FROM TRAJECTORIES TO SOPS: REUSABLE PROCEDURES FOR RELIABLE AGENT EXECUTION

Anonymous authors
Paper under double-blind review

ABSTRACT

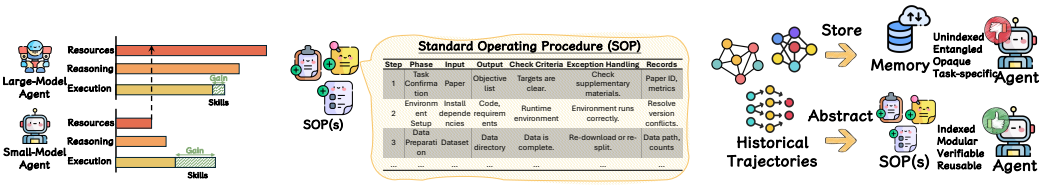
Large language model (LLM)-based multi-agent systems often rely on repeated online reasoning, while the procedural knowledge in their execution trajectories remains tied to individual tasks. We propose an SOP-centric framework that abstracts shared execution logic from teacher-generated trajectories into a codebook of reusable standard operating procedures (SOPs). Each SOP specifies procedural steps, input-output contracts, and verification rules, enabling procedures to be retrieved, composed, and verified across tasks. A large-model Overseer performs SOP-level planning and dispatches compact latent codes grounded in this codebook, while lightweight Executors follow the composed procedures with runtime verification, local repair, and selective escalation. Across AgentSkillOS, SkillsBench, and SWE-bench Verified, our framework improves the macro-average score by 33.1% over the strongest baseline. With a 9B executor, it improves performance by 79.4% over Vanilla while reducing allocated GPU-seconds per task by 18.6%. Compared with natural-language plans, latent SOP dispatch reduces serialized communication payload by a factor of 63.88 without sacrificing task quality. Ablation and perturbation studies support the importance of SOP abstraction, contract verification, and runtime recovery for reliable execution. These results highlight reusable procedures as an effective interface between large-model planning and lightweight execution.

1 INTRODUCTION

In recent years, multi-agent systems (MAS) based on large language models (LLMs) have demonstrated strong capabilities in solving complex tasks (Guo et al., 2024a; Chen et al., 2024a). They have been widely applied to multi-step reasoning (Wei et al., 2022; Yao et al., 2023), tool use (Schick et al., 2023; Yao et al., 2022), web interaction (Deng et al., 2023; Zhou et al., 2024), data analysis (Hong et al., 2025), code generation (Jimenez et al., 2024a; Wang et al., 2025; Chen et al., 2021), and document processing (Liao et al., 2025; Chen et al., 2024b). By organizing multiple agents into collaborative systems, existing methods can decompose complex tasks into subtasks (Hong et al., 2024a; Qian et al., 2024) and achieve end-to-end execution through role assignment (Wu et al., 2023; Li et al., 2023; Dibia et al., 2024), message passing (Xu et al., 2026; Liu et al., 2024a), and tool invocation (Wang et al., 2024c; Liu et al., 2025). During these executions, MAS naturally generate large-scale interaction trajectories, including task decomposition processes, tool invocation sequences, intermediate decisions, and recovery behaviors. These trajectories record valuable information about how agents solve tasks and provide a potential source of reusable procedural knowledge. However, current MAS mainly treat such trajectories as temporary execution records for individual tasks, rather than structured knowledge that can be accumulated, abstracted, and reused across different tasks.

Despite the procedural information contained in these trajectories, current MAS still solve new tasks primarily through the online reasoning and decision-making of LLM-based agents. Current mainstream MAS typically adopt an agent-centric paradigm (Wang et al., 2024b; Guo et al., 2024a), where LLMs serve as the core components of agents (Li et al., 2025). These agents rely on contextual understanding (Huang et al., 2024), planning (Qiao et al., 2024), reasoning (Wei et al., 2022), and instruction-following capabilities to perform task decomposition (Ouyang et al., 2022), make execution decisions (Yao et al., 2022), and refine intermediate results (Madaan et al., 2023). Under this paradigm, each new task is largely handled by repeatedly invoking the model's general capabilities,

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.



(a) Agent capability gap. (b) Structured SOP representation. (c) Trajectory-to-SOP abstraction.

Figure 1: Motivation for SOP-centric procedural knowledge. (a) Capability gap between large- and small-model agents and the execution gains from procedural guidance. (b) Structured SOP representation with explicit execution and verification information. (c) Abstraction of historical trajectories into indexed, modular, verifiable, and reusable procedural knowledge.

while prior execution trajectories are not explicitly represented as reusable procedural structures. To improve the utilization of historical experience, existing approaches have explored memory augmentation (Shinn et al., 2023), trajectory replay (Zhao et al., 2024; Zheng et al., 2024), and skill libraries (Wang et al., 2024a). However, these methods mainly retrieve similar experiences, reproduce previously successful behaviors, or reuse task-specific skills, leaving the reusable units closely tied to particular trajectories, contexts, or task instances. They therefore provide limited support for extracting the underlying procedures shared across tasks and organizing heterogeneous experiences into transferable, composable, and verifiable procedural knowledge. Consequently, performance improvements in current MAS still rely substantially on model scale and inference budgets (Kaplan et al., 2020; Hoffmann et al., 2022; Snell et al., 2024), rather than on the systematic abstraction and reuse of accumulated execution experience.

These limitations suggest that effective experience reuse requires abstracting the shared execution logic underlying prior trajectories into reusable procedural structures, rather than merely retrieving or replaying them. Evidence from tool-use systems, agent-based execution, and our empirical results suggests that the capability gap between large and small models varies across execution stages (Liu et al., 2024b). As illustrated in Figure 1a, small models are relatively weak in high-level task understanding, planning, and open-ended decision-making (Valmeekam et al., 2023; Xie et al., 2024), but can substantially narrow this gap when execution is guided by explicit procedures (Qin et al., 2024). This indicates that reliable execution depends not only on model capability, but also on structured constraints that can organize, guide, and verify the execution process (Yang et al., 2024; Kim et al., 2024; Khot et al., 2023). Based on this analysis, we pose the following central question:

> *How can historical execution trajectories be transformed into reusable, composable, and verifiable procedural structures?*

To answer this question, we introduce Standard Operating Procedures (SOPs) as structured abstractions of procedural knowledge. As illustrated in Figure 1b, an SOP explicitly specifies its structure and execution-related contents. Building on this idea, we propose an SOP-centric multi-agent framework in which task execution is organized around reusable procedures rather than relying solely on the implicit reasoning of monolithic agents. In our formulation, SOPs serve as the primary reusable procedural units, while skills represent atomic operations that can be composed within them. As illustrated in Figure 1c, historical execution trajectories are abstracted into structured and reusable SOP programs by identifying their shared execution logic and encoding applicability conditions, execution steps, tool interfaces, input-output constraints, and verification rules. The resulting SOPs are organized into a structured codebook that supports cross-task retrieval, reuse, and composition. During online execution, relevant SOPs are retrieved and composed into an executable SOP tree. A large-model Overseer performs task understanding and SOP-level planning, while a lightweight Executor carries out skill-level operations under explicit procedural constraints, with runtime verification and local repair ensuring execution reliability. The main contributions of this paper are summarized as follows:

- We formulate procedural knowledge reuse as a central problem in multi-agent systems and propose an SOP-centric framework that organizes accumulated execution experience into reusable structures for cross-task generalization.

