FEDRULEMEM: FEDERATED MEMORY FOR LLM AGENTS VIA EXECUTABLE RULES

Anonymous authors
Paper under double-blind review

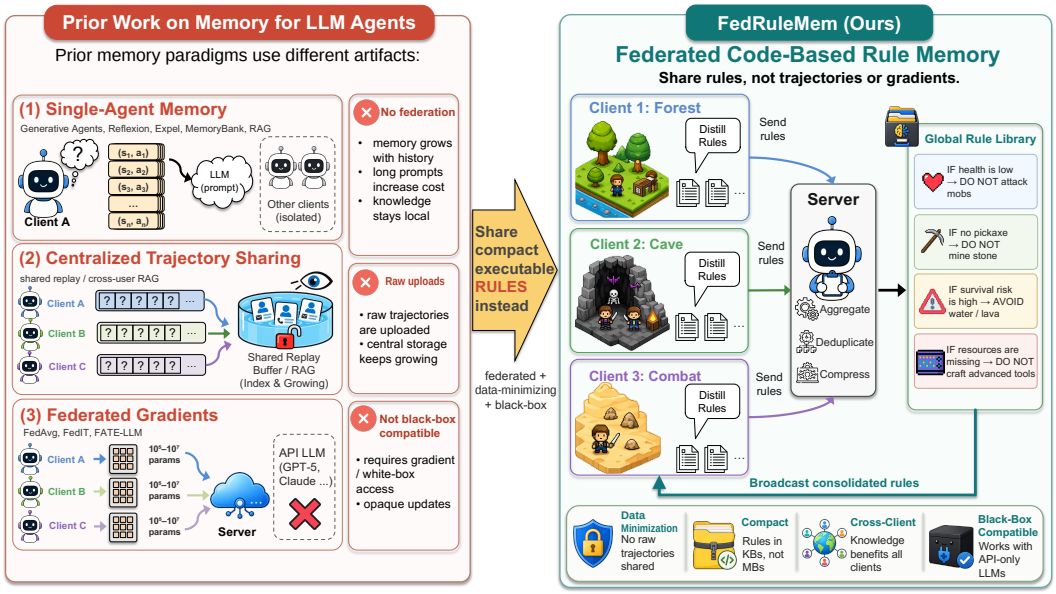


Figure 1: **FedRuleMem at a glance. Limitations of prior memory paradigms:** single-agent episodic memory (Park et al., 2023; Shinn et al., 2023; Zhao et al., 2024) cannot share knowledge across clients; centralized trajectory pooling exposes raw client trajectories to the server; gradient-based federated LLM tuning (McMahan et al., 2017; Zhang et al., 2023) requires white-box access and transmits opaque, inversion-prone weight vectors. **Our advantage:** FedRuleMem federates compact, human-auditable, executable code-based rules, achieving cross-client transfer while raw observations, actions, prompts, and task histories remain on-device, and retaining full black-box-LLM compatibility. **Workflow per round:** each client (1) interacts under an ϵ-greedy rule-guided policy, (2) synthesizes, behaviorally checks, and MDL-prunes rules locally, and (3) uploads only retained rules; the server deduplicates and broadcasts the global library, which recipients re-check against the next round's local evidence.

ABSTRACT

Large Language Model (LLM) agents typically remember experience through episodic replay or retrieval-augmented prompting. These mechanisms increase inference cost and expose detailed interaction records when shared across users. We propose **FedRuleMem**, a federated agent-memory system that shares **executable rules**: compact, stateless programs that check candidate actions and return corrective feedback. Clients synthesize rules from local failures, verify their behavior, and apply Minimum Description Length (MDL) pruning before upload. The server deduplicates retained rules and broadcasts a global library, which clients re-validate against subsequent local evidence. Raw interaction records remain local, and the acting LLM stays frozen, enabling black-box use. Across nine independent end-to-end runs on Crafter and ALFWorld, FedRuleMem reaches a Crafter Score of 13.5 ± 2.4 and ALFWorld success of $64.8\pm0.8\%$, out-

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.

performing the matched federated textual-rule control (11.4 ± 2.0 and $62.4\pm0.9\%$). On Crafter, the final library occupies 0.031 MB per client, and end-to-end token use is 61.9k per task. These results support executable rules as compact shared memory for action correction across clients. Anonymous repository: https://anonymous.4open.science/r/FedRuleMem-FE63.

1 INTRODUCTION

Large Language Model (LLM) agents are increasingly deployed as autonomous problem solvers in interactive environments, from household task completion (Shridhar et al., 2020) and open-world survival (Hafner, 2021) to web navigation (Zhou et al., 2023) and robotic manipulation (Ahn et al., 2022). A central question is *how agents should remember and reuse experience*: observing *why* an action failed and encoding that lesson can improve future decisions. Current systems store experience as episodic memory, whether as raw trajectories (Park et al., 2023), reflections (Shinn et al., 2023), or retrieved demonstrations via retrieval-augmented generation (RAG) (Lewis et al., 2020). Using these memories requires the LLM to interpret retrieved experience during action selection, and including more history increases prompt length. Sharing trajectories and detailed reflections across users can also expose client-specific states, identifiers, and behavior (Kairouz et al., 2021).

The central challenge is to share reusable action constraints across clients while limiting the propagation of incorrect rules. Existing memory paradigms address parts of this challenge. *Single-agent episodic/retrieval memory* (Park et al., 2023; Shinn et al., 2023; Zhao et al., 2024; Zhong et al., 2024) retains experience within each client. *Centralized experience pooling*, represented by our "Centralized RAG" baseline, enables sharing by uploading raw trajectories to a common retrieval index. *Federated LLM tuning* (McMahan et al., 2017; Zhang et al., 2023; Che et al., 2023; Fan et al., 2023) requires white-box access and exchanges parameter updates with known reconstruction risks (Zhu et al., 2019; Geiping et al., 2020). More closely related, federated in-context learning shares textual summaries (Wu et al., 2024), which transfer experience but still require the acting LLM to interpret the shared text. These approaches motivate a memory interface that combines cross-client sharing with executable action checks and local behavioral verification.

We propose **FedRuleMem**, which stores and shares experience as *executable rules*: small deterministic programs that check action preconditions and return structured feedback. A rule condenses failures revealing the same constraint into a reusable check, omitting episode order and most state detail. The frozen LLM chooses goals, proposes actions, and revises them using rule feedback. Building on single-agent code-as-control (Wang et al., 2023; Liang et al., 2023; Zhou et al., 2024), FedRuleMem addresses the reliability challenge introduced by federation: a syntactically valid rule can encode incorrect behavior, and aggregation can spread that error. Clients therefore verify rules against local outcomes before upload and re-validate inherited rules against subsequent evidence. The server consolidates the retained rules into a shared library. At inference, bounded action correction and a probabilistic bypass balance rule guidance with exploration.

Contributions. (i) An executable memory interface that supports cross-client action correction with frozen black-box LLMs while keeping raw interaction records local; (ii) a federated workflow that checks rules against local evidence before upload, consolidates retained rules, and re-validates inherited rules in subsequent rounds; and (iii) a nine-run evaluation on Crafter and ALFWorld showing performance gains over matched federated textual-memory controls and lower end-to-end token cost on Crafter, with analyses of scale and component effects.

2 PROBLEM SETUP

We consider a federated setting with M clients $\mathcal{I} = \{1, \dots, M\}$. Each client j has a local environment and a client-specific task distribution $\mathcal{D}_j$; in round $r \in \{1, \dots, W\}$ of W rounds, client j samples a task set $T_j^r \sim \mathcal{D}_j$ of size N_j^r and collects trajectories $\tau = ((s_0, a_0, y_0, s_1), (s_1, a_1, y_1, s_2), \dots)$ by executing an LLM agent in the environment, where s is an observation, a an action, y a scalar reward, and termination is indicated by a boolean `done`.

Rule memory. We represent memory as a *set of executable rules* R. Each rule $\rho \in R$ is a small program that evaluates a candidate action under a state. Treating s and a as dictionary-like structured objects, the rule interface is

$$\rho(s,a) \to (\texttt{feedback} \in \Sigma^*,\ \texttt{success} \in \{\textsf{True}, \textsf{False}\},\ \texttt{suggestion} \in \Sigma^*),$$

where Σ^* denotes strings and `success`=True indicates the rule is satisfied; otherwise `feedback` and `suggestion` describe the violation and corrective action. A candidate action is *valid under* R iff $\forall \rho \in R,\ \rho(s,a).\texttt{success}=\textsf{True}$. Rules are trajectory-decoupled: they encode reusable preconditions/affordances rather than state-action tuples. Each client maintains a local rule memory R_j^r; the server maintains a global memory R_{srv}^r. Both are treated as sets.

Threat model. We consider an honest-but-curious server that observes each client's upload and honest-but-curious peers that observe the broadcast library. Raw trajectories, observations, prompts, embeddings, parameters, and episode order remain local; only verified, MDL-selected rules are uploaded. This protocol provides data minimization, with neither differential privacy nor anonymity in its default form. Section 6 discusses the limitations; Appendix F details the threat analysis and an optional minimum-client-support release filter.

3 METHOD: FEDRULEMEM

FedRuleMem structures each round into *interact-and-learn*, *verify-and-prune*, and *aggregate-and-broadcast* phases (Figure 2). Let $\mathcal{S}_\psi$ be the LLM-based rule learner, $\mathcal{Q}_j^r$ the client-side quality gate, and $\mathcal{A}$ the server aggregator. Let $\mathcal{X}_j^r$ collect client j's trajectories from round r. One round is

$$\begin{aligned} \Delta R_j^r &= \mathcal{S}_\psi(\mathcal{X}_j^r), \qquad & \widetilde{R}_j^r &= \mathcal{Q}_j^r\big(R_j^{r-1} \cup \Delta R_j^r\big), \\ R_{\text{srv}}^r &= \mathcal{A}\left(\biguplus_{j=1}^{M} \widetilde{R}_j^r\right), & R_j^r &\leftarrow R_{\text{srv}}^r \quad \forall j. \end{aligned} \tag{1}$$

Only $\widetilde{R}_j^r$ is uploaded; $\mathcal{A}$ cannot access $\mathcal{X}_j^r$. Including R_j^{r-1} in the candidate set lets $\mathcal{Q}_j^r$ re-check broadcast rules against new local evidence and exclude locally rejected rules from the next upload. Rules retained by other clients may be broadcast again. Verification recurs locally, while communication scales with rule-set size. Algorithms 1 and 2 detail the procedures.

