SEPM: SELF-EVOLVING PROCEDURAL MEMORY GROUNDED IN TASK GRAPHS FOR MULTI-AGENT COLLABORATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Long-horizon multi-agent collaboration requires memory that preserves both successful actions and the coordination structure underlying them. Existing memory systems support information sharing or procedural reuse, but often leave the evolution of shared procedures disconnected from the task dependencies and execution states of collaborating agents. This coupling also limits the direct transfer of procedural memory designed for individual agents to team settings. We propose Self-Evolving Procedural Memory (SEPM), a framework that uses task dependency graphs as a common representation for online coordination and procedural memory evolution. During execution, a lightweight shared blackboard records structured evidence alongside task dependencies, supporting the detection of goal, plan, and world-state divergence and evidence-based realignment. After execution, verified actions and dependencies are distilled into reusable procedures, while recurring failures inform warnings and constraints. The shared procedure pool evolves through creation, refinement, specialization, and retirement, guided by success and cost utility and constrained by evidence and safety requirements. Together, these mechanisms preserve the collaborative structure and applicability conditions needed to reuse experience across tasks. Evaluations on ALFWorld, WebArena, and OfficeBench under AutoGen and DyLAN show that SEPM achieves the highest task scores among the compared methods in all six benchmark settings. Under AutoGen, it improves task scores over the no-added-memory baseline by 29.85, 25.86, and 13.00 percentage points, respectively. The codebase is available at `https://anonymous.4open.science/r/SEPM` and will be publicly released upon acceptance.

1 INTRODUCTION

External memory enables LLM agents to reuse experience through reflections, skills, and procedural workflows (Shinn et al., 2023; Wang et al., 2023; 2025; Fang et al., 2026). A natural route from individual agents to teams is to retain these memory units and add mechanisms for sharing, retrieval, and role-specific allocation. This adaptation-first route provides a bridge from single-agent memory to multi-agent reuse. However, making a procedure available to multiple agents does not establish the conditions under which their separate actions compose into a valid collaborative execution.

In a team, a remembered step may depend on an artifact produced by another agent, a verified state change, or a completed handoff. These conditions must remain consistent as agents act and revise their plans. When memory records what to do but leaves these dependencies to be reconstructed during execution, agents may duplicate work, proceed before prerequisites are satisfied, or reuse procedures whose assumptions no longer hold. Such failures motivate treating procedural memory and coordination as a coupled design problem: the dependencies that make experience reusable are also the dependencies that must be maintained during collaboration.

Recent multi-agent memory systems organize collaborative experience through role-specific procedures, hierarchical graphs, and team-level memory (Han et al., 2025; Zhang et al., 2025a; Wu et al., 2026). We pursue a coordination-native route in which collaborative dependencies form the interface between persistent memory and ongoing execution. The design question is how one representation

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.

can preserve the conditions of successful collaboration across tasks while making those conditions available for checking and correcting the current task.

We propose Self-Evolving Procedural Memory (SEPM), which unifies multi-agent memory and coordination through task dependency graphs. Nodes represent executable steps, and directed edges encode prerequisites between them. Retrieved procedures are instantiated into task graphs linked to agent assignments and verified execution state on a shared blackboard. During execution, these dependencies support plan comparison, prerequisite checks, and the propagation of corrections to affected agents. The resulting evidence also determines what the team remembers: successful repairs revise the active procedure, improvements simplify or strengthen its executable structure, and conditional lessons are attached as warnings. Previously unseen workflows create new entries. The same dependency semantics therefore connect the procedure that guides collaboration to the evidence that revises it.

We evaluate SEPM across four benchmarks to examine how coupling procedural memory with shared collaboration state improves multi-agent performance. On the coding, research, and database scenarios of MultiAgentBench (Zhu et al., 2025), SEPM achieves the highest Task Score (TS) and Communication Score (CS) among the compared methods in each scenario, as shown in Figure 1. We evaluate SEPM on ALFWorld, WebArena, and OfficeBench under AutoGen and DyLAN. It achieves the highest task score among the compared methods in all six benchmark settings. Under AutoGen, replacing explicit dependency graphs with textual Procedural Memory while retaining procedural information and the same retrieval rules and context budgets reduces task scores by 17.16, 8.13, and 12.34 percentage points, respectively. This comparison supports the contribution of explicit dependency graphs within the proposed architecture. On ALFWorld, SEPM also achieves higher task performance than Generative Agents and G-Memory with lower total evaluation token use.

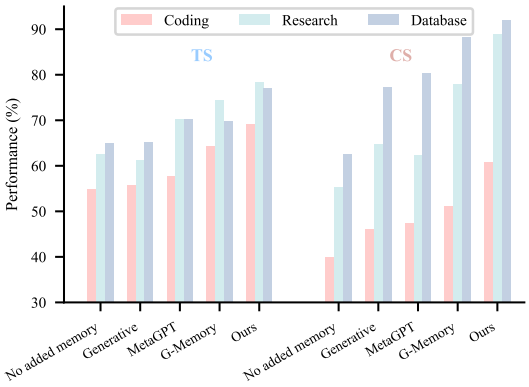


Figure 1: Task Score (TS) and Communication Score (CS) across the coding, research, and database environments of MultiAgentBench under AutoGen.

Our contributions can be summarized as follows:

(i) First, we formulate a coordination-native approach to procedural memory, using task dependencies to connect reusable experience with ongoing collaboration.

(ii) Second, we develop a shared graph representation that supports online alignment and the extraction and revision of collaborative procedures.

(iii) Third, we introduce a divergence-aware alignment mechanism that uses this shared state to detect goal, plan, and world-state inconsistencies.

2 Related Work

From single-agent procedures to multi-agent reuse. Single-agent procedural memory stores reusable skills and workflows derived from execution experience(Wang et al., 2023; 2025; Fang et al., 2026; Sarch et al., 2024; Zhang et al., 2025b). One route to multi-agent reuse transfers these units into a team and adds role-specific access or allocation(Han et al., 2025). LEGOMem adapts the single-agent Synapse and AWM baselines to task agents, while its own method constructs modular full-task and subtask memories (Han et al., 2025). This transfer route makes existing experience available to multiple agents, but the allocation of a procedure alone does not specify when a cross-agent prerequisite has been established. SEPM makes these prerequisites explicit and uses them in both collaborative execution and subsequent memory revision.

Multi-agent memory architectures and graph semantics. G-Memory organizes experience through interaction, query, and insight graphs (Zhang et al., 2025a). LEGOMem distributes pro-

cedural knowledge across orchestration and task execution (Han et al., 2025). LLMA-Mem combines episodic, procedural, and transactive memory for lifelong team learning (Wu et al., 2026). These systems establish multi-agent memory as an architectural design problem (Wang & Chen, 2025; Hao et al., 2026; Yu et al., 2026). SEPM studies a specific coupling between memory and coordination: task-graph edges express prerequisites between executable steps, and the same dependency semantics are used to instantiate procedures, compare plans, propagate corrections, and retain supported execution structure. This distinguishes the proposed use of graphs from organizing experience primarily for retrieval(Xu et al., 2025; Feng et al., 2026).

Shared workspaces and coordination. Blackboard architectures provide shared state for exchanging intermediate results and coordinating work (Hayes-Roth, 1985; Nii, 1986; Wu et al., 2024; Han & Zhang, 2025; Chen et al., 2024; Hong et al., 2024; Salemi et al., 2026). SEPM uses this architecture to associate public execution records with task nodes, agent assignments, and supporting evidence. The records describe execution outcomes, while the dependency graph identifies which later actions rely on them. Their association allows an alignment correction to inform both the current plan and the procedural guidance available to subsequent tasks.