- We introduce a trajectory-to-SOP abstraction method that extracts shared execution logic from task-specific trajectories and represents it through explicit applicability conditions, procedural steps, tool interfaces, input-output constraints, and verification rules.
- We construct a retrievable and composable SOP codebook that supports the organization, retrieval, and composition of procedural knowledge beyond instance-level trajectory matching.
- We develop an SOP-grounded execution framework in which a large-model Overseer performs procedural planning and composition, while a lightweight Executor carries out constrained execution with runtime verification and local repair, enabling efficient and reliable task completion.

2 Related Work

LLM-based Multi-Agent Systems. LLM-based multi-agent systems tackle complex tasks by coordinating agents through role assignment, communication, task decomposition, tool use, and iterative refinement (Guo et al., 2026; Li et al., 2024a; Zhu et al., 2025; Patil et al., 2024; Bo et al., 2024). AutoGen, AgentVerse, ChatDev, and MetaGPT exemplify this paradigm across diverse tasks (Wu et al., 2023; Chen et al., 2024a; Qian et al., 2024; Hong et al., 2024b). However, most systems remain agent-centric: planning and execution depend heavily on large-model reasoning, tying performance to model scale, context length, and inference budget (Yang et al., 2026; Tran & Kiela, 2026; Guo et al., 2024a). This motivates more structured and efficient execution that reduces dependence on increasingly larger models.

Tool Use, Memory, and Skill Reuse. Recent work augments LLM agents with tools, memory, historical trajectories, and skill libraries. ReAct (Yao et al., 2022), ToolLLM (Qin et al., 2024), ToolBench (Guo et al., 2024b), Reflexion (Shinn et al., 2023), ExpeL (Zhao et al., 2024), MemoryBank (Zhong et al., 2024), and Voyager (Wang et al., 2024a) demonstrate gains from tool use, experience storage, and skill reuse. However, experience is often retained as raw trajectories, textual memories, reflections, examples, or isolated skills. These representations remain context-specific and rarely encode execution conditions, input-output interfaces, step dependencies, verification criteria, or exception handling (Xiong et al., 2026; Hu et al., 2026; Luo et al., 2026; Fang et al., 2026). This limits scalable and reliable reuse and motivates explicit procedural abstraction.

Structured Workflows and SOP-guided Execution. Structured workflows and SOPs have been used to improve agent reliability. MetaGPT (Hong et al., 2024a) introduces SOP-style collaboration for software engineering, while SOP-agent (Ye et al., 2025) and related methods guide execution with human-written procedures, decision graphs, or modular workflows (Nandi et al., 2025; Khot et al., 2023). Such procedural constraints can improve controllability in long-horizon tasks (Li et al., 2024b). However, most methods rely on manually designed procedures or static prompt guidance, leaving procedural knowledge from historical trajectories underexplored (Liu et al., 2026; Yue et al., 2026). We address this gap by treating SOPs as reusable, composable, and verifiable abstractions for lightweight-agent execution.

3 Method

3.1 Overview

We consider an SOP-centric multi-agent system where standard operating procedures, rather than fixed agent roles, serve as the primary reusable units for task execution, as shown in Figure 2. Given a task query q and a tool environment $\mathcal{E}_{\text{tool}}$, the system maintains an SOP codebook $\mathcal{C} = \{\sigma_i\}_{i=1}^{N}$. Each SOP entry is represented as $\sigma_i = (\iota_i, d_i, \rho_i, \chi_i, b_i, v_i, \mu_i), \chi_i = (\chi_i^{\text{in}}, \chi_i^{\text{out}})$, where ι_i is the SOP identifier, d_i denotes the task intent, ρ_i specifies the procedural policy, χ_i defines the input-output constraints, b_i denotes tool or executor bindings, v_i is the verifier, and μ_i indicates the executor or model requirement. The framework contains an offline construction stage and an online execution stage. In the offline stage, a large teacher model generates expert trajectories in $\mathcal{E}_{\text{tool}}$, from which reusable procedural patterns are abstracted, normalized, clustered, deduplicated, and verified to construct the SOP codebook. In the online stage, given a new task q and execution state S_t, the overseer produces a discrete latent SOP code z_t instead of a full natural-language plan. The runtime

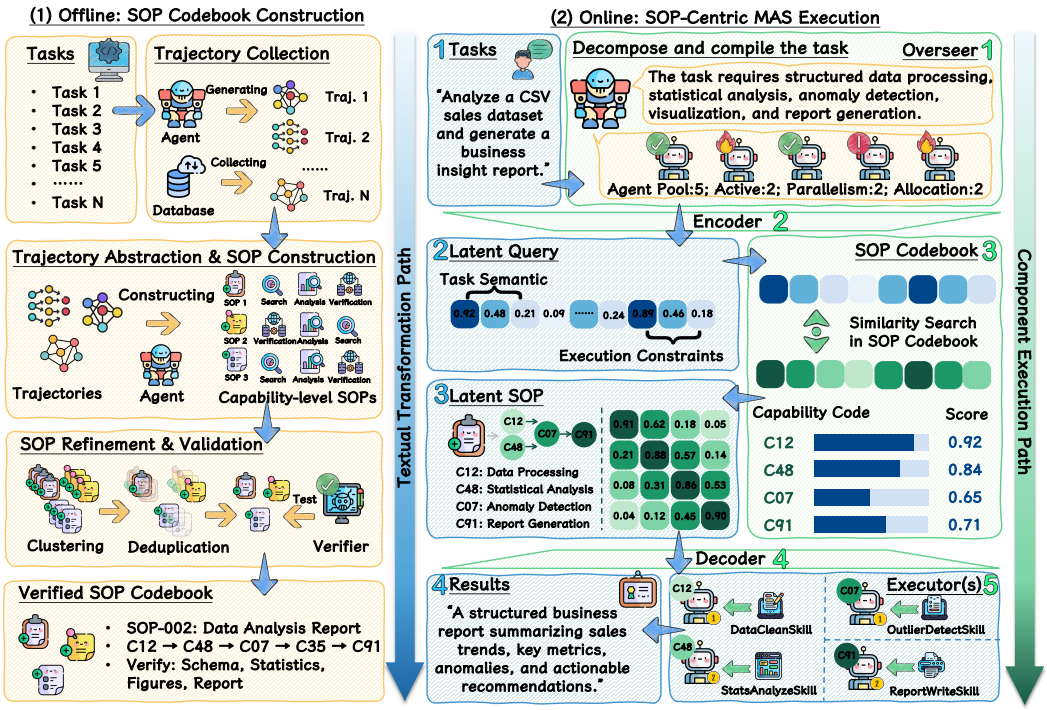


Figure 2: Overview of the proposed SOP-centric MAS framework.

grounder uses z_t, together with the current task-state constraints and tool availability, to retrieve a candidate SOP set Π_t. The selected SOPs are then compiled into an executable dependency-guided SOP structure. Lightweight executors dynamically load the selected SOPs, invoke tools according to their procedural policies, and use the associated verifiers to check whether the required constraints are satisfied. If verification fails, the runtime monitor triggers retry, local repair, alternative SOP loading, or escalation to the overseer.