3.1 RULE-GUIDED INTERACTION WITH ϵ-GREEDY BYPASS

Let $\pi_\theta(\cdot \mid h_t)$ denote the frozen LLM policy given interaction history h_t (including current state s_t), and let $R = R_j^{r-1}$ be the latest broadcast memory. With feedback f and suggestion g, define the violation set for candidate a

$$\mathcal{V}_R(s,a) = \{(\rho, f, g) : \rho(s,a) = (f, \textsf{False}, g),\ \rho \in R\}. \tag{2}$$

The base proposal is $a_t^{(0)} \sim \pi_\theta(\cdot \mid h_t)$. Whenever $\mathcal{V}_R(s_t, a_t^{(k)}) \neq \emptyset$, PACK collects violation feedback and suggestions for regeneration:

$$a_t^{(k+1)} \sim \pi_\theta\Big(\cdot \mid h_t, a_t^{(k)}, \textsc{Pack}(\mathcal{V}_R(s_t, a_t^{(k)}))\Big), \qquad 0 \le k < L. \tag{3}$$

Let $\kappa_t = \min\big(\{k \in \{0,\dots,L\} : \mathcal{V}_R(s_t, a_t^{(k)}) = \emptyset\} \cup \{L\}\big)$ and draw $\beta_t \sim \text{Bernoulli}(\epsilon)$ with bypass probability ϵ. The executed action is

$$a_t^{\text{exec}} = \begin{cases} a_t^{(0)}, & \beta_t = 1 \quad \text{(exploration bypass)}, \\ a_t^{(\kappa_t)}, & \beta_t = 0 \quad \text{(rule-guided correction)}. \end{cases} \tag{4}$$

Equivalently, the induced policy is the mixture $\pi_{\theta,R}^{\epsilon,L} = \epsilon\pi_\theta + (1-\epsilon)\mathcal{G}_R^{(L)}[\pi_\theta]$, where $\mathcal{G}_R^{(L)}$ denotes the bounded correction operator in Equations (2)–(3). Thus ϵ prevents an incomplete rule set from collapsing exploration, while L bounds corrective inference; if no feasible proposal is found by L, the last proposal is executed. The LLM plans and proposes actions, while the rules check preconditions and supply localized feedback.

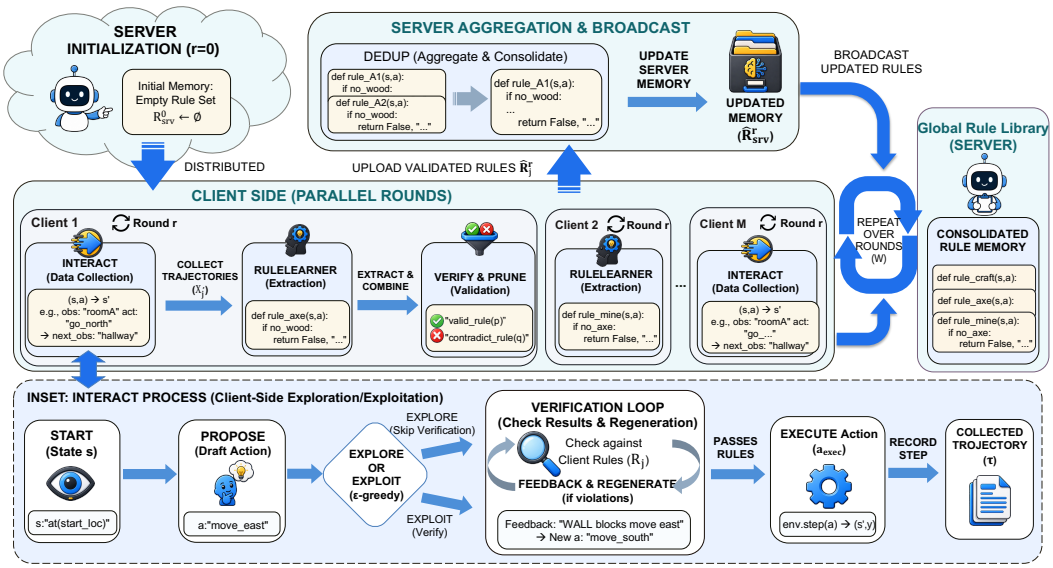


Figure 2: **FedRuleMem framework overview.** Each round has *(1) local interaction and induction*, *(2) local executability and behavioral checks followed by MDL pruning*, and *(3) exact-plus-semantic server deduplication and broadcast*. Only retained rules cross the boundary. The next candidate set contains the broadcast library, so inherited rules are checked against each recipient's new evidence before its next upload.

3.2 Local Rule Learning from Trajectories

The learner converts variable-length local trajectories into fixed-interface constraints. Let $e_{i,t} = (s_{i,t}, a_{i,t}, y_{i,t}, s_{i,t+1})$ record step t of trajectory i, including the resulting state $s_{i,t+1}$. A deterministic extractor $\chi(e_{i,t}) \in \{0, 1\}$ determines whether the attempted action succeeded from the resulting state, assigning 1 to failure and 0 to success; $\sigma_{i,t} = \textsc{Sig}(e_{i,t})$ records the failure signature. The implementation therefore assumes a structured state/action interface and some outcome evidence; Appendix B.5 tests corrupted and missing action-level feedback. This yields local negative and positive evidence

$$\mathcal{E}^r_{j,-} = \{(s_{i,t}, a_{i,t}, \sigma_{i,t}) : \chi(e_{i,t}) = 1\}, \qquad \mathcal{E}^r_{j,+} = \{(s_{i,t}, a_{i,t}) : \chi(e_{i,t}) = 0\}. \tag{5}$$

We partition $\mathcal{E}^r_{j,-}$ by action type and failure-signature similarity, obtaining clusters $\mathfrak{B}^r_j = \textsc{Partition}(\mathcal{E}^r_{j,-})$. For each $B \in \mathfrak{B}^r_j$, the LLM decoder receives B and $\Omega = (\text{state schema}, \text{rule interface}, \text{examples})$ and synthesizes

$$\rho_B = \textsc{Decode}_\psi(B; \Omega), \qquad \Delta R^r_j = \{\rho_B : B \in \mathfrak{B}^r_j,\ \textsc{WellFormed}(\rho_B) = 1\}. \tag{6}$$

WellFormed cheaply checks the AST/interface; Section 3.3 handles behavioral validation. Cluster conditioning encourages shared preconditions rather than episode identifiers. The candidate memory $R^r_{j,\text{cand}} = R^{r-1}_j \cup \Delta R^r_j$ combines inherited rules with new candidates for local re-validation. Appendices D and F give an illustrative rule and the abstraction's privacy limits.

3.3 Verification and MDL Pruning

LLM-synthesized code can be malformed, behaviorally unsafe, or redundant; federation amplifies any such error. We therefore instantiate $\mathcal{Q}^r_j$ in Equation (1) as three sequential checks.

(i) Executability. Each candidate is compiled and run in a sandbox on sampled (s, a) pairs. Let $x_\rho = 1$ iff all executions terminate and return the required tuple; candidates with $x_\rho = 0$ are discarded.

(ii) Behavioral correctness. Define the violation indicator $v_\rho(s,a) = \mathbf{1}[\rho(s,a).\texttt{success} = \textsf{False}]$. Relative to Equation (5), a rule's correctly covered failures and false rejections are

$$C_\rho = \{(s,a,\sigma) \in \mathcal{E}^r_{j,-} : v_\rho(s,a) = 1\}, \qquad U_\rho = \{(s,a) \in \mathcal{E}^r_{j,+} : v_\rho(s,a) = 1\}. \tag{7}$$

The behavioral pool is $R^r_{j,\text{beh}} = \{\rho \in R^r_{j,\text{cand}} : x_\rho = 1,\ U_\rho = \emptyset,\ C_\rho \neq \emptyset\}$. Always-accept rules have zero coverage; rules rejecting any observed positive case fail the gate. Behavioral verification checks consistency with observed outcomes, while MDL controls specificity and library size. Each rule represents one violation predicate: a failed pair governed by a *different* violation can be accepted by this rule, with unexplained failures accounted for by the residual below.

(iii) MDL pruning. We use the minimum description length principle (Rissanen, 1978) to balance rule complexity against unexplained failures. For $R \subseteq R^r_{j,\text{beh}}$, let $C(R) = \bigcup_{\rho\in R} C_\rho$. With description-length function J under a fixed prefix code, let ℓ_0 be the per-rule header cost, $\ell_{\text{code}}(\rho)$ the serialized program length, and $\ell_{\text{miss}}(e)$ the cost of encoding an unexplained failure. We select

$$\widetilde{R}^r_j = \arg\min_{R\subseteq R^r_{j,\text{beh}}} \underbrace{\sum_{\rho\in R}[\ell_0 + \ell_{\text{code}}(\rho)]}_{J(R)} + \underbrace{\sum_{e\in\mathcal{E}^r_{j,-}} \ell_{\text{miss}}(e)\,\mathbf{1}[e \notin C(R)]}_{J(\mathcal{E}^r_{j,-}|R)}. \tag{8}$$

Exact subset selection is combinatorial. Starting from $R = \emptyset$, our greedy approximation scores each remaining rule by its net reduction in description length,

$$\Delta(\rho \mid R) = \sum_{e\in C_\rho \setminus C(R)} \ell_{\text{miss}}(e) - \ell_0 - \ell_{\text{code}}(\rho), \tag{9}$$

and adds the largest positive-gain rule. With template-normalized rule costs this becomes maximum uncovered-failure coverage. The procedure stops when $\max_\rho \Delta(\rho \mid R) \le 0$, removing subsumed and weak rules while retaining a compact explanatory set (Algorithm 3).

3.4 Server Aggregation via Deduplication

Let $R^r_{\text{pool}} = \biguplus_{j=1}^M \widetilde{R}^r_j$. Stage 1 canonicalizes a rule by removing its identifier and incidental formatting: with canonicalizer $c(\cdot)$, $\rho \sim_{\text{ex}} \rho'$ iff $c(\rho) = c(\rho')$. Letting Rep choose one member of each equivalence class gives $R^r_{\text{ex}} = \text{Rep}(R^r_{\text{pool}}/\sim_{\text{ex}})$.