3 Problem Formulation

We consider a team $\mathcal{A} = \{a_1, \ldots, a_n\}$ coordinated by a manager agent over a sequence of tasks $\tau_1, \tau_2, \ldots$. Each agent maintains private memory, while the team shares a task-specific blackboard and a procedural memory pool $\mathcal{P}_j$. We study how collaborative execution can improve this pool while preserving the task dependencies and execution constraints required for subsequent reuse.

Shared execution state. At step t of task τ, the shared workspace and its task graph are

$$W_\tau^t = \big(g_\tau^\star, \mathcal{G}_\tau^t, B_\tau^t, \Sigma_\tau^t\big), \qquad \mathcal{G}_\tau^t = \big(V_\tau^t, E_\tau^t, X_\tau^t, Z_\tau^t\big). \tag{1}$$

$g_\tau^\star$ specifies the objective and constraints; B_τ^t stores public execution records; and Σ_τ^t contains verified task-relevant state at the current stage of task execution. The directed acyclic graph $\mathcal{G}_\tau^t$ represents logical subtasks through nodes V_τ^t, prerequisites through edges E_τ^t, and action descriptions through X_τ^t. The associated execution metadata Z_τ^t records agent assignments and step status. Retry and replanning semantics are given in Appendix B.4.

Procedural memory. A versioned procedure is $S_i^{(v)} = (M_i^{(v)}, \Pi_i^{(v)}, G_i^{(v)}, \mathcal{W}_i^{(v)})$. The metadata $M_i^{(v)}$ contains the title, description, applicability conditions, tags, context conditions, exclusions and version identifier. The step set $\Pi_i^{(v)}$ stores the instruction for each executable step, while the directed acyclic graph $G_i^{(v)} = (V_i^{(v)}, E_i^{(v)})$ encodes their prerequisite relations. $\mathcal{W}_i^{(v)}$ contains conditional warnings together with their triggering conditions and supporting evidence.

Sequential memory revision. For each reusable procedure, the pool retains an active revision $S_i^{(v)}$. The active revision serves as the reference baseline when a new task either reveals limitations of an existing procedure or presents an opportunity to improve it. A candidate revision is assessed against this reference using the recorded outcomes, costs, and task constraints relevant to the update. We write the resulting pool transition as

$$\mathcal{P}_{j+1} = \mathcal{U}(\mathcal{P}_j, H_{\tau_j}), \tag{2}$$

where H_{τ_j} contains the task's execution records, dependency structure, verified state, and outcome. The update operator $\mathcal{U}$ creates a procedure for a new workflow, revises an existing procedure when execution supports a repair or improvement, or attaches conditional lessons to its warnings. Structural changes preserve the prerequisites and mandatory checks needed for valid collaboration. This formulation makes memory evolution an outcome of collaborative execution: the task graph both guides the current team and identifies which changes can be retained for future reuse.

4 SEPM

SEPM couples procedural reuse and collaborative execution through a shared blackboard, divergence-aware alignment, and graph-grounded memory revision. As illustrated in Figure 2,

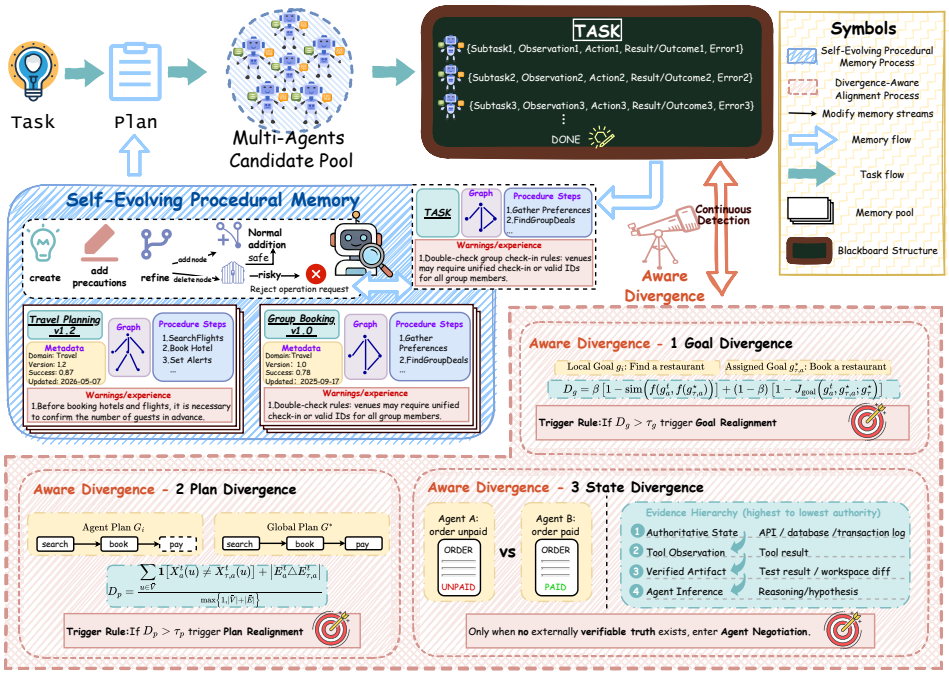


Figure 2: Overview of SEPM. Retrieved procedures instantiate collaborative task graphs. Execution records linked to task nodes support goal, plan, and state alignment. Verified outcomes and prerequisite relations support procedure creation, structural revision, and updates to conditional warnings.

retrieved procedural memories guide the decomposition and planning of target tasks. During execution, unmet prerequisites or plan deviations are identified and remediated, and these verified adjustments are then integrated into the shared procedure pool for reuse in future tasks.

4.1 Shared Blackboard for Procedural Execution

Retrieval and task instantiation. For a task query q with verified context facts F_q, retrieval considers active procedures whose context conditions are contained in F_q and whose exclusions do not intersect F_q. Let $\mathcal{C}(q, F_q)$ denote this candidate set. We rank candidates by

$$\begin{aligned} s_i(q) &= w_b(q)\,\widetilde{\mathrm{BM25}}(q, S_i) + w_v(q)\,\mathrm{sim}\big(f(q), f(S_i)\big), \\ \mathcal{R}(q) &= \mathrm{TopK}_{S_i \in \mathcal{C}(q,F_q)}\, s_i(q), \qquad K = 5, \quad w_b(q) + w_v(q) = 1. \end{aligned} \tag{3}$$

Here, f denotes the text representation function. The retrieval configuration and adaptive weighting rule are provided in Appendix B.1. The manager binds the retrieved steps to task entities and agent responsibilities while retaining applicable prerequisites and warnings.

Structured execution. During execution, agent a contributes

$$b_{a,t} = (q_{a,t}, o_{a,t}, u_{a,t}, r_{a,t}, \epsilon_{a,t}), \qquad B_\tau^{t+1} = B_\tau^t \oplus b_{a,t}, \tag{4}$$

corresponding to the assigned subtask, observation, executed action, result, and error. Each entry is linked to its graph node. Private reasoning remains local. The manager updates node status and incorporates verified observations into Σ_τ^t. These records preserve both successful actions and failures needed for subsequent procedure extraction.

4.2 Divergence-Aware Multi-Agent Alignment

For agent a, let $(g_a^t, \mathcal{G}_a^t, \Sigma_a^t)$ denote its local objective, plan, and state estimate. Its reference objective $g_{\tau,a}^\star$ and subgraph $\mathcal{G}_{\tau,a}^t$ are determined by the current assignment, including prerequisites supplied by other agents. Alignment evaluates consistency with the agent's role.