3.2 Teacher-Induced SOP Codebook Construction

To construct reusable procedural knowledge, we use a large teacher model to generate execution trajectories from a set of training tasks $\mathcal{Q}_{\text{train}}$. For each task $q_j \in \mathcal{Q}_{\text{train}}$, the teacher interacts with $\mathcal{E}_{\text{tool}}$ and produces a trajectory $\tau_j = (q_j, \{(S_{j,t}, a_{j,t}, o_{j,t})\}_{t=1}^{T_j}, y_j, e_j)$, where $S_{j,t}$, $a_{j,t}$, and $o_{j,t}$ denote the execution state, tool action, and observation at step t for task q_j, y_j is the final output, and e_j records execution metadata such as success status, tool usage, and recovery behavior. The trajectory set $\mathcal{T} = \{\tau_j\}_{j=1}^{M}$ is converted into the SOP codebook through a sequential transformation pipeline: $\mathcal{T}^{(1)} = F_{\text{abs}}(\mathcal{T}), \mathcal{T}^{(2)} = F_{\text{norm}}(\mathcal{T}^{(1)}), \mathcal{T}^{(3)} = F_{\text{cluster}}(\mathcal{T}^{(2)}), \mathcal{T}^{(4)} = F_{\text{dedup}}(\mathcal{T}^{(3)}), \mathcal{C} = F_{\text{ver}}(\mathcal{T}^{(4)})$, where F_{abs}, F_{norm}, F_{cluster}, F_{dedup}, and F_{ver} denote SOP abstraction, canonical normalization, semantic clustering, duplicate removal, and executability verification, respectively. Each verified SOP entry adopts a structured representation that supports online retrieval, composition, execution, and verification under explicit procedural constraints.

3.3 Codebook-Grounded Latent SOP Dispatch

To reduce communication and context cost, online dispatch is represented as a discrete latent SOP code rather than a full natural-language plan or an explicit SOP tree. For each SOP entry σ_i, we serialize its procedural information into x_i and encode it into a low-dimensional vector $u_i = f_\theta(x_i) \in [-1, 1]^D$. Under Finite Scalar Quantization (FSQ) (Mentzer et al., 2024), let the level configuration be $\mathbf{L} = (L_1, \ldots, L_D)$. The quantizer maps u_i into a discrete latent SOP code $z_i = Q_{\mathbf{L}}(u_i)$, $z_i = (z_{i,1}, \ldots, z_{i,D})$, $z_{i,d} \in \{1, \ldots, L_d\}$. The FSQ code space is

$\mathcal{Z}_{\mathrm{FSQ}} = \{1,\dots,L_1\} \times \cdots \times \{1,\dots,L_D\}, |\mathcal{Z}_{\mathrm{FSQ}}| = L_1 \times \cdots \times L_D$. Since multiple SOP entries may share the same discrete code, we define the latent index as $\mathcal{I}(z) = \{\sigma_i \in \mathcal{C} \mid z_i = z\}$. Given a new task q and state S_t, the overseer internally computes $u_t = f_\theta(q, S_t), z_t = Q_{\mathbf{L}}(u_t)$, and transmits only the discrete latent SOP code $h_t = z_t$. The runtime grounder first selects a code-neighborhood candidate set $\mathcal{C}_t^{(B)} = \{\sigma_i \in \mathcal{C} \mid d_{\mathrm{H}}(z_t, z_i) \le B\}$, where B is the Hamming-radius retrieval budget. It then ranks the candidate SOP entries according to discrete-code distance, input-output compatibility, and availability of the required tools and executors:

$$
\begin{aligned}
r_t(z_t, q, S_t, \sigma_i) &= -\lambda_1 \frac{d_{\mathrm{H}}(z_t, z_i)}{D} + \lambda_2 s_{\mathrm{io}}(q, S_t, \sigma_i) + \lambda_3 s_{\mathrm{avail}}(\mathcal{E}_{\mathrm{tool}}, \sigma_i), \\
\Pi_t &= \operatorname*{Top}_{\substack{k_r \\ \sigma_i \in \mathcal{C}_t^{(B)}}} r_t(z_t, q, S_t, \sigma_i).
\end{aligned}
\tag{1}
$$

Here, d_{H} denotes Hamming distance between FSQ codes, s_{io} measures input-output compatibility with the current task and state, s_{avail} checks whether the tool bindings b_i and executor/model requirement μ_i are available in $\mathcal{E}_{\mathrm{tool}}$, and k_r is the number of retrieved SOP entries.

3.4 Dynamic SOP Loading and SOP Tree Execution

After latent SOP grounding, the selected SOP entries Π_t are compiled into an executable dependency-guided SOP structure. We use the term SOP tree to describe the execution organization, while formally allowing a directed dependency graph to support parallel branches and multi-parent dependencies: $\mathcal{G}_t = (\mathcal{V}_t, \mathcal{D}_t), \mathcal{V}_t \subseteq \Pi_t, \mathcal{D}_t = \mathrm{Compile}_{\mathrm{edge}}\left(\mathcal{V}_t, \{\chi_i\}_{\sigma_i \in \mathcal{V}_t}, q, S_t\right)$, where $\mathcal{V}_t$ is the set of executable SOP nodes and $\mathcal{D}_t$ is the dependency edge set. For each dependency edge, we require $(\sigma_i, \sigma_j) \in \mathcal{D}_t \Rightarrow s_{\mathrm{dep}}(\chi_i^{\mathrm{out}}, \chi_j^{\mathrm{in}}, q, S_t) = 1$, where s_{dep} measures whether the output constraint of σ_i can satisfy the input constraint of σ_j under the current task and state. The SOP structure is executed by lightweight executors through dynamic loading. Let ℓ_t denote the executor assigned to the active SOP node σ_i. At timestep t, the executor selects a tool action according to the loaded SOP: $a_t \sim p_{\ell_t}(\cdot \mid S_t, \rho_i, b_i), o_t \sim P_{\mathcal{E}_{\mathrm{tool}}}(\cdot \mid S_t, a_t), S_{t+1} = g(S_t, a_t, o_t)$. The verifier then checks node-level completion: $\eta_i = v_i(S_{t+1}; \chi_i, q), \eta_i \in \{0, 1\}$. Verified nodes release their successors for execution. Independent nodes can be dispatched in parallel, while dependent nodes are executed according to $\mathcal{D}_t$. If $\eta_i = 0$, the runtime monitor triggers retry, local repair, alternative SOP loading, or escalation to the overseer. This enables lightweight executors to complete complex tasks by following dynamically loaded SOPs rather than planning from scratch.