Stage 2 groups rules with semantically similar descriptions. Rules in R^r_{ex} are bucketed by action u; for each rule, a sentence transformer (Reimers & Gurevych, 2019) maps its comment description d_ρ to the normalized embedding $z_\rho = \phi(d_\rho)/\|\phi(d_\rho)\|_2$. Within an action bucket, semantic distance is

$$\text{Dist}(\rho,\rho') = 1 - z_\rho^\top z_{\rho'}, \qquad \text{act}(\rho) = \text{act}(\rho'). \tag{10}$$

DBSCAN (Ester et al., 1996) with radius η and minimum neighborhood size ν (including the point itself) partitions the buckets into clusters $\mathfrak{K}^r$ and noise set $\mathcal{Z}^r$. We retain the longest comment ($|d_\rho|$ counts characters) and every noise point:

$$R^r_{\text{srv}} = \mathcal{Z}^r \cup \bigcup_{K\in\mathfrak{K}^r} \left\{ \arg\max_{\rho\in K} |d_\rho| \right\}, \qquad R^r_j \leftarrow R^r_{\text{srv}}\ \ \forall j. \tag{11}$$

Exact equivalence removes byte-level redundancy, while action-conditioned clustering avoids cross-action merges. Retaining DBSCAN noise points preserves rules with no nearby counterpart. The server broadcasts the consolidated library in Equation (11), and recipients re-validate inherited rules against new local evidence in the next round. Appendix F discusses an optional client-support release filter.

4 Experiments

We evaluate FedRuleMem on two text-interface benchmarks: **Crafter** (Hafner, 2021) (open-world survival, heterogeneous client seeds/curricula) and **ALFWorld** (Shridhar et al., 2020) (household tasks, evaluation set T of 134 tasks). **RQ1 (Effectiveness):** How does executable rule memory compare with local memory, centralized pooling, and federated textual memory? **RQ2 (Efficiency):** What memory, token, and runtime costs accompany these performance gains? **RQ3 (Scaling):** How does performance change as more clients participate? **RQ4 (Component effects):** How do verification, MDL pruning, deduplication, and action-correction settings affect performance and cost?

Algorithm 1 FedRuleMem: Federated Rule Memory Consolidation

Require: Clients $\mathcal{I}=\{1,\dots,M\}$; rounds W; task counts N_j^r; rate ϵ; regen. limit L

$R_{\text{srv}}^0 \leftarrow \emptyset$; $R_j^0 \leftarrow \emptyset\ \forall j \in \mathcal{I}$
for $r = 1$ **to** W **do**
  for all $j \in \mathcal{I}$ **in parallel do**
    Sample $T_j^r \sim \mathcal{D}_j$ with $|T_j^r|=N_j^r$
    $\mathcal{X}_j^r \leftarrow \emptyset$
    for all $\xi \in T_j^r$ **do**
      $\tau \leftarrow \textsc{Interact}(\text{env}, \xi, R_j^{r-1}, \epsilon, L)$
      $\mathcal{X}_j^r \leftarrow \mathcal{X}_j^r \cup \{\tau\}$
    end for
    $\Delta R_j^r \leftarrow \textsc{RuleLearner}(\mathcal{X}_j^r)$
    $R_{j,\text{cand}}^r \leftarrow R_j^{r-1} \cup \Delta R_j^r$
    $\widetilde{R}_j^r \leftarrow \textsc{VerifyAndPrune}(R_{j,\text{cand}}^r, \mathcal{X}_j^r)$
    Upload $\widetilde{R}_j^r$ to server
  end for
  $R_{\text{srv}}^r \leftarrow \textsc{Dedup}\left(\biguplus_{j=1}^M \widetilde{R}_j^r\right)$
  Broadcast R_{srv}^r; $R_j^r \leftarrow R_{\text{srv}}^r\ \forall j$
end for
return R_{srv}^W

Algorithm 2 INTERACT: ϵ-greedy rule-guided interaction with feedback-driven regeneration

Require: env; task ξ; rules R; rate ϵ; regen. limit L

$s \leftarrow \text{env.reset}(\xi)$; $\tau \leftarrow ()$
$h \leftarrow (s)$; done $\leftarrow$ False
while not done **do**
  $a \leftarrow \textsc{LLMPropose}(h)$; $a_{\text{exec}} \leftarrow a$
  if $\textsc{Rand}() \geq \epsilon$ **then**
    for $k = 1$ **to** L **do**
      $\mathcal{V} \leftarrow \mathcal{V}_R(s, a_{\text{exec}})$
      if $\mathcal{V} = \emptyset$ **then**
        break
      end if
      $f \leftarrow \textsc{Pack}(\mathcal{V})$
      $a_{\text{exec}} \leftarrow \textsc{LLMRegen}(h, a_{\text{exec}}, f)$
    end for
  end if
  $(s', y, \text{done}) \leftarrow \text{env.step}(a_{\text{exec}})$
  Append $(s, a_{\text{exec}}, y, s')$ to τ
  Append (a_{exec}, y, s') to h; $s \leftarrow s'$
end while
return τ

Setup. We run $M = 16$ clients for $W = 10$ rounds (temperature 0.7, $\epsilon = 0.25$, $L = 5$). We use GPT-5 for Crafter and GPT-4o-mini for ALFWorld, with matched actors across methods within each benchmark. Appendix B.3 varies synthesis capability and tests three Qwen models as both actor and synthesizer. Crafter clients have fixed world seeds and curricula; ALFWorld tasks are re-randomized evenly across clients each round to avoid fitting a fixed slice. Unless stated otherwise, results report mean $\pm$ sample standard deviation over nine independent end-to-end runs varying task allocation and stochastic decoding; diagnostic rates pool events across runs. See Appendix G for details.

Baselines. We group methods by *how memory is shared* and *what crosses the client boundary*; Appendix H gives full descriptions and Table 15 lists the communicated artifacts. **(A) Local, no sharing: ReAct** (Yao et al., 2023), **Reflexion** (Shinn et al., 2023), local **RAG** (Lewis et al., 2020; Park et al., 2023), **ExpeL** (Zhao et al., 2024), and **Local Rules**, which maintains separate rule memories on all 16 clients. **(B) Multi-agent, centralized: Centralized Reflexion** and **Centralized RAG** pool reflections or trajectories, respectively. **(C) Multi-agent, artifact-sharing: Federated Reflection**, adapted **FICAL** (Wu et al., 2024), and **FedTextRule** keep trajectories local but exchange textual artifacts; FedTextRule matches our federation, MDL, and L=5 correction budget while replacing executable checks with frozen-LLM judgments. **FedRuleMem** differs in behaviorally verifying and executing code rules. Gradient-based FedAvg/FedIT is excluded because the API-only actors expose no gradients.

Evaluation metrics. Crafter metrics are **Reward** (cumulative sparse reward) and **Score** (shifted geometric mean of achievement success rates; Appendix G); ALFWorld uses final-round **Success Rate** over 134 tasks. Table 1 reports nine-run means and standard deviations, including task-family results; Appendix B gives controlled comparisons and uncertainty.

4.1 MAIN RESULTS

Table 1 compares memory methods and federated textual controls; Appendix B.1 details the closest controls and matched budgets.

RQ1. FedRuleMem achieves the highest aggregate performance on both benchmarks (Table 1). Compared with FedTextRule, it gains 2.1 Crafter Score points and 2.4 ALFWorld percentage points, supporting the executable-rule pipeline over shared textual rules. Sharing also improves on Local Rules by 0.9 Crafter Reward and 5.8 ALFWorld percentage points. The advantage persists under

Table 1: Results on Crafter and ALFWorld (mean ± sample standard deviation over nine end-to-end runs). ALFWorld reports average and task-family success (%) over $|T| = 134$ tasks.

Method	Crafter		ALFWorld Success Rate (%, ↑)						
	Reward (↑)	Score (↑)	Avg.	Pick	Clean	Heat	Cool	Examine	PickTwo
(A) Local, no sharing									
ReAct (Yao et al., 2023)	6.8 ± 1.7	7.1 ± 1.6	48.4 ± 2.1	63.9 ± 5.1	31.2 ± 5.3	59.4 ± 4.3	$\mathbf{67.7 \pm 4.6}$	30.9 ± 4.9	37.9 ± 6.0
Reflexion (Shinn et al., 2023)	6.5 ± 1.6	6.1 ± 0.9	51.4 ± 1.9	46.3 ± 4.9	59.5 ± 4.6	63.3 ± 3.8	57.7 ± 4.4	37.0 ± 4.8	35.3 ± 5.1
RAG	7.2 ± 2.1	7.5 ± 1.9	57.5 ± 1.6	76.9 ± 3.7	69.5 ± 3.3	50.2 ± 3.8	58.2 ± 4.0	42.6 ± 4.8	32.7 ± 4.3
ExpeL (Zhao et al., 2024)	7.6 ± 1.8	9.9 ± 1.8	60.4 ± 1.5	75.9 ± 3.5	69.9 ± 3.2	55.6 ± 3.6	59.8 ± 3.5	49.4 ± 4.3	39.9 ± 3.9
FedRuleMem (Local Rules Only)	8.0 ± 1.7	10.5 ± 2.9	59.0 ± 1.4	$\mathbf{77.8 \pm 2.9}$	$\mathbf{73.1 \pm 2.8}$	68.1 ± 3.1	48.1 ± 3.7	30.9 ± 4.0	37.9 ± 4.3
(B) Multi-agent, centralized									
Centralized Reflexion	7.1 ± 2.6	8.8 ± 2.3	57.4 ± 1.6	69.0 ± 3.7	55.6 ± 3.9	58.0 ± 3.8	64.0 ± 3.5	61.7 ± 3.3	30.7 ± 4.9
Centralized RAG	7.8 ± 2.3	9.4 ± 2.1	61.8 ± 1.1	66.7 ± 2.1	50.5 ± 2.8	$\mathbf{80.2 \pm 2.3}$	$\mathbf{67.7 \pm 2.1}$	$\mathbf{62.3 \pm 2.4}$	42.5 ± 3.9
(C) Multi-agent, federated									
Federated Reflection	7.6 ± 2.2	10.2 ± 2.1	60.7 ± 1.7	69.9 ± 4.0	59.5 ± 4.3	61.4 ± 3.4	64.6 ± 3.5	$\mathbf{62.3 \pm 4.6}$	42.5 ± 4.9
FICAL (adapted) (Wu et al., 2024)	8.0 ± 1.8	10.9 ± 1.6	61.9 ± 1.3	71.8 ± 2.8	62.7 ± 2.8	63.8 ± 3.1	61.9 ± 3.4	59.9 ± 3.7	45.8 ± 3.9
FedTextRule	8.5 ± 1.9	11.4 ± 2.0	62.4 ± 0.9	75.0 ± 2.1	62.7 ± 1.7	56.5 ± 2.2	61.4 ± 1.6	50.0 ± 2.8	66.7 ± 2.9
FedRuleMem (Ours)	$\mathbf{8.9 \pm 1.6}$	$\mathbf{13.5 \pm 2.4}$	$\mathbf{64.8 \pm 0.8}$	$\mathbf{77.8 \pm 2.1}$	64.9 ± 1.9	58.5 ± 2.3	64.0 ± 2.5	52.5 ± 2.9	$\mathbf{69.3 \pm 2.6}$