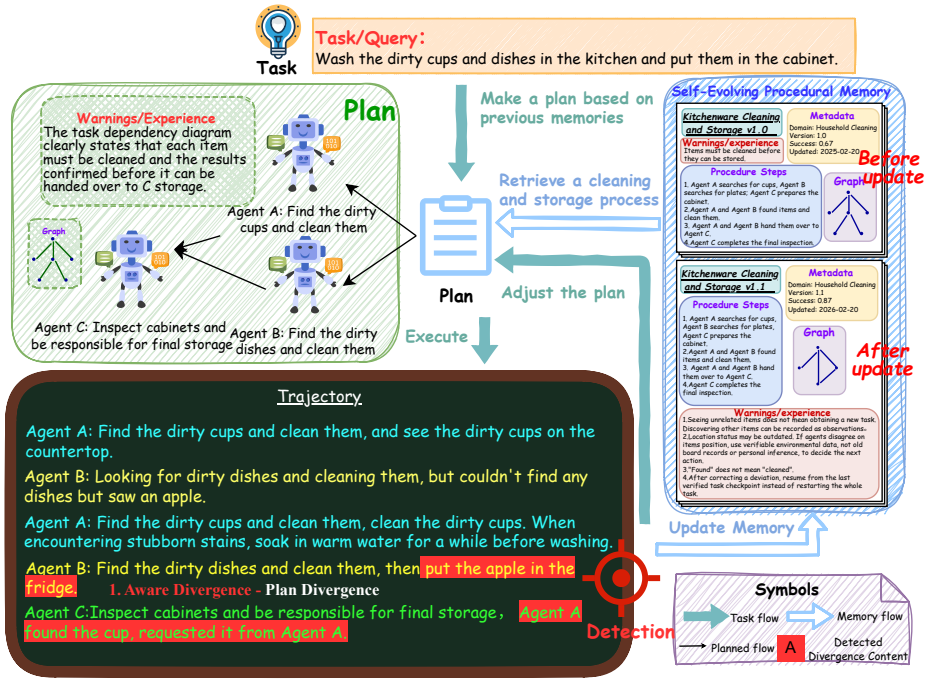


Figure 3: Illustrative cleaning-and-storage case. Agent B introduces an unrelated action, while Agent C requests a cup before cleaning has been verified. Realignment restores the assigned plan and the cleaning-before-handoff prerequisite. The revised procedure retains this dependency and adds guidance on state verification and resumption from verified checkpoints.

Goal divergence. We combine a discrepancy between goal-text representations with a task-conditioned goal-consistency score:

$$D_g = \beta \left[1 - \mathrm{sim}\big(f(g_a^t), f(g_{\tau,a}^\star)\big)\right] + (1-\beta)\left[1 - J_{\mathrm{goal}}(g_a^t, g_{\tau,a}^\star; g_\tau^\star)\right]. \tag{5}$$

$\beta \in [0,1]$ balances the two terms. The first term measures dissimilarity between the textual representations of the current and assigned goals. The second term, $J_{\mathrm{goal}} \in [0,1]$, assesses whether the intended operation is consistent with the agent's assignment and the global task constraints. This distinction matters when similar goal descriptions imply different operations, such as finding and booking a restaurant. The representation and scoring details are provided in Appendix B.2.

Plan divergence. Plan divergence measures differences between an agent's local plan and its assigned plan in both actions and dependencies. After matching corresponding nodes, let $\bar{V} = V_a^t \cup V_{\tau,a}^t$ and $\bar{E} = E_a^t \cup E_{\tau,a}^t$. We define

$$D_p = \frac{\sum_{u \in \bar{V}} \mathbf{1}\left[X_a^t(u) \neq X_{\tau,a}^t(u)\right] + \left|E_a^t \triangle E_{\tau,a}^t\right|}{\max\left\{1, |\bar{V}| + |\bar{E}|\right\}}. \tag{6}$$

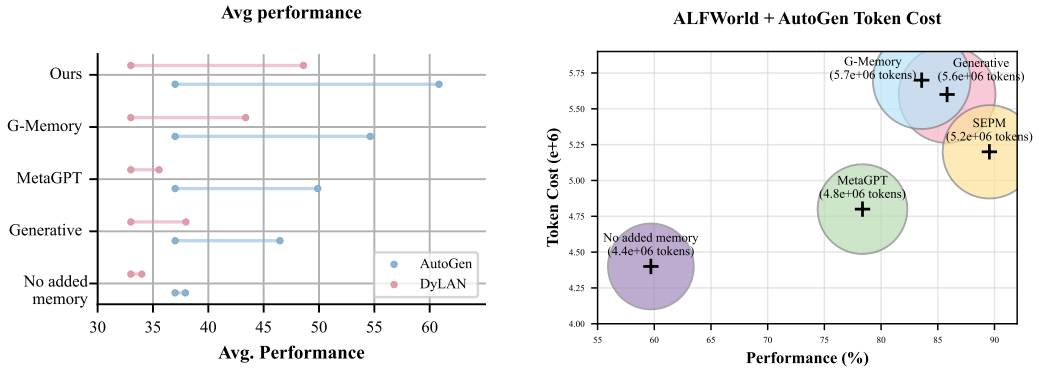
The node-action term $\sum_{u \in \bar{V}} \mathbf{1}[X_a^t(u) \neq X_{\tau,a}^t(u)]$ counts action mismatches between corresponding nodes, while the edge term $|E_a^t \triangle E_{\tau,a}^t|$ counts dependency relations that occur in only one plan. Their sum is normalized by the combined plan size, so larger D_p indicates greater plan divergence. Node matching and missing-node handling are detailed in Appendix B.2.

State divergence and realignment. For each task-relevant fact k, let $\mathcal{E}_k^+$ denote the qualified evidence reported by the agents. We define

$$\begin{aligned} C_s &= \left\{k : \exists e, e' \in \mathcal{E}_k^+,\ \mathrm{val}(e) \neq \mathrm{val}(e')\right\}, \\ z_a^t &= \mathbf{1}[D_g > \tau_g \ \vee\ D_p > \tau_p \ \vee\ C_s \neq \varnothing]. \end{aligned} \tag{7}$$

C_s contains task-relevant facts for which the agents provide conflicting qualified evidence. When $z_a^t = 1$, the manager resolves the detected goal, plan, or state conflict and propagates the correction to affected agents. Evidence qualification and conflict resolution are detailed in Appendix B.3.

Figure 3 illustrates how these signals lead to different corrections. Agent A is assigned to clean cups, Agent B to clean dishes, and Agent C to inspect the cabinet and complete storage. Agent B's



(a) Average task score by host.

(b) ALFWorld performance versus token cost.

Figure 4: Task performance and token cost. (a) Mean task scores across ALFWorld, WebArena, and OfficeBench; each segment connects the no-added-memory baseline to a memory strategy within the same host. (b) ALFWorld task scores versus total input and output tokens over the complete evaluation under AutoGen, measured in millions.

action of placing an apple in the refrigerator introduces an action outside its assigned plan. The correction redirects B to the dish-cleaning subtask. Agent C's request for a cup raises a different issue: locating the cup establishes its presence, but does not establish that cleaning is complete. The handoff therefore requires evidence of the cleaning outcome before storage proceeds. The dependency links A's verified completion to C's next action, while cabinet inspection can proceed independently. Thus, realignment concerns both the actions assigned to each agent and the evidence required for a cross-agent handoff.