4 Experiments

In this section, we conduct extensive experiments to answer: **(RQ1)** How effectively does our SOP-centric framework improve task execution across diverse agent benchmarks compared with existing agent- and skill-based methods? **(RQ2)** After historical execution trajectories are transformed into reusable SOPs, can the resulting procedural knowledge be effectively transferred to lightweight executors while reducing computational and communication overhead? **(RQ3)** How do the SOP codebook, latent instruction grounding, runtime verification, and other key components affect the reliability and robustness of the framework?

4.1 Experimental Setup

Benchmarks. We evaluate our framework on three complementary agent benchmarks covering artifact-oriented tool use, general procedural execution, and repository-level software engineering:

- **AgentSkillOS** (Li et al., 2026a): A benchmark containing 30 artifact-oriented tasks, evenly distributed across five categories: data computation, document creation, motion and video processing, visual creation, and web interaction. Each task is evaluated using deterministic structural, schema, file, and content checks.
- **SkillsBench** (Li et al., 2026b): We use the frozen 82-task snapshot adopted by our evaluation pipeline. The benchmark covers heterogeneous tool-use and procedural execution scenarios, including information processing, file manipulation, content generation, and environment interaction. Success is determined by the benchmark-specific final task verifier.

Table 1: Overall performance comparison across three benchmarks. All LLM-based methods use QWEN3.5-9B under the same evaluation setup. Results are reported as mean ± standard deviation. Best results are bolded and runner-ups are underlined.

Method	AgentSkillOS	SkillsBench	SWE-bench	Avg.
Vanilla	30.83 ± 1.67	4.57 ± 1.60	22.20 ± 0.63	19.20
ReAct (Yao et al., 2022)	25.83 ± 1.61	5.79 ± 1.63	17.80 ± 0.66	16.48
Plan-and-Execute (LangChain Team, 2023)	30.83 ± 1.57	7.01 ± 1.61	20.40 ± 0.67	19.42
Reflexion (Shinn et al., 2023)	36.67 ± 2.72	8.54 ± 1.00	23.20 ± 0.83	22.80
BM25 Skill Retrieval (Robertson & Zaragoza, 2009)	32.50 ± 1.57	7.01 ± 1.64	21.60 ± 0.68	20.37
SBERT Skill Retrieval (Reimers & Gurevych, 2019)	<u>42.50</u> ± 1.47	9.76 ± 1.03	23.60 ± 0.61	25.29
AgentSkillOS (Li et al., 2026a)	40.83 ± 1.63	<u>10.98</u> ± 1.21	25.80 ± 0.85	25.87
Agentless (Xia et al., 2025)	35.83 ± 1.77	4.57 ± 0.61	25.20 ± 0.83	21.87
OpenHands (Wang et al., 2025)	40.00 ± 2.72	6.10 ± 1.00	27.40 ± 0.81	24.50
AutoCodeRover (Zhang et al., 2024)	36.67 ± 2.72	7.32 ± 1.53	28.20 ± 0.83	24.06
SWE-Fixer (Xie et al., 2025)	34.17 ± 1.66	6.10 ± 1.27	<u>28.80</u> ± 0.85	23.02
SWE-Search (Antoniades et al., 2025)	32.50 ± 1.34	6.10 ± 1.33	26.40 ± 0.89	21.67
MASAI (Arora et al., 2024)	35.83 ± 1.59	7.01 ± 0.61	26.00 ± 0.73	22.95
AutoGen (Wu et al., 2023)	32.50 ± 1.67	4.88 ± 1.11	22.80 ± 0.93	20.06
Ours	**56.67** $^{1.84\times}_{\pm 1.04}$	**15.85** $^{3.47\times}_{\pm 0.93}$	**30.80** $^{1.39\times}_{\pm 0.63}$	**34.44** $^{1.79\times}$

- **SWE-bench Verified** (Jimenez et al., 2024b): A repository-level software engineering benchmark consisting of 500 verified issue-resolution instances from 12 real-world repositories. Agents must inspect the repository, identify the cause of an issue, and generate an executable code patch. We use the official test split and determine success exclusively through the `resolved` field produced by the official evaluation harness.

Model and Tool Configuration. Unless otherwise specified, we use Qwen3.5-122B-A10B (Qwen Team, 2026) as the Overseer for task understanding, planning, and SOP construction, and Qwen3.5-9B as the executor for tool invocation and procedural execution. Fixing the executor across methods allows us to isolate the gains introduced by SOP-based guidance. All three benchmarks share the top-1,000 tool library provided by AgentSkillOS, with benchmark-specific adapters mapping the common tools to their respective execution environments.

Metrics. We report benchmark-specific task success using strict objective completion for AgentSkillOS, final-verifier pass for SkillsBench, and the official `resolved` field for SWE-bench Verified. Overall performance is measured by the equal-weighted macro average, $\mathrm{MacroAvg} = (A + S + W)/3$. Efficiency is evaluated using allocated GPU-seconds per task, performance per GPU-second, and serialized payload per episode. For framework analysis, we further report codebook compression and coverage, grounding accuracy, component-wise Macro Drop, and runtime failure-recovery and escalation rates.

Baselines. We compare our method with unstructured agents, skill-retrieval methods, structured workflows, and controlled variants of our framework. The baselines include No Skill, ReAct, Plan-and-Execute, Reflexion, Random, BM25, Dense and Oracle Retrieval, retrieved skill DAGs, AgentSkillOS-style execution, LangGraph, AutoGen, MetaGPT, and benchmark-specific software engineering agents. We further consider SOP Only, Tree Only, and SOP without Runtime Repair. All methods share the same models, tool library, execution constraints, and evaluation protocol within each benchmark, while external systems are adapted to our unified runners for fair comparison.

Experimental Platform. All experiments were conducted under a unified hardware and software environment to ensure fair comparison and reproducibility of the reported latency, energy, and execution results. Specifically, the evaluation was performed on a server running Ubuntu 22.04.5 LTS with Python 3.12.12, PyTorch 2.8.0+cu128, and CUDA 12.8. The platform was equipped with 8 NVIDIA H100 80GB HBM3 GPUs, dual Intel Xeon Platinum 8562Y+ CPUs, and 1.0 TiB system memory.

Table 2: Executor scaling and performance–compute trade-offs across QWEN3.5 model sizes. Performance is the macro-average score across three benchmarks, and compute cost is measured in GPU-seconds per task. Red superscripts denote changes relative to Vanilla.