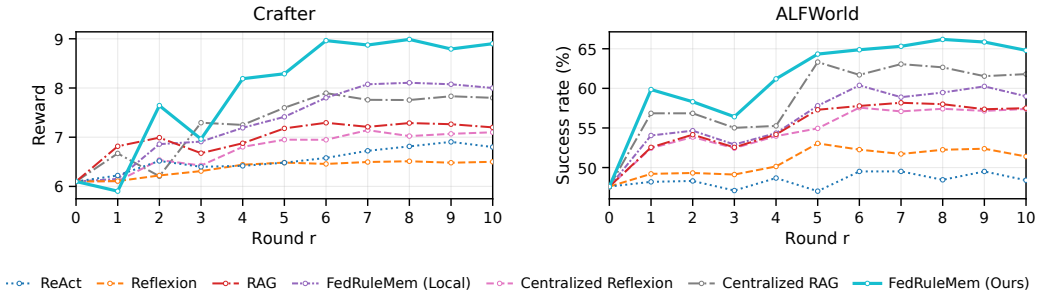


Figure 3: **Learning curves across rounds r with total budget W=10.** Mean Crafter Reward (left) and ALFWorld Success Rate (right) for seven methods. Round 0 is the initial evaluation before memory learning; round 10 results correspond to Table 1.

the matched L=5 revision budget: Centralized RAG reaches 9.8 ± 1.7 Crafter Score and $62.3 \pm 0.9\%$ ALFWorld success, remaining below FedRuleMem (Appendix Table 3).

Figure 3 shows FedRuleMem maintaining an advantage over Local Rules in later rounds on both benchmarks. At the task-family level, FedRuleMem ties for best Pick success and leads on PickTwo, while other memories lead on the remaining families (Table 1). Appendix B.4 further examines transfer to clients with no local examples from a held-out task family.

4.2 Efficiency: End-to-End Token and Runtime Cost

RQ2. FedRuleMem uses fewer tokens and less time per Crafter task while making more LLM calls than FICAL and FedTextRule (Figure 4; Appendix Table 16). It uses 61.9k tokens and 60.8 seconds, compared with 68.5k tokens and 66.7 seconds for FICAL, and 69.6k tokens and 70.2 seconds for FedTextRule. The respective call counts are 10.8, 9.1, and 10.5: call count alone does not capture the cost of memory-guided interaction. The final rule library occupies 0.031 MB per client (Appendix I).

4.3 Federation Scaling and Ablations

RQ3. Performance improves with more clients, with most gains reached by M=16 (Figure 5(a); Appendix Table 5). Increasing M from 1 to 32 raises mean Crafter Reward from 8.0 to 8.9 and ALFWorld success from 59.0% to 65.0%. Only 0.2 ALFWorld percentage points are gained beyond M=16. One possible explanation is that additional clients contribute fewer new failure patterns. Appendix Tables 11 and 12 examine the complementary effects of client-distribution shifts and conflicting rules.

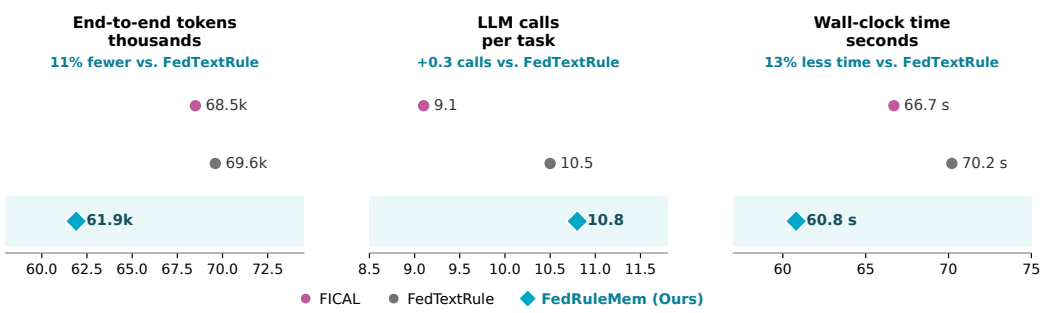


Figure 4: **End-to-end Crafter efficiency for the matched artifact-sharing methods** after $W = 10$ rounds with $M = 16$ clients. Panels report total tokens (inference plus amortized learning), LLM calls, and wall-clock time per task; lower is better. Token-component breakdowns, exact values, and additional centralized baselines are in Appendix Table 16.

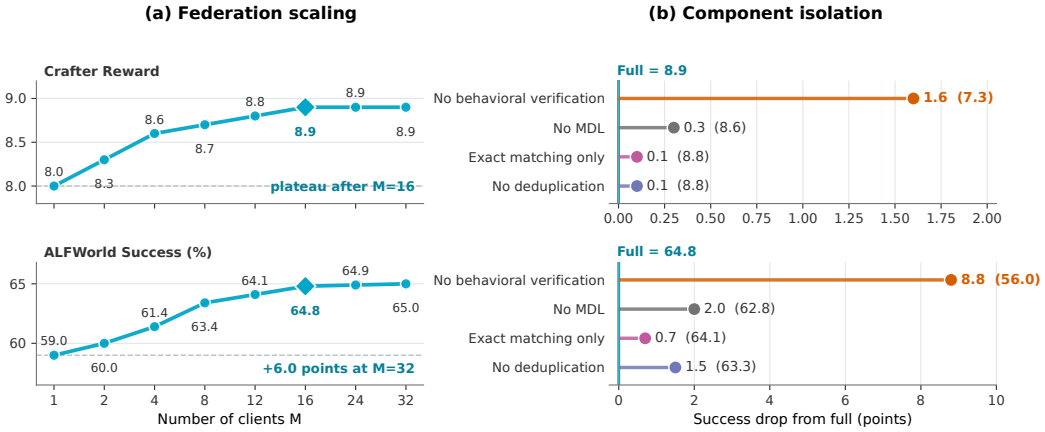


Figure 5: **Federation scaling and component isolation** at r=10 over nine complete runs. **(a)** Mean performance vs. M (M=1 uses one client); gains plateau after M=16. Standard deviations are reported in Table 5 rather than drawn as error bars. **(b)** Drops when behavioral verification, MDL, semantic deduplication, or both deduplication stages are removed. Exact values: Tables 5 and 6.

RQ4. Behavioral verification has the largest effect on performance, while MDL and deduplication mainly control library size (Figure 5(b); Appendix Table 6). Removing only behavioral verification lowers ALFWorld success from 64.8% to 56.0%. Removing only MDL nearly doubles the library from 23.6 to 46.9 rules and yields 62.8% success. Exact-only aggregation reaches 64.1% with 34.8 rules; retaining all verified duplicates reaches 63.3% with 61.4 rules. These comparisons show how rule selection and deduplication keep the library compact while preserving performance. The ϵ, L, and W sweeps in Appendix Table 7 further characterize the default operating point.

RQ4 (continued): regeneration limit L. The correction budget reaches a performance plateau at L=5 (Figure 6; Appendix Table 8). Raising L from 1 to 5 gains 0.9 Crafter Reward and 5.8 ALFWorld percentage points; L=10 adds no mean gain with 81% more regenerations. We therefore use L=5, averaging 0.21 regenerations per step. Revisions receive the state, rejected action, and memory feedback; competing memories receive the same maximum budget without extra environment transitions (Appendix Table 3).

Interpretation of the controls. The Local Rules comparison tests the value of sharing under the same client allocation, whereas FedTextRule tests the executable-rule pipeline against shared textual rules with matched federation, MDL, and correction budgets. The FedTextRule comparison therefore evaluates behavioral verification and execution jointly.

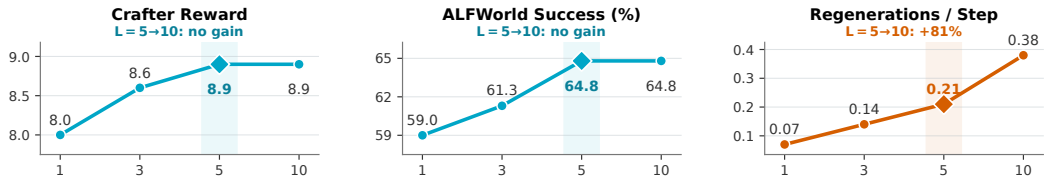


Figure 6: **Regeneration-budget quality–cost tradeoff** (M=16, final round). Default L=5 reaches the performance plateau; L=10 gives no mean gain at 81% more regenerations. Nine-run ALFWorld uncertainty and the L=0 control are in Table 7.

The component ablations distinguish the performance contribution of verification from the library-size reductions of MDL and deduplication. Together, these results support sharing a compact, behaviorally screened rule library, with verification and compression serving complementary roles.

5 Related Work

See Appendix J for an extended discussion. **Memory for LLM agents.** Episodic/retrieval-augmented memory, including Generative Agents (Park et al., 2023), Reflexion (Shinn et al., 2023), ExpeL (Zhao et al., 2024), and MemoryBank (Zhong et al., 2024), stores trajectories or textual lessons that must be interpreted in context; FedRuleMem instead executes compact constraints. **Programmatic/rule-based control.** Code-as-control systems, such as Voyager (Wang et al., 2023), Code-as-Policies (Liang et al., 2023), and WALL-E (Zhou et al., 2024; 2025), treat executable code as controllers in the *single-agent* regime; FedRuleMem extends code-as-memory to federated multi-client settings, emphasizing *constraints* (what not to do) rather than skills, with verification and MDL pruning supporting more reliable broadcast. **Federated learning and distillation.** FedAvg (McMahan et al., 2017) and FedIT (Zhang et al., 2023) require gradient access, which our frozen API-only actors do not expose. FICAL (Wu et al., 2024) federates in-context textual experience. FedRuleMem federates executable memory through a shared rule interface for frozen LLM agents. This setting makes the lifecycle of a shared check central: local verification tests its behavior, consolidation limits redundant broadcasts, and recipient-side re-validation reassesses inherited checks against the receiving client's accumulated evidence.