4.3 Self-Evolving Procedural Memory

Extraction from verified collaboration. At task completion, let $H_\tau = (B_\tau^{T_\tau}, \mathcal{G}_\tau^{T_\tau}, \Sigma_\tau^{T_\tau}, y_\tau)$ contain the terminal records, graph, verified state, and outcome. The reusable execution structure is

$$\begin{aligned} V_\tau^+ &= \{u \in V_\tau^{T_\tau} : H_\tau \models u\}, \\ E_\tau^+ &= \{(u,w) \in E_\tau^{T_\tau} : u, w \in V_\tau^+,\ H_\tau \models (u,w)\}. \end{aligned} \tag{8}$$

$H_\tau \models u$ indicates that the recorded action and its outcome are verified. For a dependency (u, w), the required support concerns whether the outcome of u establishes a prerequisite of w; temporal precedence alone does not establish this relation. Extraction $\widehat{S}_\tau = \Phi(H_\tau, V_\tau^+, E_\tau^+)$ converts supported actions into reusable steps, retains their prerequisite relations, and associates recurring failures with conditional warnings or preconditions. Appendix B.5 describes procedure extraction and revision.

4.4 Revision Validation and Promotion

Candidate validation and promotion. Execution histories provide candidate procedures and revisions. Candidate validation screens the proposed change, while promotion determines whether it becomes an active procedure version. Let $\mathcal{T}_{S'}$ denote the recorded reproduction trials for candidate S'. Promotion requires

$$\begin{aligned} \mathrm{Accept}(S'; \mathcal{T}_{S'}) = G_{\text{safety}} \wedge G_{\text{state}} \wedge G_{\text{causal}} \\ \wedge\, G_{\text{reproduction}} \wedge [\overline{u}(S') \geq 0]. \end{aligned} \tag{9}$$

The gates check safety, verified state, causal attribution, and reproduction requirements. The mean net benefit $\overline{u}(S')$ is computed over all recorded trials, including failed trials. A candidate without recorded trials cannot be promoted. Appendix B.5 specifies the candidate checks, trial utility, and promotion summary score.

5 Experiments

We evaluate whether graph-grounded procedural memory improves collaborative task performance across two hosts. We then examine sensitivity to the joint choice of execution and memory models,

the contributions of the blackboard, procedural memory, and alignment, and the effect of replacing explicit dependency graphs with textual procedures. Token consumption is measured alongside task performance to characterize the cost of the complete evaluation.

5.1 Experimental Setup

ALFWorld evaluates embodied procedural interaction (Shridhar et al., 2021); WebArena evaluates stateful web tasks (Zhou et al., 2024); and OfficeBench evaluates workflows spanning office applications (Wang et al., 2024). We report benchmark-specific task scores using the evaluation criteria of each environment, with higher scores indicating better task performance.

The host comparison includes no added memory, Generative Agents (Park et al., 2023), MetaGPT (Hong et al., 2024), G-Memory (Zhang et al., 2025a), and SEPM under AutoGen (Wu et al., 2024) and DyLAN (Liu et al., 2024). The main planning, task-execution, and procedural memory models, as well as the goal-consistency judge, use `gpt-5-mini` in the main comparison. The sensitivity analysis keeps AutoGen, the main planning agent, and the goal-consistency judge fixed while jointly varying the task-execution and procedural memory model over twelve alternative endpoints. The component analysis progressively introduces the shared blackboard, procedural memory, and divergence-aware alignment. The final comparison reports total input and output token consumption recorded during the corresponding ALFWorld evaluations under AutoGen. Appendix C documents the evaluation scope and reporting details.

Table 1: Task performance across host multi-agent systems. Boldface marks the highest score within each benchmark setting. Arrows indicate percentage-point changes relative to the same host without added memory.

MAS	Memory	ALFWorld	WebArena	OfficeBench
AutoGen	No added memory	59.70↑0.00	12.44↑0.00	41.67↑0.00
	Generative	85.82↑26.12	15.27↑2.83	38.33↓3.34
	MetaGPT	78.36↑18.66	25.99↑13.55	45.33↑3.66
	G-Memory	83.58↑23.88	30.30↑17.86	50.00↑8.33
	SEPM	**89.55**↑29.85	**38.30**↑25.86	**54.67**↑13.00
DyLAN	No added memory	53.73↑0.00	14.90↑0.00	33.33↑0.00
	Generative	65.67↑11.94	15.27↑0.37	33.00↓0.33
	MetaGPT	49.25↓4.48	22.41↑7.51	35.00↑1.67
	G-Memory	61.94↑8.21	25.49↑10.59	42.67↑9.34
	SEPM	**68.66**↑14.93	**29.80**↑14.90	**47.33**↑14.00

We additionally evaluate the five memory configurations on the coding, research, and database environments of MultiAgentBench. Appendix C.3 specifies the evaluation configuration and scoring protocol used to assess performance across these environments.

5.2 Performance across Host Systems

SEPM achieves the highest observed task score in all six benchmark settings (Table 1). Relative to the strongest alternative in each setting, it improves ALFWorld, WebArena, and OfficeBench by 3.73, 8.00, and 4.67 percentage points under AutoGen, and by 2.99, 4.31, and 4.66 points under DyLAN. The consistent ranking across hosts supports the usefulness of the proposed memory coordination design under different orchestration mechanisms.

The margins nevertheless vary across tasks and hosts. The largest advantage appears on WebArena under AutoGen and on OfficeBench under DyLAN, indicating that the benefit depends on how the host executes a particular class of workflows. The baseline results show a related pattern: Generative Agents performs best among the alternatives on ALFWorld, whereas G-Memory leads on WebArena and OfficeBench. Thus, the comparison favors different memory designs across environments, while SEPM remains effective across all three.

Figure 4a summarizes the overall improvement: mean task score rises from 37.94% to 60.84% under AutoGen and from 33.99% to 48.60% under DyLAN. The component and representation analyses below examine how shared execution state, procedural evolution, and explicit dependencies contribute to these gains.

5.3 Sensitivity to Execution and Memory Models

We evaluate sensitivity to the joint choice of task-execution and procedural memory model. AutoGen, the main planning agent, and the goal-consistency judge remain fixed, while each listed

endpoint is used for both task execution and procedural memory management. Table 2 includes the main configuration from Table 1 as a reference, together with twelve alternative endpoints.

Performance varies substantially with the joint execution-and-memory model, with score ranges of 35.82, 28.70, and 22.00 percentage points on ALFWorld, WebArena, and OfficeBench. Within the Qwen3.5 configurations, increasing model size from 2B to 27B improves all benchmarks, whereas the 0.8B and 2B configurations perform similarly. These results suggest that the framework depends on the model's ability to execute procedural guidance and maintain useful memories, with limited gains between the two smallest configurations.

The strongest endpoint differs by benchmark. `deepseek-v4-flash-0731` achieves the highest ALFWorld score, while `gpt-5.6-terra` leads on WebArena and OfficeBench. Relative to the main configuration, the latter improves these benchmarks by 2.24, 4.93, and 13.00 points, respectively. The uneven gains indicate that model choice has different consequences across task environments. Since execution and memory management change together, this experiment measures their joint effect.

Table 2: Sensitivity to execution and memory models. The main planning agent and goal-consistency judge are fixed. AW, WA, and OB denote ALFWorld, WebArena, and OfficeBench, respectively.

Model	AW	WA	OB
qwen3.5-0.8b	58.21	14.53	45.67
qwen3.5-2b	57.46	14.78	45.67
qwen3.5-9b	67.16	16.01	49.67
qwen3.5-27b	81.34	30.42	55.67
gemma-4-12b-it	72.39	22.54	53.00
gemma-4-31b-it	77.61	28.69	54.33
deepseek-v4-flash-0731	**93.28**	38.05	64.33
qwen3.8-max	89.55	39.41	63.67
glm-5.2	85.82	37.93	60.67
kimi-k3	90.30	42.36	65.33
gpt-5-mini (main)	89.55	38.30	54.67
gpt-5.6-terra	91.79	**43.23**	**67.67**
claude-opus-5	92.54	42.49	65.33

5.4 Component Contributions

Figure 5b evaluates procedural memory and divergence-aware alignment on a shared blackboard. The blackboard provides the task dependency graph and structured execution records on which both mechanisms operate. We therefore retain this structural foundation when evaluating the contribution of either mechanism.