	Performance ↑			Compute Cost ↓	
Executor	**Vanilla**	**Skills**	**Ours**	**Vanilla**	**Ours**
QWEN3.5-2B	4.42	**6.08**	4.25 $^{-0.17}$	226.41	245.89 $^{+8.61\%}$
QWEN3.5-4B	**22.24**	18.35	21.00 $^{-1.24}$	554.87	595.90 $^{+7.39\%}$
QWEN3.5-9B	19.20	25.87	**34.44** $^{+15.24}$	306.06	249.24 $^{-18.56\%}$
QWEN3.5-27B	27.32	30.74	**32.52** $^{+5.21}$	959.37	746.57 $^{-22.18\%}$
QWEN3.5-122B	35.35	34.48	**38.81** $^{+3.46}$	775.65	1046.67 $^{+34.94\%}$

4.2 MAIN RESULTS (RQ1)

Observation 1: The SOP-centric framework achieves the highest scores across all three benchmarks under the matched evaluation setting. As shown in Table 1, our full framework ranks first on AgentSkillOS, SkillsBench, and SWE-bench Verified under the same executor, tool pool, and evaluation protocol, achieving scores of 56.67, 15.85, and 30.80. It obtains an average score of 34.44 and surpasses the strongest baseline, AgentSkillOS, by 33.1%. Compared with the Vanilla executor, it achieves performance gains of 1.84×, 3.47×, and 1.39× across the three benchmarks. These gains span general tool use, multi-step skill execution, and software engineering, showing that the improvement is not driven by a single task domain. The results demonstrate that organizing procedural knowledge as reusable SOPs provides more effective and consistent execution guidance than agent-centric reasoning and direct skill retrieval.

4.3 EFFICIENCY AND TRANSFER ANALYSIS (RQ2)

Observation 2: Procedural transfer is most effective at the 9B executor scale, while overly small or large executors reduce its overall efficiency.

As shown in Table 2 and Fig. 3, QWEN3.5-9B achieves the best trade-off, improving performance by 79.4% over Vanilla and 33.1% over Skills while reducing compute cost by 18.6%, yielding a 120.3% efficiency uplift. The 27B executor remains beneficial but with a smaller 53.0% uplift. In contrast, the 122B executor incurs 34.9% higher compute cost, making its overall efficiency negative despite continued performance gains. The 2B and 4B results further indicate that insufficient executor capacity also limits the benefit of procedural guidance.

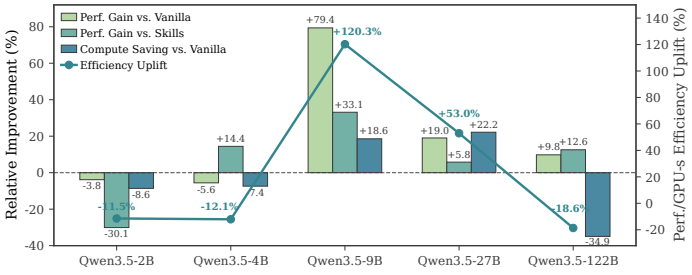


Figure 3: Executor scaling across Qwen3.5 model sizes. Bars show relative performance gains over Vanilla and Skills and compute savings over Vanilla; the line reports performance-per-GPU-second efficiency uplift.

Observation 3: Latent instruction improves task quality and substantially reduces dispatch payload, while execution-side overhead remains workload dependent. As shown in Table 3, Latent Instruction reaches the highest Macro Score and Macro Success of 27.10 and 20.36, both approximately 1.10× those of the Natural-Language Plan, while reducing the serialized payload from 4,860.00 to only 76.08 bytes per episode, a 63.88× reduction. SOP Tree provides an intermediate trade-off, improving task quality while moderately reducing pay-

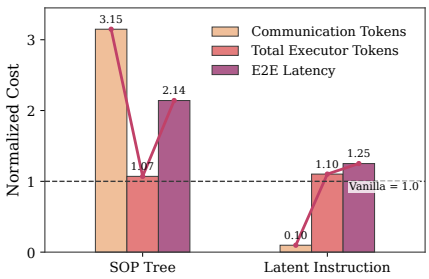


Figure 4: Normalized communication and execution costs relative to Vanilla.

Table 3: Task quality and serialized payload across dispatch formats. Red superscripts show relative improvement factors over the Natural-Language Plan.

Dispatch Format	Macro Score ↑	Macro Success ↑	Payload Bytes/Episode ↓
Natural-Language Plan	24.69	18.43	4,860.00
SOP Tree	26.26	19.93	3,215.05
Latent Instruction	**27.10** $^{1.10\times}$	**20.36** $^{1.10\times}$	**76.08** $^{63.88\times}$

load. This indicates that the substantial compression of Latent Instruction does not come at the cost of task quality. Figure 4 shows that Latent Instruction reduces normalized communication tokens to 0.10 of the Vanilla baseline, while executor tokens and end-to-end latency increase to 1.10 and 1.25, respectively. Thus, its communication benefit does not imply a reduction in execution-side consumption or end-to-end latency.

4.4 Framework Analysis (RQ3)

Observation 4: Our SOP-centric framework compresses execution experience into a compact yet broadly executable codebook, while latent grounding further improves precise and execution-aware SOP selection. As shown in Table 4, 1,086 trajectories and 400 raw candidates are consolidated into 54 final SOPs, achieving 86.50% compression while retaining 87.91% task coverage and 94.44% lightweight executability. Each SOP is supported by 13.59 trajectories on average, indicating that the codebook captures reusable procedural patterns rather than task-specific traces. Figure 5 further shows that FSQ complements semantic retrieval. Compared with the trained encoder, FSQ+Encoder improves R@1 by 1.25×, while R@5 changes only marginally (1.02×). Larger gains appear in Contract Pass (1.14×) and Verified Accuracy (1.15×), suggesting that FSQ primarily helps distinguish executable and contract-compatible SOPs from already-retrieved candidates rather than simply increasing top-k recall.

Table 4: Construction and quality statistics of the SOP codebook across three trajectory sources.