6 Limitations and Future Work

Our implementation assumes text-only structured state/action schemas, identifiable outcome evidence, and sandboxed deterministic checks; multimodal perception and action adapters are outside the evaluated pipeline. Constraint rules complement planning but do not represent rich procedural skills or irreducible transition stochasticity. Synthesis quality matters, especially on Crafter, and incorrect action feedback degrades performance despite local rejection. Rule execution scales linearly with library size; independently learned rules can conflict, although next-round local re-validation substantially mitigates the tested injections. Finally, keeping trajectories local provides data minimization, not formal privacy: rules may retain identifying literals or distributional fingerprints. Singleton rules remain broadcast, with no differential-privacy guarantee.

7 Conclusion

FedRuleMem represents shared agent memory as executable action checks and combines local behavioral verification with server consolidation and recipient-side re-validation. Across Crafter and ALFWorld, this pipeline improves on matched federated textual-memory controls while retaining a compact rule library and frozen black-box actors. Shared rules turn local failure experience into corrective feedback that other clients can reuse. The results support executable constraints as a reusable memory interface for federated LLM agents.

AI USE STATEMENT

Generative AI tools assisted with manuscript writing, manuscript checks, mathematical reasoning, and code implementation. The authors take responsibility for the final manuscript, code, and reported results, including content developed with this assistance.

REPRODUCIBILITY STATEMENT

Algorithms 1 and 2 specify the federated loop and rule-guided interaction; Appendix C details rule selection. Section 4 and Appendix G describe the models, benchmarks, hyperparameters, task allocation, and aggregation across runs. Appendices B.1 and H describe the matched controls, and Appendix I reports cost accounting. The anonymous code repository is linked in the abstract.

REFERENCES

Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, et al. Do as I can, not as I say: Grounding language in robotic affordances. *arXiv preprint arXiv:2204.01691*, 2022.

Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for privacy-preserving machine learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pp. 1175–1191, 2017.

Tianshi Che, Ji Liu, Yang Zhou, Jiaxiang Ren, Jiwen Zhou, Victor Sheng, Huaiyu Dai, and Dejing Dou. Federated learning of large language models with parameter-efficient prompt tuning and adaptive optimization. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 7871–7888, 2023.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD)*, pp. 226–231, 1996.

Tao Fan, Yan Kang, Guoqiang Ma, Weijing Chen, Wenbin Wei, Lixin Fan, and Qiang Yang. FATE-LLM: A industrial grade federated learning framework for large language models. *arXiv preprint arXiv:2310.10049*, 2023.

Jonas Geiping, Hartmut Bauermeister, Hannah Dröge, and Michael Moeller. Inverting gradients-how easy is it to break privacy in federated learning? *Advances in Neural Information Processing Systems*, 2020.

Danijar Hafner. Benchmarking the spectrum of agent capabilities. *arXiv preprint arXiv:2109.06780*, 2021.

Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, et al. Inner monologue: Embodied reasoning through planning with language models. In *Conference on Robot Learning*. PMLR, 2023.

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

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive NLP tasks. *Advances in Neural Information Processing Systems*, 33: 9459–9474, 2020.

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

Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. Code as policies: Language model programs for embodied control. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 9493–9500. IEEE, 2023.

Tao Lin, Lingjing Kong, Sebastian U Stich, and Martin Jaggi. Ensemble distillation for robust model fusion in federated learning. *Advances in Neural Information Processing Systems*, 33:2351–2363, 2020.

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

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. MemGPT: Towards LLMs as operating systems. *arXiv preprint arXiv:2310.08560*, 2023. URL https://arxiv.org/abs/2310.08560.

Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, pp. 1–22, 2023.

Qwen Team. Qwen3.5 open-weight model cards. Hugging Face model cards, 2026. URL https://huggingface.co/Qwen/Qwen3.5-9B. Additional checkpoints: https://huggingface.co/Qwen/Qwen3.5-27B and https://huggingface.co/Qwen/Qwen3.5-35B-A3B.

Nils Reimers and Iryna Gurevych. Sentence-BERT: Sentence embeddings using siamese BERT-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 3982–3992, 2019.

Jorma Rissanen. Modeling by shortest data description. *Automatica*, 14(5):465–471, 1978.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems*, 36:68539–68551, 2023.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36:8634–8652, 2023.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. ALFWorld: Aligning text and embodied environments for interactive learning. *arXiv preprint arXiv:2010.03768*, 2020.

Dan Wang, Siping Shi, Yifei Zhu, and Zhu Han. Federated analytics: Opportunities and challenges. *IEEE Network*, 36(1):151–158, 2022.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

Panlong Wu, Kangshuo Li, Junbao Nan, and Fangxin Wang. Federated in-context LLM agent learning. *arXiv preprint arXiv:2412.08054*, 2024.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *ICLR*, 2023.

Jianyi Zhang, Saeed Vahidian, Martin Kuo, Chunyuan Li, Ruiyi Zhang, Tong Yu, Yufan Zhou, Guoyin Wang, and Yiran Chen. Towards building the federated GPT: Federated instruction tuning. *arXiv preprint arXiv:2305.05644*, 2023. URL https://arxiv.org/abs/2305.05644.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. ExpeL: LLM agents are experiential learners. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38 (17):19632–19642, 2024.

Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. MemoryBank: Enhancing large language models with long-term memory. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(17):19724–19731, 2024.

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, et al. WebArena: A realistic web environment for building autonomous agents. *arXiv preprint arXiv:2307.13854*, 2023.

Siyu Zhou, Tianyi Zhou, Yijun Yang, Guodong Long, Deheng Ye, Jing Jiang, and Chengqi Zhang. WALL-E: World alignment by rule learning improves world model-based LLM agents. *arXiv preprint arXiv:2410.07484*, 2024.

Siyu Zhou, Tianyi Zhou, Yijun Yang, Guodong Long, Deheng Ye, Jing Jiang, and Chengqi Zhang. WALL-E 2.0: World alignment by NeuroSymbolic learning improves world model-based LLM agents. *arXiv preprint arXiv:2504.15785*, 2025.

Ligeng Zhu, Zhijian Liu, and Song Han. Deep leakage from gradients. *Advances in Neural Information Processing Systems*, 32, 2019.

A Additional details

This appendix reports the controlled comparisons and robustness studies supporting the main findings, followed by algorithms, a rule example, property and privacy analyses, implementation details, baseline definitions, end-to-end cost, and detailed related work.

B Additional experimental results

Unless noted, every performance entry below is the mean $\pm$ sample standard deviation over nine independent end-to-end runs of the complete pipeline; runs resample client task allocations and stochastic LLM decoding. Diagnostic rates are pooled across these runs.

B.1 Controlled memory and regeneration comparisons

Table 2 compares recent textual-memory and federated artifact-sharing controls using the same benchmark actor, task allocation, and interaction budget. FedTextRule is the closest representation control: it shares the same structured condition/feedback/suggestion fields, federated schedule, MDL gate, and L=5 correction budget, but a frozen LLM interprets natural-language rules instead of running behaviorally verified code. FedRuleMem improves Crafter Score by 2.1 points and ALFWorld success by 2.4 points over this control, supporting the effectiveness of the executable-rule pipeline.

Table 2: Recent memory and federated controls under matched actors and interaction budgets. FICAL is adapted to refresh and merge client tool-use summaries once per federated round.

Method	Memory setting	Crafter Score	ALF success (%)
ExpeL (Zhao et al., 2024)	local textual lessons	9.9 ± 1.8	60.4 ± 1.5
Federated Reflection	shared textual reflections	10.2 ± 2.1	60.7 ± 1.7
FICAL (adapted) (Wu et al., 2024)	shared tool-use summaries	10.9 ± 1.6	61.9 ± 1.3
FedTextRule	shared textual rules	11.4 ± 2.0	62.4 ± 0.9
FedRuleMem	**verified executable rules**	$\mathbf{13.5 \pm 2.4}$	$\mathbf{64.8 \pm 0.8}$

Table 3 controls for the bounded revision loop. Every "with $L{=}5$" method receives the same maximum number of pre-execution revisions using the same state, candidate action, and frozen actor; no method receives extra environment transitions. Under this matched budget, FedRuleMem resolves 78% of triggered failures and repeats the same failure in 10.8% of revisions, summarizing the corrective behavior of rule feedback under the matched revision budget.

Table 3: Matched $L{=}5$ regeneration control. Performance cells are Crafter Score / ALFWorld success (%). "Regen success" is the fraction of revisions that remove the triggering failure; "recurrence" repeats it.

Method	Default (no added loop)	Same $L{=}5$ loop	Regen success (↑)	Same-failure recurrence (↓)
Centralized Reflexion	8.8 ± 2.3 / 57.4 ± 1.6	9.1 ± 1.9 / 58.2 ± 1.4	31%	55.9%
Centralized RAG	9.4 ± 2.1 / 61.8 ± 1.1	9.8 ± 1.7 / 62.3 ± 0.9	36%	58.6%
FedTextRule	–	11.4 ± 2.0 / 62.4 ± 0.9	44%	26.7%
FedRuleMem	–	$\mathbf{13.5 \pm 2.4}$ / $\mathbf{64.8 \pm 0.8}$	**78%**	**10.8%**

B.2 Full-pipeline uncertainty, federation scale, and component isolation

Table 4: Crafter Reward and Score over nine complete runs.

Method	Reward (↑)	Score (↑)
ReAct (Yao et al., 2023)	6.8 ± 1.7	7.1 ± 1.6
Reflexion (Shinn et al., 2023)	6.5 ± 1.6	6.1 ± 0.9
RAG	7.2 ± 2.1	7.5 ± 1.9
Local Rules	8.0 ± 1.7	10.5 ± 2.9
Centralized Reflexion	7.1 ± 2.6	8.8 ± 2.3
Centralized RAG	7.8 ± 2.3	9.4 ± 2.1
FedRuleMem	$\mathbf{8.9 \pm 1.6}$	$\mathbf{13.5 \pm 2.4}$

The client-count sweep in Table 5 shows performance gains with diminishing returns: ALFWorld improves by 6.0 points from $M{=}1$ to 32, but only 0.2 points beyond $M{=}16$.