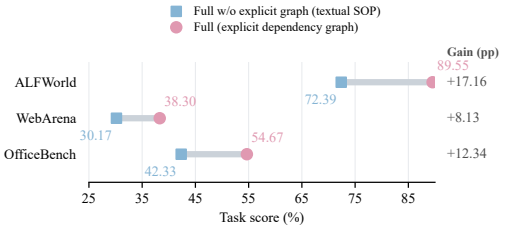
The shared blackboard improves task scores by 9.70, 15.27, and 3.33 percentage points on ALFWorld, WebArena, and OfficeBench, respectively. Its largest gain occurs on WebArena, suggesting that maintaining a common execution state is particularly useful when successive actions interact with a changing web environment.

Building self-evolving memory on this foundation yields an additional 17.17 points on ALFWorld, 6.28 on WebArena, and 0.33 on OfficeBench. The large ALFWorld gain is consistent with the value of retaining reusable action sequences and their prerequisites. On OfficeBench, however, procedural reuse alone provides little improvement over the blackboard, suggesting that access to a workflow is insufficient when its execution depends on coordinated state changes across applications.

Alignment changes this pattern. Adding it to Blackboard + memory improves OfficeBench by 9.34 points, compared with 2.98 on ALFWorld and 4.31 on WebArena. Conversely, the OfficeBench gain from memory increases from 0.33 points without alignment to 3.67 points with alignment. This pattern suggests that maintaining execution consistency helps the team realize the benefit of reusable office procedures. The full configuration performs best on all three benchmarks, with task-dependent contributions from memory evolution and online alignment.

5.5 Explicit Graphs versus Textual Procedures

We introduce Full w/o explicit graph, graph objects and adjacency relations are replaced by textual procedures that state each step's prerequisites. This intervention changes the dependency representation and its associated operations without introducing additional task information or model capabilities.Replacing explicit graphs with textual prerequisites reduces task scores by 17.16, 8.13, and 12.34 percentage points on ALFWorld, WebArena, and OfficeBench, respectively (Figure 5a). Because the control retains procedural content and prerequisite descriptions, the performance gap suggests that preserving this information in memory is insufficient to obtain the full benefit of the proposed design.



(a) Comparison with textual procedural memory.

Condition	AW	WA	OB
No added components	59.70	12.44	41.67
Blackboard only	69.40	27.71	45.00
Blackboard + memory	86.57	33.99	45.33
Blackboard + alignment	82.09	34.48	51.00
Full SEPM	**89.55**	**38.30**	**54.67**

(b) Component ablation.

Figure 5: Ablation results under AutoGen. **(a)** Full SEPM versus the textual procedural memory control. **(b)** Component configurations. AW, WA, and OB denote ALFWorld, WebArena, and OfficeBench. Memory denotes the complete cross-task procedure extraction, revision, and reuse mechanism built on the shared blackboard.

Explicit edges make the same dependency structure available to prerequisite checking, plan comparison, and correction propagation. A change to one step can therefore be related directly to the downstream steps that depend on it. The textual control requires these relations to be recovered from descriptions during the corresponding operations. The results support the combined use of explicit dependencies and graph-based execution operations for turning procedural knowledge into coordinated behavior.

5.6 Cost analysis

SEPM achieves 89.55% success using 5.2 million tokens over the 134 ALFWorld tasks (Figure 4b). Relative to no added memory, token consumption increases by approximately 18.2%, while task success improves by 29.85 percentage points. The additional computation therefore accompanies a larger fraction of completed tasks.

Compared with Generative Agents and G-Memory, SEPM improves success by 3.73 and 5.97 points while reducing total token consumption by approximately 7.1% and 8.8%, respectively. It thus achieves a better trade-off between performance and token consumption than both memory baselines in this evaluation. MetaGPT consumes fewer tokens but reaches lower success, representing a different trade-off. These results show that the additional coordination and memory-update mechanisms can coexist with lower overall token consumption than the two competing memory systems.

6 Conclusion and Limitations

Conclusion. We introduced SEPM, which connects procedural memory with multi-agent coordination through shared task dependencies. The blackboard grounds execution in a common graph, alignment maintains consistency during collaboration, and procedural memory evolves through execution-supported revisions across tasks. Procedural memory provides a large gain on ALFWorld, while OfficeBench benefits more from its combination with alignment. Under AutoGen, SEPM achieves task scores of 89.55%, 38.30%, and 54.67% on ALFWorld, WebArena, and OfficeBench, respectively, outperforming the information-matched textual control on all three benchmarks. On ALFWorld, it also uses fewer tokens than Generative Agents and G-Memory.

Limitations. The current SEPM implementation represents collaborative procedures as directed acyclic graphs of executable steps and explicit prerequisite relations. This formulation targets workflow-oriented collaboration in which task dependencies and relevant execution states can be identified during interaction. More open-ended settings may contain iterative or less explicit coordination structures. Incorporating richer temporal and dependency representations is a natural extension of the framework and could broaden its applicability to these forms of collaboration.

AI Use Statement

Generative AI assisted with prose revision, result-table formatting, visualization code, and consistency checks against the supplied manuscript, implementation, figures, and experiment records.

Ethics Statement

This work studies procedural memory and coordination for language-model agents in established benchmark environments. The experiments do not involve human participants or the collection of new personal data, and the evaluated agents are not deployed in user-facing or consequential real-world settings. The framework may inherit biases, unsafe behavior, and factual errors from its underlying language models; its memory and alignment mechanisms should not be interpreted as guarantees of safety or correctness.

Reproducibility Statement

We provide the algorithmic specification of SEPM, including its execution flow, retrieval and alignment rules, procedure extraction and verification, retry behavior, and version-update mechanism, in Appendices A–B.6. Section 5.1 and Appendix C document the evaluated host systems, model configurations, comparison conditions, memory initialization and persistence, benchmark environments, task counts, execution budgets, random-seed construction, score aggregation, and token accounting. Appendix D reports supplementary results and additional analyses. We will publicly release the implementation, benchmark adapters, experiment configurations, and evaluation scripts to support reproduction of the reported results.

References

Weize Chen, Yusheng Su, Jingwei Zuo, Cheng Yang, Chenfei Yuan, Chi-Min Chan, Heyang Yu, Yaxi Lu, Yi-Hsin Hung, Chen Qian, Yujia Qin, Xin Cong, Ruobing Xie, Zhiyuan Liu, Maosong Sun, and Jie Zhou. Agentverse: Facilitating multi-agent collaboration and exploring emergent behaviors. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun (eds.), *International Conference on Learning Representations*, volume 2024, pp. 20094–20136, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/578e65cdee35d00c708d4c64bce32971-Paper-Conference.pdf.

Runnan Fang, Yuan Liang, Xiaobin Wang, Jialong Wu, Shuofei Qiao, Pengjun Xie, Fei Huang, Huajun Chen, and Ningyu Zhang. Memp: Exploring agent procedural memory. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 17490–17502, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-395-1. doi: 10.18653/v1/2026.findings-acl.866. URL https://aclanthology.org/2026.findings-acl.866/.