Source	Traj.	Raw Cand.	Final SOPs	Compression↑	Avg. Support/ SOP ↑	Coverage↑	Verif. Yield ↑	Light Exec. ↑
AgentSkillOS	295	128	20	84.38	20.95	90.00	71.43	95.00
SkillsBench	291	76	10	86.84	3.90	89.02	71.43	90.00
SWE-bench	500	196	24	87.76	11.50	87.60	72.73	95.83
Codebook	**1,086**	**400**	**54**	**86.50**	**13.59**	**87.91**	**72.00**	**94.44**

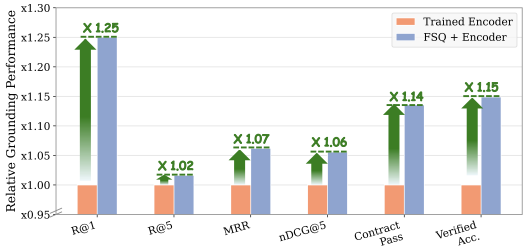
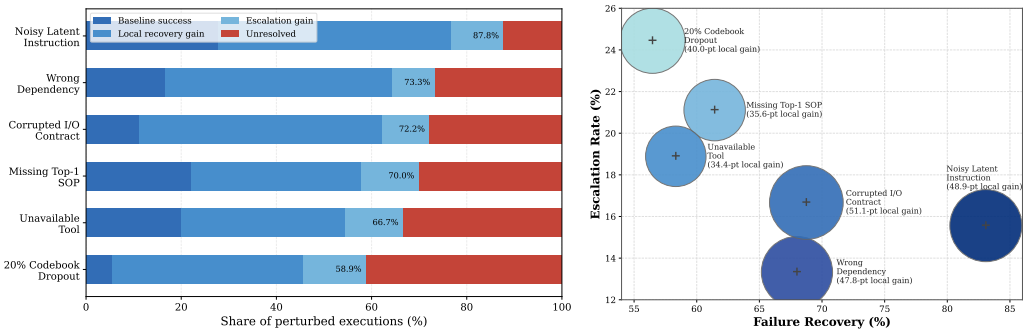


Figure 5: Relative grounding performance of FSQ+Encoder versus the trained encoder, normalized to 1.0, across retrieval and execution-aware metrics.

Observation 5: All major components contribute to the final performance, with SOP abstraction, runtime recovery, and contract verification producing the largest degradations when removed. As shown in Table 5, removing SOP abstraction causes the largest Macro Avg. drop of 8.95 points, followed by runtime recovery and contract verification with drops of 7.10 and 6.29 points, respectively. These components also consistently affect all three benchmarks, showing that the gains of the full framework are not driven by a single task domain. SOP normalization and deduplication provide a moderate 4.24-point gain, while removing the latent code reduces the Macro Avg. by 2.53 points. Disabling escalation produces the smallest but still consistent drop of 1.92 points, indicating that most execution failures can be handled by the core grounding and local recovery mechanisms, while escalation mainly provides complementary support for residual failures.

Observation 6: Lightweight local recovery resolves most runtime failures, while escalation mainly provides complementary recovery for harder residual cases. As shown in Fig. 6a, local recovery contributes the largest additional success share across all perturbation types, whereas



(a) Recovery decomposition across runtime perturbations. (b) Failure recovery versus escalation rate; bubble area denotes local-recovery gain.

Figure 6: Runtime robustness and recovery under different perturbation types.

Table 5: Component ablation of the SOP-centric framework. Drop denotes the absolute decrease in macro-average performance relative to the full system.

Variant	AgentSkillOS↑	SkillsBench↑	SWE-bench↑	Avg. ↑	Drop
Ours	**56.67**	**15.85**	**30.80**	**34.44**	–
w/o SOP Abstraction	43.33	8.54	24.60	25.49	8.95
w/o SOP Norm./Dedup.	50.00	12.20	28.40	30.20	4.24
w/o Latent Code	53.33	13.41	29.00	31.91	2.53
w/o Contract Verification	46.67	10.98	26.80	28.15	6.29
w/o Runtime Recovery	46.67	9.76	25.60	27.34	7.10
w/o Escalation	53.33	14.63	29.60	32.52	1.92

escalation provides a smaller but consistent gain. Full recovery reaches 87.8% success under noisy latent instructions and remains above 66% for dependency, contract, missing-SOP, and unavailable-tool perturbations. Codebook dropout is the most challenging case, with full recovery decreasing to 58.9%, indicating that missing procedural knowledge is harder to repair than locally detectable execution errors. Fig. 6b further reveals a clear difference in recovery difficulty across failure types. Noisy latent instructions achieve the highest failure recovery with relatively limited escalation, while wrong dependencies and corrupted I/O contracts also obtain large local-recovery gains. In contrast, missing SOPs and codebook dropout require more escalation while achieving lower recovery rates. These results show that local repair is particularly effective for structured and diagnosable failures, whereas knowledge-level failures benefit more from escalation.

5 Conclusion

We presented an SOP-centric framework that transforms prior execution trajectories into reusable, composable, and verifiable procedures. A learned SOP codebook enables latent dispatch and dynamic composition, allowing lightweight Executors to follow explicit constraints under Overseer guidance. Across AgentSkillOS, SkillsBench, and SWE-bench Verified, the framework improves macro-average performance by 33.1% over the strongest baseline. At the 9B scale, allocated compute decreases by 18.6% relative to Vanilla, while latent dispatch reduces serialized payload by 63.88× relative to natural-language plans without sacrificing task quality. Compute savings depend on executor capacity, and payload compression does not imply lower end-to-end latency. Ablations and perturbation studies highlight the importance of SOP abstraction, contract verification, and local recovery, with escalation providing complementary support. Overall, our results support reusable procedural knowledge as a foundation for effective and robust lightweight agent execution.
Limitation: The main limitation is that the framework's generalization, reliability, and end-to-end efficiency remain constrained by SOP coverage, verifier quality, and the current evaluation scope. See Appendix A.8 for additional limitations and discussion.

AI USE STATEMENT

Generative AI tools were used solely for language polishing, including improving grammar, clarity, fluency, and conciseness of author-written text. They were not used to generate research ideas, methods, experiments, data, analyses, results, or citations. All AI-assisted edits were reviewed and verified by the authors, who take full responsibility for the final content of this work.

REFERENCES

Sandhini Agarwal, Lama Ahmad, Jason Ai, Sam Altman, Andy Applebaum, Edwin Arbus, Rahul K Arora, Yu Bai, Bowen Baker, Haiming Bao, et al. gpt-oss-120b & gpt-oss-20b model card. *arXiv preprint arXiv:2508.10925*, 2025.

Antonis Antoniades, Albert Örwall, Kexun Zhang, Yuxi Xie, Anirudh Goyal, and William Wang. Swe-search: Enhancing software agents with monte carlo tree search and iterative refinement. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 64485–64515, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/a1e6783e4d739196cad3336f12d402bf-Paper-Conference.pdf.

Daman Arora, Atharv Sonwane, Nalin Wadhwa, Abhav Mehrotra, Saiteja Utpala, Ramakrishna Bairi, Aditya Kanade, and Nagarajan Natarajan. Masai: Modular architecture for software-engineering ai agents, 2024. URL https://arxiv.org/abs/2406.11638.

Xiaohe Bo, Zeyu Zhang, Quanyu Dai, Xueyang Feng, Lei Wang, Rui Li, Xu Chen, and Ji-Rong Wen. Reflective multi-agent collaboration based on large language models. 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. 138595–138631. Curran Associates, Inc., 2024. doi: 10.52202/079017-4397. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/fa54b0edce5eef0bb07654e8ee800cb4-Paper-Conference.pdf.

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.