Table 5: Federation-size sweep at $r{=}10$. The $M{=}1$ setting uses one client; the Local Rules baseline maintains 16 isolated client memories.

M clients	1	2	4	8	12	16	24	32
Crafter Reward	8.0 ± 1.7	8.3 ± 1.3	8.6 ± 1.8	8.7 ± 1.5	8.8 ± 2.1	8.9 ± 1.6	8.9 ± 2.3	8.9 ± 1.9
ALF success (%)	59.0 ± 1.4	60.0 ± 1.8	61.4 ± 1.2	63.4 ± 0.9	64.1 ± 1.1	64.8 ± 0.8	64.9 ± 0.6	65.0 ± 0.7

Table 6 removes behavioral verification and MDL separately, and also separates exact matching from semantic clustering. Behavioral checking is the main correctness gate; MDL and deduplication primarily reduce weak, overly specific, or redundant rules.

B.3 Actor and synthesizer sensitivity

Table 9 fixes the benchmark actors and varies only the rule synthesizer. Verification rate and downstream performance both depend on synthesis capability, especially for Crafter, but the Qwen3.5-27B synthesizer approaches the main configuration. Table 10 then uses each Qwen3.5 checkpoint (Qwen Team, 2026) for both acting and synthesis; FedRuleMem improves the identical ReAct backbone in all six model–task combinations.

B.4 Client heterogeneity, unseen-task transfer, and conflicts

We vary ALFWorld task-family proportions while preserving the shared state/action schema. Smaller Dirichlet α produces stronger non-IID partitions; the hardest split assigns distinct task families to clients. FedRuleMem drops 4.0 points from balanced to task-family partitioning, versus 9.3 for isolated Local Rules, and receives a larger benefit from complementary clients.

Table 6: Separate component ablations at M=16, r=10. "Rules" is final broadcast-library size.

Variant	Crafter Reward	ALF success (%)	Rules
Full	$\mathbf{8.9 \pm 1.6}$	$\mathbf{64.8 \pm 0.8}$	$\mathbf{23.6 \pm 1.9}$
w/o behavioral verification	7.3 ± 2.5	56.0 ± 2.2	31.7 ± 2.3
w/o MDL	8.6 ± 1.9	62.8 ± 1.4	46.9 ± 2.5
Exact matching only	8.8 ± 1.5	64.1 ± 1.0	34.8 ± 2.4
Neither exact nor semantic deduplication	8.8 ± 2.0	63.3 ± 1.8	61.4 ± 2.5

Table 7: ALFWorld hyperparameter and total-round-budget sweeps. The default is $(\epsilon, L, M, W) = (0.25, 5, 16, 10)$. Each W entry reports final performance under that total budget; Figure 3 instead tracks intermediate rounds r within the fixed W=10 budget.

Variable	Tested values	ALF success (%), mean $\pm$ std
ϵ	0/0.1/0.25/0.5/0.75/1	$60.9 \pm 1.8/63.7 \pm 0.9/\mathbf{64.8 \pm 0.8}/63.8 \pm 1.3/61.4 \pm 1.7/48.6 \pm 2.5$
L	0/1/3/5/10	$48.6 \pm 2.5/59.0 \pm 1.9/61.3 \pm 1.4/\mathbf{64.8 \pm 0.8}/64.8 \pm 0.8$
Round budget W	1/3/5/7/10/15	$51.2 \pm 2.4/56.8 \pm 2.0/60.5 \pm 1.6/62.9 \pm 1.0/\mathbf{64.8 \pm 0.8}/65.1 \pm 0.7$

Table 8: Values underlying Figure 6. Regeneration counts are per environment step.

L	Crafter Reward	ALF success (%)	Avg. regenerations
1	8.0	59.0	0.07
3	8.6	61.3	0.14
5	**8.9**	**64.8**	0.21
10	8.9	64.8	0.38

Table 9: Fixed acting model; only the rule synthesizer changes. "Paper" uses GPT-5 for Crafter and GPT-4o-mini for ALFWorld.

Synthesizer	Crafter verified (%)	Crafter Score	ALF verified (%)	ALF success (%)
Qwen3.5-9B	58.7	11.6 ± 1.7	73.9	59.8 ± 1.6
Qwen3.5-27B	76.8	12.9 ± 1.3	87.2	63.4 ± 0.7
Qwen3.5-35B-A3B	73.4	12.6 ± 1.5	84.3	62.8 ± 1.0
Paper configuration	**81.6**	$\mathbf{13.5 \pm 2.4}$	**90.1**	$\mathbf{64.8 \pm 0.8}$

Table 10: Each open model used for both acting and synthesis. Cells show ReAct $\rightarrow$ FedRuleMem with the identical backbone.

Backbone	Crafter Score	ALF success (%)
Qwen3.5-9B	$5.1 \pm 1.4 \rightarrow \mathbf{8.7 \pm 1.8}$	$42.6 \pm 2.3 \rightarrow \mathbf{55.4 \pm 1.7}$
Qwen3.5-27B	$6.2 \pm 1.2 \rightarrow \mathbf{11.8 \pm 1.6}$	$47.0 \pm 1.5 \rightarrow \mathbf{62.1 \pm 0.8}$
Qwen3.5-35B-A3B	$5.9 \pm 1.9 \rightarrow \mathbf{11.2 \pm 2.2}$	$46.1 \pm 1.8 \rightarrow \mathbf{60.9 \pm 1.2}$

Table 11: ALFWorld success (%) under client distribution shift.

Client partition	Local Rules	FICAL (adapted)	**FedRuleMem**
Balanced random	59.0 ± 1.4	61.9 ± 1.3	$\mathbf{64.8 \pm 0.8}$
Dirichlet ($\alpha = 0.5$)	57.3 ± 1.8	60.4 ± 1.6	$\mathbf{63.6 \pm 1.1}$
Dirichlet ($\alpha = 0.1$)	52.8 ± 2.4	59.1 ± 1.7	$\mathbf{62.9 \pm 1.5}$
Task-family split	49.7 ± 2.5	56.9 ± 2.1	$\mathbf{60.8 \pm 1.8}$

For a client with no local examples from one held-out task family, Local Rules obtains $44.2 \pm 2.4\%$ success, whereas rules learned only by other clients raise success to $58.6 \pm 1.6\%$. This targeted 14.4-point result directly measures cross-client transfer. Conversely, Table 12 injects behaviorally conflicting rules into the broadcast library. Re-checking inherited rules at the next local gate rejects most injected conflicts and substantially limits the performance loss.

Table 12: Robustness to injected cross-client conflicts (ALFWorld success, %).

Injected conflicts	0%	10%	20%	30%
No recipient-side re-validation	64.8 ± 0.8	62.7 ± 1.8	58.6 ± 2.2	50.4 ± 2.5
FedRuleMem	$\mathbf{64.8 \pm 0.8}$	$\mathbf{64.3 \pm 0.9}$	$\mathbf{63.3 \pm 1.2}$	$\mathbf{61.2 \pm 1.6}$
Rejected at next local gate	–	81.7%	89.4%	92.1%

B.5 Failure-signal and interface robustness

Crafter contains sparse and delayed achievement outcomes. ALFWorld provides terminal success and an action-level execution message (e.g., `Nothing happens`); we flip or remove only this action-level message while retaining terminal outcomes and state transitions. Table 13 shows that verification rejects an increasing share of inconsistent candidates as noise rises. Missing messages are less harmful than incorrect ones because state transitions remain available to determine action outcomes.

Table 13: ALFWorld robustness to noisy or missing action-level feedback.

Training signal	Native	10% noise	20% noise	30% noise	No action feedback
Success (%)	64.8 ± 0.8	64.0 ± 1.1	62.2 ± 1.6	58.3 ± 2.4	62.4 ± 1.5
Candidate rejection	34%	42%	57%	76%	37%

For stochastic transitions, behavioral checking can retain preconditions that remain consistent across positive and negative cases, and the ϵ bypass preserves exploration. Section 6 discusses the structured interface and its expressiveness.

C Algorithms

Algorithm 3 Greedy MDL-based rule pruning (used inside VerifyAndPrune)

Require: Behaviorally valid pool $R^r_{j,\text{beh}}$; failure instances $\mathcal{E}^r_{j,-}$

$R^\star \leftarrow \emptyset$; $R_{\text{remaining}} \leftarrow R^r_{j,\text{beh}}$
while $R_{\text{remaining}} \neq \emptyset$ **do**
  $\Delta_{\text{best}} \leftarrow -\infty$; $\rho_{\text{best}} \leftarrow$ None
  for all rules $\rho \in R_{\text{remaining}}$ **do**
    $\Delta J_{\text{complexity}} \leftarrow J(R^\star \cup \{\rho\}) - J(R^\star)$
    $\Delta J_{\text{residual}} \leftarrow J(\mathcal{E}^r_{j,-} \mid R^\star) - J(\mathcal{E}^r_{j,-} \mid R^\star \cup \{\rho\})$
    $\Delta_{\text{net}} \leftarrow \Delta J_{\text{residual}} - \Delta J_{\text{complexity}}$
    if $\Delta_{\text{net}} > \Delta_{\text{best}}$ **then**
      $\Delta_{\text{best}} \leftarrow \Delta_{\text{net}}$; $\rho_{\text{best}} \leftarrow \rho$
    end if
  end for
  if $\Delta_{\text{best}} \leq 0$ **then**
    break
  else
    $R^\star \leftarrow R^\star \cup \{\rho_{\text{best}}\}$
    $R_{\text{remaining}} \leftarrow R_{\text{remaining}} \setminus \{\rho_{\text{best}}\}$
  end if
end while
return $R^\star$

D ILLUSTRATIVE EXECUTABLE RULE

The following illustrative rule demonstrates the interface using one necessary condition for crafting a wooden pickaxe in Crafter: the inventory must contain at least one unit of wood. In this structured action representation, `action_name` identifies the action type, `args["tool_name"]` identifies the item to craft, and `inventory` maps item names to counts.

```python
# Illustrative rule: crafting a wooden pickaxe requires wood.
def check_wood_for_pickaxe(state, action):
    if action["action_name"] != "make":
        return "", True, ""
    if action["args"]["tool_name"] != "wood_pickaxe":
        return "", True, ""

    if state["inventory"].get("wood", 0) < 1:
        feedback = ("Rule violation: Crafting a wooden pickaxe "
                    "requires at least one unit of wood.")
        suggestion = "Collect wood before crafting the pickaxe."
        return feedback, False, suggestion

    return "No violation detected by this rule.", True, ""
```

The rule returns the standardized `(feedback, success, suggestion)` tuple and checks an inventory property rather than a stored trajectory. A `True` result means only that this rule detects no violation; other rules may check additional requirements, such as a nearby crafting table. This example illustrates the interface rather than reproducing a rule from an experimental run.