Tao Feng, Chongrui Ye, Tian-Wen Luo, Jing Xu, Xueqiang Xu, Haozhen Zhang, Zhigang Hua, Yan Xie, Shuang Yang, Ge Liu, and Jiaxuan You. Expgraph: Model-agnostic experience learning with graph-structured memory for llm agents, 2026. URL https://arxiv.org/abs/2605.30712.

Bochen Han and Songmao Zhang. Exploring advanced llm multi-agent systems based on blackboard architecture, 2025. URL https://arxiv.org/abs/2507.01701.

Dongge Han, Camille Couturier, Daniel Madrigal Diaz, Xuchao Zhang, Victor Rühle, and Saravan Rajmohan. Legomem: Modular procedural memory for multi-agent llm systems for workflow automation, 2025. URL https://arxiv.org/abs/2510.04851.

Guangya Hao, Yunbo Long, and Zhuokai Zhao. Self-evolving multi-agent systems via decentralized memory, 2026. URL https://arxiv.org/abs/2605.22721.

Barbara Hayes-Roth. A blackboard architecture for control. *Artificial Intelligence*, 26(3):251–321, 1985. ISSN 0004-3702. doi: https://doi.org/10.1016/0004-3702(85)90063-3. URL https://www.sciencedirect.com/science/article/pii/0004370285900633.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Ceyao Zhang, Jinlin Wang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. Metagpt: Meta programming for a multi-agent collaborative framework. In *International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=VtmBAGCN7o.

Zijun Liu, Yanzhe Zhang, Peng Li, Yang Liu, and Diyi Yang. A dynamic LLM-powered agent network for task-oriented agent collaboration. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=XII0Wp1XA9.

H. Penny Nii. PART ONE: The blackboard model of problem solving and the evolution of blackboard architectures. *AI Magazine*, 7(2):38–53, June 1986. ISSN 0738-4602. doi: 10.1609/aimag.v7i2.537. URL https://doi.org/10.1609/aimag.v7i2.537.

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. UIST '23, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400701320. doi: 10.1145/3586183.3606763. URL https://doi.org/10.1145/3586183.3606763.

Alireza Salemi, Mihir Parmar, Palash Goyal, Yiwen Song, Jinsung Yoon, Hamed Zamani, Hamid Palangi, and Tomas Pfister. LLM-based multi-agent blackboard system for information discovery in data science, 2026. URL https://openreview.net/forum?id=egTQgf89Lm.

Gabriel Sarch, Lawrence Jang, Michael Tarr, William W. Cohen, Kenneth Marino, and Katerina Fragkiadaki. Vlm agents generate their own memories: Distilling experience into embodied programs of thought. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 75942–75985. Curran Associates, Inc., 2024. doi: 10.52202/079017-2418. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/8ac50fd0a4eeeb1f077b17bb7c5353c3-Paper-Conference.pdf.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: language agents with verbal reinforcement learning. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 8634–8652. Curran Associates, Inc., 2023. doi: 10.52202/075280-0377. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/1b44b878bb782e6954cd888628510e90-Paper-Conference.pdf.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. ALFWorld: aligning text and embodied environments for interactive learning. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=0IOX0YcCdTn.

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. In *Intrinsically-Motivated and Open-Ended Learning Workshop @NeurIPS2023*, 2023. URL https://openreview.net/forum?id=nfx5IutEed.

Yu Wang and Xi Chen. Mirix: Multi-agent memory system for llm-based agents, 2025. URL https://arxiv.org/abs/2507.07957.

Zilong Wang, Yuedong Cui, Li Zhong, Zimin Zhang, Da Yin, Bill Yuchen Lin, and Jingbo Shang. OfficeBench: benchmarking language agents across multiple applications for office automation. *arXiv preprint arXiv:2407.19056*, 2024. URL https://arxiv.org/abs/2407.19056.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=NTAhi2JEEE.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W White, Doug Burger, and Chi Wang. Autogen: Enabling next-gen LLM applications via multi-agent conversations. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=BAakY1hNKS.

Shanglin Wu, Yuyang Luo, Yueqing Liang, Kaiwen Shi, Yanfang Ye, Ali Payani, and Kai Shu. Scaling teams or scaling time? memory enabled lifelong learning in llm multi-agent systems, 2026. URL https://arxiv.org/abs/2604.03295.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-mem: Agentic memory for LLM agents. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=FiM0M8gcct.

Zhongming Yu, Naicheng Yu, Hejia Zhang, Wentao Ni, Mingrui Yin, Jiaying Yang, Yujie Zhao, and Jishen Zhao. Multi-agent memory from a computer architecture perspective: Visions and challenges ahead, 2026. URL https://arxiv.org/abs/2603.10062.

Guibin Zhang, Muxin Fu, Kun Wang, Guancheng Wan, Miao Yu, and Shuicheng YAN. G-memory: Tracing hierarchical memory for multi-agent systems. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025a. URL https://openreview.net/forum?id=mmIAp3cVS0.

Qizheng Zhang, Michael Wornow, and Kunle Olukotun. Agentic plan caching: Test-time memory for fast and cost-efficient llm agents. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 103270–103296. Curran Associates, Inc., 2025b. doi: 10.52202/085713-3451. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/9549f7d06700f0966d5f938f1d11022a-Paper-Conference.pdf.

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. WebArena: a realistic web environment for building autonomous agents. In *International Conference on Learning Representations*, pp. 15585–15606, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/4410c0711e9154a7a2d26f9b3816d1ef-Paper-Conference.pdf.

Kunlun Zhu, Hongyi Du, Zhaochen Hong, Xiaocheng Yang, Shuyi Guo, Zhe Wang, Zhenhailong Wang, Cheng Qian, Xiangru Tang, Heng Ji, and Jiaxuan You. Multiagentbench: Evaluating the collaboration and competition of llm agents. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics*, pp. 8580–8622, 2025. doi: 10.18653/v1/2025.acl-long.421. URL https://aclanthology.org/2025.acl-long.421/.

A Algorithm Overview

Algorithm 1 summarizes how SEPM connects collaborative execution to procedural memory evolution. Retrieved procedures initialize a task dependency graph; structured execution records support alignment during the task and memory updates after it.

Algorithm 1 SEPM: collaborative execution and procedural memory evolution

Require: Task τ, procedure pool $\mathcal{P}$, execution budget $\mathcal{B}_{\text{exec}}$
Ensure: Task outcome y and updated procedure pool $\mathcal{P}$
$\mathcal{R} \leftarrow \text{RetrieveCompatible}(\tau, \mathcal{P})$
$(\mathcal{G}, B, \Sigma) \leftarrow \text{InitializeTask}(\tau, \mathcal{R})$
while task unfinished and budget remains **do**
    Execute a ready step, or obtain missing evidence and replan
    Append $(\text{task}, \text{observation}, \text{action}, \text{result}, \text{error})$ to B
    Update step status and resolve verified state Σ from the recorded evidence
    if goal, plan, or state divergence is detected **then**
        Realign affected agents and update $\mathcal{G}$ and its prerequisites
    end if
end while
$y \leftarrow \text{Outcome}(\tau, B, \Sigma)$; $H \leftarrow (B, \mathcal{G}, \Sigma, y)$
for each prechecked candidate S' with reference S proposed from H **do**
    $\mathcal{T}_{S'} \leftarrow \text{RecordedTrials}(S')$
    if $\text{Accept}(S'; \mathcal{T}_{S'})$ **then**
        Create or atomically revise the active procedure in $\mathcal{P}$
    end if
end for
Atomically attach supported conditional lessons to the relevant procedures
return $y, \mathcal{P}$

Execution and update operations. A ready step has verified prerequisites in the current task state. When no step is ready, the host controller obtains the missing observation or revises the plan. Execution, observation, retries, and replanning use the task's existing budget. The outcome y records completion or the terminal failure or unresolved status. Each retrieved procedure retains the revision used as the reference for its update.