Weize Chen, Yusheng Su, Jingwei Zuo, Cheng Yang, Chenfei Yuan, Chi-Min Chan, Heyang Yu, Yaxi Lu, Yi-Hsin Hung, Chen Qian, et al. Agentverse: Facilitating multi-agent collaboration and exploring emergent behaviors. In *International Conference on Learning Representations*, volume 2024, pp. 20094–20136, 2024a.

Yukang Chen, Shengju Qian, Haotian Tang, Xin Lai, Zhijian Liu, Song Han, and Jiaya Jia. Longlora: Efficient fine-tuning of long-context large language models. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun (eds.), *International Conference on Learning Representations*, volume 2024, pp. 8220–8238, 2024b. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/211ab571cc9f3802afa6ffff52ae3e5b-Paper-Conference.pdf.

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Sam Stevens, Boshi Wang, Huan Sun, and Yu Su. Mind2web: Towards a generalist agent for the web. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 28091–28114. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/5950bf290a1570ea401bf98882128160-Paper-Datasets_and_Benchmarks.pdf.

Victor Dibia, Jingya Chen, Gagan Bansal, Suff Syed, Adam Fourney, Erkang Zhu, Chi Wang, and Saleema Amershi. AUTOGEN STUDIO: A no-code developer tool for building and debugging multi-agent systems. In Delia Irazu Hernandez Farias, Tom Hope, and Manling Li

(eds.), *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 72–79, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-demo.8. URL https://aclanthology.org/2024.emnlp-demo.8/.

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. URL https://aclanthology.org/2026.findings-acl.866/.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Taicheng Guo, Xiuying Chen, Yaqi Wang, Ruidi Chang, Shichao Pei, Nitesh V. Chawla, Olaf Wiest, and Xiangliang Zhang. Large language model based multi-agents: A survey of progress and challenges. In Kate Larson (ed.), *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence, IJCAI-24*, pp. 8048–8057. International Joint Conferences on Artificial Intelligence Organization, 8 2024a.

Xudong Guo, Kaixuan Huang, Jiale Liu, Wenhui Fan, Natalia Vélez, Qingyun Wu, Huazheng Wang, Thomas L. Griffiths, and Mengdi Wang. Embodied llm agents learn to cooperate in organized teams. *IEEE Transactions on Computational Social Systems*, 13(2):2514–2530, 2026. doi: 10.1109/TCSS.2025.3637527.

Zhicheng Guo, Sijie Cheng, Hao Wang, Shihao Liang, Yujia Qin, Peng Li, Zhiyuan Liu, Maosong Sun, and Yang Liu. StableToolBench: Towards stable large-scale benchmarking on tool learning of large language models. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 11143–11156, Bangkok, Thailand, August 2024b. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-acl.664. URL https://aclanthology.org/2024.findings-acl.664/.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, DDL Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 10, 2022.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, zili wang, Steven Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. Metagpt: Meta programming for a multi-agent collaborative framework. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun (eds.), *International Conference on Learning Representations*, volume 2024, pp. 23247–23275, 2024a. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/6507b115562bb0a305f1958ccc87355a-Paper-Conference.pdf.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, 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 *The Twelfth International Conference on Learning Representations*, 2024b. URL https://openreview.net/forum?id=VtmBAGCN7o.

Sirui Hong, Yizhang Lin, Bang Liu, Bangbang Liu, Binhao Wu, Ceyao Zhang, Danyang Li, Jiaqi Chen, Jiayi Zhang, Jinlin Wang, Li Zhang, Lingyao Zhang, Min Yang, Mingchen Zhuge, Taicheng Guo, Tuo Zhou, Wei Tao, Robert Tang, Xiangtao Lu, Xiawu Zheng, Xinbing Liang, Yaying Fei, Yuheng Cheng, Yongxin Ni, Zhibin Gou, Zongze Xu, Yuyu Luo, and Chenglin Wu. Data interpreter: An LLM agent for data science. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 19796–19821, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-256-5. doi: 10.18653/v1/2025.findings-acl.1016. URL https://aclanthology.org/2025.findings-acl.1016/.

Qisheng Hu, Quanyu Long, and Wenya Wang. When continual learning moves to memory: A study of experience reuse in llm agents. *arXiv preprint arXiv:2604.27003*, 2026.

Xu Huang, Weiwen Liu, Xiaolong Chen, Xingmei Wang, Hao Wang, Defu Lian, Yasheng Wang, Ruiming Tang, and Enhong Chen. Understanding the planning of llm agents: A survey. *arXiv preprint arXiv:2402.02716*, 2024.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues? In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun (eds.), *International Conference on Learning Representations*, volume 2024, pp. 54107–54157, 2024a. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/edac78c3e300629acfe6cbe9ca88fb84-Paper-Conference.pdf.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. SWE-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations*, 2024b. URL https://openreview.net/forum?id=VTF8yNQM66.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

Tushar Khot, Harsh Trivedi, Matthew Finlayson, Yao Fu, Kyle Richardson, Peter Clark, and Ashish Sabharwal. Decomposed prompting: A modular approach for solving complex tasks. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=_nGgzQjzaRy.

Sehoon Kim, Suhong Moon, Ryan Tabrizi, Nicholas Lee, Michael W. Mahoney, Kurt Keutzer, and Amir Gholami. An LLM compiler for parallel function calling. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 24370–24391. PMLR, 21–27 Jul 2024. URL https://proceedings.mlr.press/v235/kim24y.html.

LangChain Team. Plan-and-execute agents. LangChain Blog, May 2023. [Online]. Available: https://www.langchain.com/blog/plan-and-execute-agents. Accessed: Aug. 27, 2026.

Ao Li, Yuexiang Xie, Songze Li, Fugee Tsung, Bolin Ding, and Yaliang Li. Agent-oriented planning in multi-agent systems. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 19495–19517, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/31610e68fe41a62e460e044216a10766-Paper-Conference.pdf.

Guohao Li, Hasan Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. Camel: Communicative agents for "mind" exploration of large language model society. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 51991–52008. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/a3621ee907def47c1b952ade25c67698-Paper-Conference.pdf.

Hao Li, Chunjiang Mu, Jianhao Chen, Siyue Ren, Zhiyao Cui, Yiqun Zhang, Lei Bai, and Shuyue Hu. Organizing, orchestrating, and benchmarking agent skills at ecosystem scale, 2026a. URL https://arxiv.org/abs/2603.02176.