E PROPERTIES AND ANALYSIS

FedRuleMem has four operational characteristics. **(i) Compression.** Rule execution per candidate is $O(|R|\omega)$ and per-round upload is $O(|\widetilde{R}_j^r|\bar{b})$, where ω and $\bar{b}$ are average rule runtime and byte length; the measured final library contains 23.6 ± 1.9 rules and occupies 0.031 MB. **(ii) Candidate-coverage growth.** Before deduplication, set union retains uploaded rules. Appendix E.2 analyzes expected failure-mode coverage under explicit discovery and validity assumptions. **(iii) Data minimization.** The shared artifact omits trajectory order and most observations; raw trajectories remain local. Section 6 discusses the limits of this boundary, and Appendix F details the threat model. **(iv) Black-box compatibility.** Rules guide a frozen API-only LLM through executable checks and in-context corrective feedback, using the same rule interface across actors.

E.1 MEMORY COMPRESSION AND EFFICIENCY

We compare the per-step inference cost and per-round communication cost of rule-based memory against two baselines: *long-context retrieval* (RAG-style episodic memory) and *gradient-based federated learning* (e.g., FedAvg with LoRA adapters).

Per-step inference cost. At each decision step, the agent must consult its memory to produce or refine an action. Let $|R|$ denote the number of rules in the current memory, and let ω denote the average wall-clock time to execute a single rule $\rho(s, a)$ (a short Python function). Checking one candidate action against all rules costs $O(|R| \cdot \omega)$. With up to L action revisions, rule execution costs at most $O((L+1)|R| \cdot \omega)$ per environment step; LLM proposal and regeneration costs are accounted for separately in Appendix I. In practice, $|R|$ is small (typically 10–30 after MDL pruning) and ω is negligible (pure Python dictionary lookups), so verification completes in milliseconds. In contrast, RAG-based episodic memory retrieves P past trajectory segments into the LLM's context window, increasing the prompt length by $O(P \cdot \mu)$ tokens, where μ is the average segment length. The added context increases LLM inference work; under standard full attention, the attention computation scales quadratically with sequence length. More critically, episodic retrieval does not provide *structured* corrective feedback: it relies on the LLM to implicitly extract lessons from retrieved episodes, whereas rules provide explicit, actionable suggestions via the $(\texttt{feedback}, \texttt{suggestion})$ fields.

Per-round communication cost. Each client uploads only its verified rule set $\widetilde{R}_j^r$, which consists of short Python functions. Let $\bar{b}$ denote the average byte size of a single rule. The upload cost per client per round is $O(|\widetilde{R}_j^r| \cdot \bar{b})$. After MDL pruning, $|\widetilde{R}_j^r|$ is typically on the order of a few dozen rules ($\sim$ 10–30), each a few hundred bytes, yielding total uploads in the kilobyte range. By comparison, gradient-based FL transmits model updates of size $O(\zeta)$, where ζ is the number of trainable parameters. Even with parameter-efficient methods such as LoRA, ζ ranges from 10^5 to 10^7, and each parameter is a 16- or 32-bit float. Rule uploads are therefore orders of magnitude smaller. Furthermore, rules are human-readable and auditable, whereas gradient vectors are opaque.

Storage. MDL selection and server deduplication reduce the number and redundancy of retained rules (Section 3.4). In the evaluated Crafter configuration, the final broadcast rule library occupies 0.031 MB per client (Appendix I); the component ablations in Table 6 quantify how selection and deduplication affect library size.

E.2 Generalization and coverage

A potential benefit of aggregation is access to failure modes that one client has not observed. Let $\mathcal{F}$ be the failure-mode universe and $\mathcal{F}_j \subseteq \mathcal{F}$ the subset for which client j discovers and retains a behaviorally valid rule.

Idealized coverage model. The pre-deduplication pool satisfies $\text{Coverage}(R_{\text{pool}}^r) = \bigcup_{j=1}^{M} \mathcal{F}_j$. If discoveries are independent with common probability γ, all retained rules are correct, and deduplication replaces a cluster with a functionally equivalent representative, expected coverage is $|\mathcal{F}|\big(1-(1-\gamma)^M\big)$. This expression explains diminishing novelty as M grows. The assumptions can fail: a representative may not preserve every precondition, distribution-specific rules can conflict, and recipient-side re-validation can remove a rule. Consequently neither final coverage nor task performance is guaranteed monotone. The nine-run scale sweep in Table 5 is consistent with the idealized curve through $M{=}16$ and then saturates.

Coverage under client heterogeneity. When client distributions are different but share a state/action schema, their failure sets may overlap less and provide complementary rules. In Table 11, the gain over Local Rules grows from 5.8 points under a balanced split to 11.1 under a task-family split; Table 12 quantifies the opposing conflict risk.

E.3 Relation to federated learning

Memory artifacts vs. model parameters. Traditional FL approaches such as FedAvg (McMahan et al., 2017) aggregate local gradients or weights to update a global parametric model; in the LLM setting, this includes FedIT (Zhang et al., 2023). FedRuleMem does not update the LLM's parameters at all; it federates *symbolic memory artifacts* (rules) that execute locally and supply corrective feedback to the fixed model at inference time, making it applicable to black-box API-only LLMs where gradient access is unavailable.

Context optimization vs. parameter optimization. FedRuleMem shares with Federated Knowledge Distillation (Lin et al., 2020) and Federated Analytics (Wang et al., 2022) the use of client-derived information for collaboration. Each client executes the broadcast rules locally to check candidate actions, and the resulting violation feedback enters the frozen LLM's context for action revision. The shared memory therefore influences inference through executable checks and in-context feedback.

Non-IID distributions. Rule union preserves client-derived constraints as individual, inspectable functions. Recipients test these constraints against local evidence and exclude rejected rules from their next upload. Appendix B.4 evaluates this process under task-partition shifts and injected conflicts.

F Extended privacy analysis

Threat model. We consider an *honest-but-curious* server that follows the protocol but observes each client-labeled upload $\widetilde{R}_j^r$, and peers that observe only the broadcast R_{srv}^r. They may infer local

trajectory content, task distributions, or environment configurations. Malicious clients, side channels, compromised endpoints, and transport security are outside scope; authenticated encrypted transport is assumed. The protocol minimizes the data disclosed to these observers by sharing verified rules while retaining trajectories locally.

Data minimization through rule abstraction. The learner converts local interaction evidence into stateless functions $\rho(s, a) \mapsto (\texttt{feedback}, \texttt{success}, \texttt{suggestion})$ that encode reusable action checks. Clients upload retained functions, while the trajectories, episode order, and observations used to synthesize and check them remain local. Recipients execute these functions against their own states and candidate actions.

Data minimization via local verification. Three local steps reduce what is uploaded: executability removes malformed programs, behavioral checking removes rules inconsistent with observed positive or negative cases, and MDL penalizes long rules with little marginal failure coverage. These checks favor compact general constraints and can remove instance-specific content.

Default deduplication and optional support filtering. Default semantic deduplication preserves DBSCAN noise points (Equation (11)); a unique rule is therefore still broadcast. Deduplication is a compression and consistency mechanism, not q-anonymity. An optional release rule can require a semantic cluster to contain uploads from at least q distinct clients before broadcast. This limits peer attribution of a released rule to one client, but the server still sees the labeled uploads, semantic similarity need not imply identical sensitive content, and this filter is not enabled in our experiments.

Residual risks and limitations. Section 6 discusses identifying content in the shared artifact. The protocol also has the following limits:

- **Rule and distribution fingerprinting.** A unique rule or the set uploaded by one client may reveal its task distribution. The server can attribute uploads directly; peers may infer sources from singleton broadcasts or auxiliary knowledge.
- **No DP guarantee.** FedRuleMem does not add calibrated noise to uploads and does not satisfy (λ, δ)-differential privacy, where λ is the privacy-loss budget and δ the failure-probability allowance. Integrating DP (by noising rule-selection probabilities or applying the exponential mechanism to MDL selection) is an important direction for future work.

FedRuleMem therefore offers a smaller, inspectable disclosure surface than sharing complete trajectories in our threat model, while leaving meaningful rule-level leakage. Deployments requiring formal privacy need explicit secret scanning/redaction, client-level support or secure aggregation, and a mechanism such as differential privacy with a stated budget.

G Benchmarks, Metrics, and Implementation Details

Crafter. Crafter (Hafner, 2021) is an open-world survival benchmark with randomly generated 2D worlds requiring foraging, crafting, combat, and tool-building skills. We report (1) **Reward** (↑): episode cumulative sparse reward ($+1$ per unlocked achievement; -0.1 per HP lost; $+0.1$ per HP regained); and (2) **Score** (↑): the benchmark's shifted geometric mean over achievement success rates, emphasizing rare achievements. For H achievement types, let $\upsilon_\iota \in [0, 100]$ be the percentage of evaluated episodes that unlock achievement ι, counting each achievement at most once per episode. Within each run,

$$\text{Score} = \exp\left(\frac{1}{H}\sum_{\iota=1}^{H}\log(1+\upsilon_\iota)\right) - 1. \tag{12}$$

The logarithm is natural. The shift makes zero-success achievements contribute $\log(1) = 0$ rather than forcing the entire score to zero; the resulting score lies in $[0, 100]$. We aggregate the resulting run-level scores as described in Section 4. Each client has a fixed world seed and curriculum bias (e.g., combat-heavy or resource-scarce), creating non-IID but schema-compatible experience.