ProposeUpdates extracts supported steps and dependencies from H to form new procedures or revisions. Both new procedures and revisions require trial-based promotion under Equation 9. Trial records are supplied through the reproduction-recording interface; candidates without sufficient promotion evidence remain outside the active pool. Updates retain their applicability conditions, and conditional lessons are stored with their supporting evidence. Warning updates can be retained even when the task does not support a structural revision. The commit operation preserves revision history as described in Appendix B.6.

B Method Details

B.1 Procedure Retrieval

Retrieval filters the active procedure pool by context conditions and exclusions, then ranks compatible entries using Equation 3. The default configuration retrieves $K = 5$ procedures. BM25 uses $k_1 = 1.5$ and $b = 0.75$; its scores are normalized by the maximum in the candidate set. The default text representation f uses 256-dimensional BLAKE2b-based signed feature hashing with ℓ_2 normalization and a bounded cosine-based similarity. Indexed fields include the title, description, task family, applicability, tags, context conditions, warnings, and step instructions.

Adaptive retrieval initializes the BM25 and vector logits to zero and applies a softmax to obtain their weights. The implementation's routing cues modify these logits as follows:

Query cues	Modified logit	Increment
`code`, `debug`, `implement`, `coding`	Vector	1.0
`latest`, `current`, `real-time`	BM25	0.8
`routine`, `repeat`, `standard`	BM25	0.6

With no routing cue, both weights are 0.50. Disabling adaptive retrieval also fixes $w_b = w_v = 0.50$.

B.2 Goal and Plan Alignment

Goal comparison. Goal divergence combines similarity between hashed text representations with task-conditioned goal consistency. The representation term uses the mapping f described in Appendix B.1. For J_{goal}, `gpt-5-mini` receives the agent's current goal, its assigned goal, and the global task objective with its constraints. It returns a score in $[0, 1]$, where higher values indicate greater consistency of the intended operation with the assignment and global constraints. We set $\beta = 0.55$ and trigger goal realignment when $D_g > 0.35$.

Plan comparison. Nodes are matched by persistent subtask identifiers assigned during task-graph instantiation. A different action field at a matched identifier counts as one node edit; a missing node receives a null action label. Edge differences are the symmetric difference of the directed edge sets. Ownership and completion status are stored separately and do not enter the score. Equation 6 therefore measures a normalized discrepancy under fixed node correspondence. Plan realignment is triggered when $D_p > 0.40$.

B.3 Evidence-Based State Resolution

A state key k identifies an entity-attribute pair. Each evidence item e records a value, authority tier $h(e)$, source $s(e)$, observation time $t(e)$, and confidence $c(e)$. Evidence is eligible when $c(e) \geq 0.70$:

$$\mathcal{E}_k^+ = \{e \in \mathcal{E}_k : c(e) \geq 0.70\}. \tag{10}$$

If this set is empty, the state remains unresolved and the controller requests an observation. Otherwise, the resolver selects the highest available tier,

$$h_k^\star = \max_{e \in \mathcal{E}_k^+} h(e), \qquad \mathcal{E}_k^\star = \{e \in \mathcal{E}_k^+ : h(e) = h_k^\star\}. \tag{11}$$

The authority order is environment or official state, tool observation, verified artifact, and agent inference. Agreement within the highest qualified tier establishes the reported value. For conflicting observations from the same source and tier, the latest observation is used; a tie remains unresolved. Conflicts across sources require a fresh authoritative or tool observation. Confidence determines eligibility rather than breaking ties between contradictory values.

Agent inference alone cannot establish a verified external state. The controller first seeks external evidence and uses negotiation when no external source is available. A newer report that conflicts with stored state triggers re-observation of the relevant key. State resolution operates on individual keys and same-source observation times; it does not implement global temporal reasoning or a time-to-live policy.

B.4 Task Graphs, Retries, and Replanning

The task graph represents the current prerequisite structure. Each logical step has a persistent identifier and an action description. Self-loops, duplicate edges, and cyclic dependencies are rejected. Repeating an action creates another execution record for its logical step rather than a backward edge in the graph.

After a failed check, the controller may retry the step or the manager may produce a revised acyclic plan. Unchanged steps retain their identifiers; new steps receive new identifiers. A failed or modified step requires fresh verification of affected downstream conditions, while unaffected prerequisites retain their supporting evidence. Execution ends when the task completes, the host budget is exhausted, or the controller returns an unresolved state.

B.5 Procedure Extraction and Revision

The terminal history H_τ associates blackboard records with the task graph, verified state, and outcome. Extraction uses the supported node and edge sets in Equation 8: a retained node has evidence for its action and outcome, and a retained edge has evidence for the corresponding prerequisite relation. The resulting procedure abstracts supported steps and retains their dependencies, applicability conditions, and conditional warnings.

Execution of a previously unseen workflow can produce a candidate procedure, while experience with an existing procedure can produce a candidate revision. Candidates remain separate from the active procedure pool until recorded trials satisfy the promotion requirements. Execution failures and other supported observations may also be retained as conditional lessons. Recording a candidate or an execution event does not itself create an active procedure version. Failures can yield conditional warnings without promoting an unverified structural change. Retries and repairs are part of the current execution.

Candidate prechecks. At the proposal and validation stages, a candidate without recorded trials is screened using its supplied benefit and cost:

$$u_{\text{claim}} = b_{\text{claim}} - \lambda c_{\text{claim}}, \qquad \lambda = 1.0. \tag{12}$$

This quantity is a candidate-level estimate used for prechecking. It does not replace the reproduction evidence required for promotion.

Trial-based promotion. For candidate S', let $\mathcal{T}_{S'}$ contain all recorded reproduction trials. Each trial t supplies a candidate reward, a baseline reward, and a cost:

$$u_t = r_t^{\text{candidate}} - r_t^{\text{baseline}} - \lambda c_t. \tag{13}$$

Promotion evaluates the mean net benefit

$$\overline{u}(S') = \begin{cases} \dfrac{1}{|\mathcal{T}_{S'}|} \displaystyle\sum_{t \in \mathcal{T}_{S'}} u_t, & |\mathcal{T}_{S'}| > 0, \\ -\infty, & |\mathcal{T}_{S'}| = 0. \end{cases} \tag{14}$$

Failed trials are included in this average. The baseline reward is supplied in each trial record for the reference procedure or no-candidate baseline under the same validation setting. It is not inferred from the aggregate benchmark tables.

Promotion requires the safety, state-verification, causal-attribution, reproduction, and nonnegative-net-benefit gates to pass. The default safety-risk threshold is 0.50, the causal-confidence threshold is 0.65, and the minimum reproduction count is 1.

Validation summary score. After successful promotion, the created procedure version stores

$$v_{\text{summary}} = \min\left\{1,\ 0.4c_{\text{causal}} + 0.3\frac{n_q}{\max(m, n_q)} + 0.3(1 - r_{\text{safety}})\right\}, \tag{15}$$

where c_{causal} is the candidate's supplied causal confidence, n_q is the number of qualified trials, m is the minimum reproduction count, and r_{safety} is the safety-risk score. With the default $m = 1$, the reproduction term equals 0.3 whenever at least one trial qualifies. The summary score is stored as version metadata and is not a separate promotion gate.

