted from the SkillOpt prompt templates). Opening:

> You are solving one task instance from {dataset}. The following skill document is the guidance you must
>
> follow, and nothing else in this prompt overrides it. . . .
>
> (Skill document)
>
> . . . (the currently published document d_{r-1}) . . .
>
> (Task instance)
>
> . . .

Body (summarised): The instance is inserted verbatim; the model is told to reason before answering, to emit exactly one answer block, and to use the answer format the dataset's evaluator expects; the decoding budget and the turn limit are pinned here. . . .

A.2 Optimizer prompt (adapted from the SkillOpt prompt templates). Opening:

> You are improving a shared skill document using the evidence of one private client. Below are the
>
> current document, the executor's trajectories on this client's slice, and the experience accumulated so
>
> far. . . .
>
> (Current document)
>
> . . . (Trajectories on this client's slice) . . . (Experience) . . .
>
> . . .

Body (summarised): The model is asked to diagnose why the executor failed on this slice, to propose a bounded set of edits over named rules (append / insert / replace), to justify each edit by behavioural evidence rather than by gold answers, and to return one structured patch; the per-round edit budget and the document's size ceiling are pinned here. . . .

A.3 Merge prompt (our commit protocol). Opening:

> You are the merge step of a federated skill evolution. You receive the currently published document and a
>
> set of commits, each of which packages a patch (edits over named rules) together with an
>
> experience digest (what the round's trajectories taught that client). . . .

Body (summarised): The prompt fixes the disposition taxonomy that must be applied to every submitted edit, namely keep, deduplicate, promote to a shared rule, and reject as invalid, together with the lossless-first, full-coverage rule (an edit is dropped only when it duplicates an already-kept edit or is structurally invalid), the requirement that the published document be one that a participant actually proposed (no averaging, no free rewriting of client text), and the requirement to return a disposal record that accounts for every participant's contribution. The experience digests are declared to be optimizer-only context and must never be written into the document. . . .

A.4 L1 sanitizer prompt. Opening:

> You are the privacy stage of a federated skill evolution. Rewrite the payload below so that it carries no
>
> private content: remove or replace entities, references and any other identifying material. . . .
>
> (Payload)

> . . . (the commit's free-text fields) . . .

Body (summarised): The instruction enumerates the categories to be stripped and requires each removal to be replaced by a typed placeholder rather than deleted, so the payload keeps its shape and stays readable to the merge stage; it states that the structural content must survive verbatim (rule identifiers, edit operations, section headers), so the patch remains applicable after sanitization; and it asks for the payload to be re-emitted field by field with no commentary. . . .

A.5 L2 private-value extraction prompt. Opening:

> You are given a sample of one client's private training data. Extract the literal values whose appearance in text transmitted to another party would constitute a leak: file paths, table identifiers and
>
> comparable artifacts. . . .

Body (summarised): The output is fixed as a flat list of literal strings, with no paraphrase and no generalisation, with the values grouped by kind (path, identifier, other), and the instruction prefers a small, high-precision set over an exhaustive one, because the set is used as a rejection test. The resulting set is frozen and serves as the offline audit reference of §3.5: during the run, any payload whose rule text contains an element of the set is rejected and returned to L1 for a rewrite. . . .