ALFWorld. ALFWorld (Shridhar et al., 2020) is a text-based household environment with six task types requiring multi-step high-level goal completion. We evaluate all $|T| = 134$ tasks each round and report the fraction completed. Table 14 gives the exact task-family composition. We count each `trial_*` directory (equivalently, each `traj_data.json`) as one task instance. The split contains 47 distinct task-configuration directories: 41 contain three trials, five contain two, and one contains a single trial. Tasks solved earlier are not carried forward. The tasks are evenly repartitioned across M=16 clients each round (8–9 each), so a client cannot improve merely by repeatedly seeing one fixed slice. The held-out-family experiment in Appendix B.4 provides the direct cross-client-transfer control.

Table 14: Composition of the ALFWorld evaluation split used in our experiments. Percentages use all 134 task instances as the denominator.

Task family	Directory prefix	Count	Share (%)
Pick & Place	`pick_and_place_simple`	24	17.91
Examine in Light	`look_at_obj_in_light`	18	13.43
Clean & Place	`pick_clean_then_place_in_recep`	31	23.13
Heat & Place	`pick_heat_then_place_in_recep`	23	17.16
Cool & Place	`pick_cool_then_place_in_recep`	21	15.67
Pick Two & Place	`pick_two_obj_and_place`	17	12.69
Total		**134**	**100.00**

Implementation. Agents use GPT-5 on Crafter and GPT-4o-mini on ALFWorld, both with temperature 0.7 and top-p 0.95. All methods within a benchmark use the identical actor; Tables 9–10 evaluate alternative synthesizers and actors. The learner groups failures by action type, batches up to 10 cases per prompt, and emits one standardized rule per pattern. We set $\epsilon = 0.25$, $L = 5$, $\eta = 0.3$, and $\nu = 2$ (`min_samples`). Rule code is parsed before sandboxed execution against the fixed schema; safe sandboxing is an implementation assumption rather than a property of synthesized code. Nine full runs were conducted on Ubuntu 20.04.6 with an Intel i5-13600KF CPU, 64 GB RAM, and one RTX 4060 Ti 16 GB GPU; a round takes approximately 45 minutes for ALFWorld and 60 minutes for Crafter, dominated by remote LLM inference. We report final-round means and sample standard deviations unless stated otherwise.

Additional efficiency metrics. We also track rule count $|R|$, memory size (KB/MB) after each round, average regeneration per environment step when verification is enabled, and token overhead (LLM tokens per episode/task) for methods that use regeneration or long-context retrieval.

Asset licenses. Crafter and ALFWorld are released under the MIT License. The evaluated Qwen3.5-9B, Qwen3.5-27B, and Qwen3.5-35B-A3B checkpoints are released under Apache-2.0 (Qwen Team, 2026); GPT-5 and GPT-4o-mini are accessed under the provider's API terms. Our experiments follow the corresponding usage terms.

H BASELINES

All methods use identical benchmark actors, action spaces, task allocations, and local interaction budgets. They differ in memory representation, local filtering, and what is shared. Table 15 summarizes the boundary.

(A) Local, no cross-client sharing. Each client's memory is fully isolated; knowledge accumulated by one agent never reaches another.

- **ReAct** (Yao et al., 2023): interleaves chain-of-thought reasoning (Wei et al., 2022) with action calls within a single episode, with no persistent memory across episodes; this is the canonical no-memory baseline.
- **Reflexion** (Shinn et al., 2023): after each failed episode, generates a natural-language reflection appended to the client's *local* memory and prepended to future prompts as a "verbal policy". Reflections never leave the client.

Table 15: Memory location and content communicated to the server.

Setting	Method	Raw trajectories	Text artifacts	Code rules	Gradients
(A) Local	ReAct	–	–	–	–
	Reflexion	–	local	–	–
	RAG	local	–	–	–
	ExpeL	–	local	–	–
	Local Rules	–	–	local	–
(B) Centralized	Centralized Reflexion	–	✓	–	–
	Centralized RAG	✓	–	–	–
(C) Artifact-sharing	Federated Reflection	–	✓	–	–
	FICAL (adapted)	–	✓	–	–
	FedTextRule	–	✓	–	–
	FedRuleMem	**–**	**–**	**✓ (verified)**	**–**

- **RAG** (Lewis et al., 2020; Park et al., 2023): each client stores its *own* past trajectories (or summarized episodes) in a local vector index and retrieves top-P items into the prompt at each step. No cross-client flow.
- **ExpeL** (Zhao et al., 2024): extracts, scores, and retrieves natural-language lessons from prior experience, with the learned lesson memory isolated per client.
- **Local Rules**: all 16 clients learn separate rule memories using the same task allocation and local interaction budgets as FedRuleMem, without upload, aggregation, or broadcast. Figure 5(a) separately varies the number of participating clients, including a single-client (M=1) setting.

(B) Multi-agent, centralized. These baselines upload detailed client-side artifacts to a common server.

- **Centralized Reflexion (pooled verbal memory)**: clients upload their reflections to the server, which concatenates them into a shared verbal corpus accessible to all clients. The shared artifact consists of textual summaries of client experience.
- **Centralized RAG (pooled trajectory retrieval)**: all clients upload raw trajectories to one retrieval index and query top-P segments with the same local-RAG hyperparameters.

(C) Multi-agent, artifact-sharing. These methods exchange derived text or code artifacts while keeping raw trajectories and model parameters local.

- **Federated Reflection**: clients upload locally consistency- and coverage-checked reflections; the server applies the same exact and semantic deduplication before broadcast.
- **FICAL (adapted)** (Wu et al., 2024): clients summarize tool-use experience, the server merges summaries once per round, and agents retrieve relevant passages while acting.
- **FedTextRule**: uses textual conditions, feedback, and suggestions with aligned federation, MDL filtering, and L=5 correction, but relies on the frozen LLM rather than executable checks.
- **FedRuleMem**: uploads only code rules that pass executability, behavioral, and MDL gates; the server deduplicates them and recipients re-check them against subsequent local evidence.

I Extended efficiency analysis

Per-client memory. FedRuleMem's broadcast global rule library occupies just 0.031 MB per client, roughly 77× smaller than Centralized RAG's 2.4 MB pooled trajectory index. Even Centralized Reflexion (0.042 MB), whose memory consists of concatenated natural-language reflections rather than raw tuples, is slightly larger than FedRuleMem's compiled rule set. The compression advantage of executable rules is substantial: many failure episodes that reveal the same precondition collapse

into a single short Python function, whereas trajectory indexes grow with the number of unique episodes stored.

Memory tokens. Memory-only prompt tokens are distinct from total inference tokens. Centralized RAG uses 41.3k memory tokens per task, compared with 3.9k for FedRuleMem and 8.0k for Centralized Reflexion. These components are included in the inference totals in Table 16; they should not be added to those totals again. The end-to-end comparison also accounts for the amortized learning cost.

End-to-end cost. Figure 4 visualizes the complete per-task cost for the three matched artifact-sharing methods. Table 16 gives the exact values and adds centralized controls. Inference includes acting, memory prompts, and regeneration; learning amortizes reflection, summary, or rule synthesis over reuse, while runtime also includes retrieval, rule execution, behavioral verification, and aggregation. Verification and MDL use no LLM calls. FedRuleMem's 10.8 calls comprise 8.0 action-generation, 0.8 amortized synthesis, and 2.0 regeneration calls. Short rule-guided prompts give the lowest total token count and wall time among these methods.

Table 16: End-to-end cost per Crafter task. Inference includes acting, memory prompts, and regeneration; learning includes amortized reflection, summarization, or synthesis. Times are wall-clock averages.

Method	Inference tokens	Learning tokens	Total tokens	LLM calls	Time (s)
Centralized Reflexion	61.3k	2.8k	64.1k	8.8	63.7
Centralized RAG	93.8k	0.6k	94.4k	8.0	78.6
Centralized RAG + $L{=}5$	101.6k	0.6k	102.2k	10.9	91.8
FICAL (adapted)	62.9k	5.6k	68.5k	9.1	66.7
FedTextRule	64.7k	4.9k	69.6k	10.5	70.2
FedRuleMem	**58.3k**	3.6k	**61.9k**	10.8	**60.8**

J Detailed related work

This section expands the compact related-work discussion of Section 5 with the full context behind each line of prior work.

J.1 Memory and experience management in LLM agents

The standard paradigm for improving LLM agent performance over time relies on *episodic memory* and RAG. Generative Agents (Park et al., 2023) retrieves observations by semantic similarity; Reflexion (Shinn et al., 2023) conditions future actions on verbal post-hoc lessons; ExpeL (Zhao et al., 2024) extracts and retrieves textual insights. MemGPT (Packer et al., 2023) manages hierarchical context, and Inner Monologue (Huang et al., 2023) injects grounded feedback into planning. In a distributed setting, detailed traces or reflections incur context overhead and can expose states or identifiers when shared. Our Federated Reflection, FICAL, and FedTextRule controls separate the benefit of compact textual sharing from executable rules.

J.2 Programmatic and rule-based control

LLMs can synthesize executable control programs (Chen et al., 2021). Voyager (Wang et al., 2023) builds a local JavaScript skill library; Code as Policies (Liang et al., 2023) grounds embodied control in programs; Toolformer (Schick et al., 2023) learns API calls; and WALL-E (Zhou et al., 2024; 2025) extracts executable symbolic knowledge from one agent's exploration. FedRuleMem extends executable memory to cross-client reuse: agents synthesize action-precondition rules, check them against local outcomes before upload, consolidate them at the server, and re-validate them against recipient-side evidence.

J.3 FEDERATED LEARNING AND KNOWLEDGE DISTILLATION

FedAvg (McMahan et al., 2017) aggregates updates to a parametric model; FedProx (Li et al., 2020) addresses non-IID optimization, while secure aggregation (Bonawitz et al., 2017) allows the server to compute an aggregate without observing individual client updates. Federated instruction tuning (Zhang et al., 2023) and parameter-efficient prompt tuning (Che et al., 2023) require gradient access; FATE-LLM (Fan et al., 2023) provides a framework for federated LLM training. FICAL (Wu et al., 2024) instead merges natural-language tool-use experience for in-context agents. Federated Analytics (Wang et al., 2022) studies collaborative analysis over decentralized data, while FedDF (Lin et al., 2020) fuses uploaded client models through server-side ensemble distillation. FedRuleMem instead exchanges executable action-precondition rules among clients with frozen actors, and differs from FICAL in behaviorally checking and executing the shared memory.