B.6 Atomic Version Updates

An accepted structural or warning update creates a new revision v'. The update becomes active only if the current head still matches the revision v used to construct the candidate:

$$\left(S_i^{(v)}, \text{head}_i = v\right) \xrightarrow[\text{head}_i = v]{\text{accepted update}} \left(S_i^{(v')}, \text{head}_i = v'\right). \tag{16}$$

A failed head comparison leaves the active revision unchanged. Superseded revisions remain in history, while retrieval uses the active revision.

C Experimental Protocols

C.1 Hosts, Models, and Comparison Conditions

The main comparison evaluates no added memory, Generative Agents, MetaGPT, G-Memory, and SEPM under AutoGen and DyLAN. Each host retains its native orchestration and within-task context. The main planning, execution, and procedural-memory models, together with the goal-consistency judge, use `gpt-5-mini`. The model-sensitivity experiment fixes AutoGen, the main planning agent, and the goal-consistency judge while jointly changing the execution and procedural-memory endpoint. The benchmark tasks, execution budgets, and evaluators remain unchanged across these configurations. Table 2 contains twelve alternative endpoints and the main configuration.

The component experiment compares the host baseline, blackboard only, blackboard with procedural memory, blackboard with alignment, and the full system. The textual-procedure control retains the blackboard, memory, alignment, retrieval rules, context budget, and update mechanism. It expresses prerequisites in text and uses textual references for plan comparison and correction propagation in place of explicit graph objects and adjacency relations.

C.2 Memory Initialization and Persistence

Each benchmark condition run starts with an empty procedure pool. Tasks within the run use the same SQLite database, allowing stored execution records, candidates, trials, and accepted procedure versions to persist across the task sequence. Separate experimental conditions use separate database paths. A resumed run continues from its existing database and checkpoint rather than initializing a new pool.

The generic runtime uses an output-specific database path, which can be overridden through `TEAM_MEMORY_DB`. At agent or task start, the adapter requests up to $K = 5$ compatible procedures. Context-specific variants are eligible only when their context conditions are satisfied by the facts supplied to the adapter.

Tasks follow the sequence selected by the benchmark adapter. For ALFWorld, each task resets the environment, and its case seed is

$$s_i = s_{\text{base}} \times 1000003 + i,$$

where i is the official task index. WebArena and OfficeBench use their respective runtime initialization and reset procedures.

C.3 Benchmark Environments and Evaluation

ALFWorld. We evaluate 134 tasks in the text-based environment with a maximum of 30 actions per task. Evaluation uses the task feedback returned by the ALFWorld environment. The environment resets between tasks, while the memory mechanism retains experience across the evaluated sequence.

WebArena. We evaluate 812 tasks through BrowserGym's action interface using accessibility-tree observations. Tasks are executed in the sequence selected by the WebArena adapter. Each method updates memory from completed interactions using its own update mechanism. Task scores are obtained from WebArena's functional evaluator.

OfficeBench. OfficeBench (Wang et al., 2024) evaluates workflows involving documents, spreadsheets, email, and calendar operations. We evaluate all 300 tasks, comprising 93 single-application, 95 two-application, and 112 three-application tasks. The environment runs in Docker and provides tool APIs organized into Word, Excel, Calendar, Email, System, and OCR-PDF capabilities (Han et al., 2025). Each task specifies an initial collection of files and application data on which the agents operate. After execution, task-specific evaluators inspect the resulting files, application state, or submitted answers against the expected outcome. These checks include exact matching and fuzzy keyword matching where required by the task. We aggregate the task-level evaluation scores over all 300 tasks.

Table 3: Task and communication scores on MultiAgentBench. All methods use AutoGen, `gpt5.5-mini` for execution, and `claude-opus-5` for evaluation. TS and CS retain the benchmark's reporting scale.

	Coding		Research		Database	
Method	TS	CS	TS	CS	TS	CS
No added memory	54.90	39.81	62.40	55.16	64.93	62.43
Generative	55.74	46.09	61.16	64.64	65.21	77.16
MetaGPT	57.62	47.23	70.11	62.27	70.14	80.29
G-Memory	64.20	51.17	74.29	77.77	69.73	88.20
SEPM	**69.09**	**60.64**	**78.37**	**88.91**	**77.03**	**91.94**

MultiAgentBench. We evaluate 100 tasks in each of the coding, research, and database environments (Zhu et al., 2025; Wu et al., 2026). All five conditions use AutoGen, with `gpt5.5-mini` for execution and `claude-opus-5` for evaluation. Task definitions, agent profiles, environment rules, and communication graphs are retained. Each task permits at most three iterations, and memory persists across tasks within a condition's sequence. The generation temperature is 0.7 with a maximum sequence length of 1024; coding uses 0.1 and 4096, respectively.

Task Score (TS) measures output quality or correctness; Communication Score (CS) measures communication and planning. Research judgments of innovation, safety, and feasibility are normalized into TS by the benchmark evaluator. Scores retain the benchmark's reporting scale. The communication graph specifies which agents communicate, while the SEPM task graph specifies prerequisites between executable steps.

C.4 Evaluation Units and Aggregation

Each task condition pair is evaluated once with base seed 0. The evaluation includes 134 ALFWorld tasks, 812 WebArena tasks, and 300 OfficeBench tasks. We retain the task-level scores returned by the corresponding benchmark evaluation pipeline and aggregate them over the evaluated task set.

Figure 4a reports an unweighted arithmetic mean of the three displayed benchmark scores for each method configuration. Differences between displayed scores are reported in points on the same reporting scale. MultiAgentBench TS and CS are reported separately and are excluded from this mean.

C.5 Token Accounting

Token consumption is the sum of input and output tokens over the complete 134-task ALFWorld evaluation under AutoGen. Each total is paired with the task score from the same run in Table 1. The reported totals are 4.4, 5.6, 4.8, 5.7, and 5.2 million tokens for no added memory, Generative Agents, MetaGPT, G-Memory, and SEPM, respectively.

D Supplementary Results and Analysis

D.1 Task Quality and Communication on MultiAgentBench

Table 3 reports the values underlying Figure 1. Relative to G-Memory, SEPM increases TS by 4.89, 4.08, and 7.30 points in coding, research, and database, respectively, and CS by 9.47, 11.14, and 3.74 points. The gains occur in both output and communication scores, which is consistent with the framework's joint treatment of task execution and coordination. Their magnitudes differ across environments: research has the largest CS gain over G-Memory, whereas database has the largest TS gain. These aggregate scores describe complementary outcomes rather than establishing that one causes the other.

D.2 Conditional Contributions of Memory and Alignment

The component experiment forms a two-factor comparison conditional on the shared blackboard. Let s_B, s_{BM}, s_{BA}, and s_{BMA} denote scores with the blackboard, memory, and alignment components indicated by the subscripts. The descriptive interaction contrast is

$$I = s_{BMA} - s_{BM} - s_{BA} + s_B. \tag{17}$$

Table 4: Conditional gains from the component experiment, in percentage points. Differences are computed from the displayed task scores.

Comparison	ALFWorld	WebArena	OfficeBench
Add memory to blackboard	17.17	6.28	0.33
Add memory to blackboard + alignment	7.46	3.82	3.67
Add alignment to blackboard	12.69	6.77	6.00
Add alignment to blackboard + memory	2.98	4.31	9.34
Interaction I	-9.71	-2.46	3.34

On ALFWorld and WebArena, each extension has a smaller marginal gain when the other is present, indicating overlapping benefits on the observed score scale. OfficeBench shows the opposite pattern: memory adds 0.33 points to the blackboard alone but 3.67 points when alignment is available. Alignment similarly adds more when memory is present. This pattern suggests that the value of reusable office procedures depends in part on maintaining the shared state needed to execute them. The contrast is descriptive and is computed from one run per condition.