Xiangyi Li, Yimin Liu, Wenbo Chen, Bingran You, Zonglin Di, Yifeng He, Shenghan Zheng, Kyoung Whan Choe, Jiankai Sun, Shuyi Wang, Chujun Tao, Binxu Li, Xuandong Zhao, Hejia Geng, Xiaojun Wu, Junwei Zhou, Xiaokun Chen, Hanwen Xing, Yubo Li, Qunhong Zeng, Di Wang, Yuanli Wang, Roey Ben Chaim, Penghao Jiang, Haotian Shen, Luyang Kong, Xinyi Liu, Runhui Wang, Xuanqing Liu, Jiachen Li, Xin Lan, Yueqian Lin, Wengao Ye, Junwei He, Songlin Li, Yue Zhang, Yipeng Gao, Yijiang Li, Ze Ma, Liqiang Jing, Tianyu Wang, Kaixin Li, Yiqi

Xue, Haoran Lyu, Yizhuo He, Yuchen Tian, Shutong Wu, Bowei Wang, Yixuan Gao, Bo Chen, Litong Liu, Sikai Cheng, Jiajun Bao, Shuaicheng Tong, Shuwen Xu, Terry Yue Zhuo, Tinghan Ye, Qi Qi, Miao Li, Longtai Liao, Zelin Tan, Chang Shi, Xilin Tang, Srinath Tankasala, Boqin Yuan, Yaoyao Qian, Jianhong Tu, Chenguang Wang, Yizhou Sun, Wei Wang, Aaron Taylor, Ziyue Yang, Changkun Guan, Zhikang Dong, Xinyu Zhang, Steven Dillmann, Han chung Lee, and Dawn Song. Skillsbench: Benchmarking how well agent skills work across diverse tasks, 2026b. URL https://arxiv.org/abs/2602.12670.

Yunxuan Li, Yibing Du, Jiageng Zhang, Le Hou, Peter Grabowski, Yeqing Li, and Eugene Ie. Improving multi-agent debate with sparse communication topology. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 7281–7294, Miami, Florida, USA, November 2024a. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-emnlp.427. URL https://aclanthology.org/2024.findings-emnlp.427/.

Zelong Li, Wenyue Hua, Hao Wang, He Zhu, and Yongfeng Zhang. Formal-llm: Integrating formal language and natural language for controllable llm-based agents. *arXiv preprint arXiv:2402.00798*, 2024b.

Zihan Liao, Jun Wang, Hang Yu, Lingxiao Wei, Jianguo Li, Jun Wang, and Wei Zhang. E2LLM: Encoder elongated large language models for long-context understanding and reasoning. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 19201–19230, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.970. URL https://aclanthology.org/2025.emnlp-main.970/.

Wei Liu, Chenxi Wang, Yifei Wang, Zihao Xie, Rennai Qiu, Yufan Dang, Zhuoyun Du, Weize Chen, Cheng Yang, and Chen Qian. Autonomous agents for collaborative task under information asymmetry. 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. 2734–2765. Curran Associates, Inc., 2024a. doi: 10.52202/079017-0090. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/0534abc9e6db91683d82186ef0d68202-Paper-Conference.pdf.

Weiwen Liu, Xu Huang, Xingshan Zeng, xinlong hao, Shuai Yu, Dexun Li, Shuai Wang, Weinan Gan, Zhengying Liu, Yuanqing Yu, Zezhong WANG, Yuxian Wang, Wu Ning, Yutai Hou, Bin Wang, Chuhan Wu, Wang Xinzhi, Yong Liu, Yasheng Wang, Duyu Tang, Dandan Tu, Lifeng Shang, Xin Jiang, Ruiming Tang, Defu Lian, Qun Liu, and Enhong Chen. Toolace: Winning the points of llm function calling. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 41359–41381, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/663865ea167425c6c562cb0b6bcf76c7-Paper-Conference.pdf.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. Agentbench: Evaluating llms as agents. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun (eds.), *International Conference on Learning Representations*, volume 2024, pp. 52989–53046, 2024b. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/e9df36b21ff4ee211a8b71ee8b7e9f57-Paper-Conference.pdf.

Yimeng Liu, Misha Sra, Jeevana Priya Inala, and Chenglong Wang. Reuseit: Synthesizing reusable ai agent workflows for web automation. In *Proceedings of the 31st International Conference on Intelligent User Interfaces*, IUI '26, pp. 885–908. ACM, March 2026. doi: 10.1145/3742413.3789083. URL http://dx.doi.org/10.1145/3742413.3789083.

Jinghao Luo, Yuchen Tian, Chuxue Cao, Ziyang Luo, Hongzhan Lin, Kaixin Li, Chuyi Kong, Ruichao Yang, and Jing Ma. From storage to experience: A survey on the evolution of llm agent memory mechanisms. *Preprints*, January 2026. doi: 10.20944/preprints202601.0618.v2. URL https://doi.org/10.20944/preprints202601.0618.v2.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 46534–46594. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/91edff07232fb1b55a505a9e9f6c0ff3-Paper-Conference.pdf.

Fabian Mentzer, David Minnen, Eirikur Agustsson, and Michael Tschannen. Finite scalar quantization: Vq-vae made simple. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun (eds.), *International Conference on Learning Representations*, volume 2024, pp. 51772–51783, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/e2dd53601de57c773343a7cdf09fae1c-Paper-Conference.pdf.

Subhrangshu Nandi, Arghya Datta, Nikhil Vichare, Indranil Bhattacharya, Huzefa Raja, Jing Xu, Shayan Ray, Giuseppe Carenini, Abhi Srivastava, Aaron Chan, et al. Sop-bench: Complex industrial sops for evaluating llm agents. *arXiv preprint arXiv:2506.08119*, 2025.

NVIDIA. Nemotron 3 Nano: Open, efficient mixture-of-experts hybrid Mamba-Transformer model for Agentic reasoning, 2025. URL https://arxiv.org/abs/2512.20848. Technical report.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 27730–27744. Curran Associates, Inc., 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/b1efde53be364a73914f58805a001731-Paper-Conference.pdf.

Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. Gorilla: Large language model connected with massive apis. 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. 126544–126565. Curran Associates, Inc., 2024. doi: 10.52202/079017-4020. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/e4c61f578ff07830f5c37378dd3ecb0d-Paper-Conference.pdf.

Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, Juyuan Xu, Dahai Li, Zhiyuan Liu, and Maosong Sun. ChatDev: Communicative agents for software development. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 15174–15186, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.810. URL https://aclanthology.org/2024.acl-long.810/.

Shuofei Qiao, Runnan Fang, Ningyu Zhang, Yuqi Zhu, Xiang Chen, Shumin Deng, Yong Jiang, Pengjun Xie, Fei Huang, and Huajun Chen. Agent planning with world knowledge model. 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. 114843–114871. Curran Associates, Inc., 2024. doi: 10.52202/079017-3646. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/d032263772946dd5026e7f3cd22bce5b-Paper-Conference.pdf.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, dahai li, Zhiyuan Liu, and Maosong Sun. Toolllm: Facilitating large language models to master 16000+ real-world apis. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun (eds.), *International Conference on Learning Representations*, volume 2024, pp. 9695–9717, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/28e50ee5b72e90b50e7196fde8ea260e-Paper-Conference.pdf.

Qwen Team. Qwen3.5: Towards native multimodal agents, February 2026. URL https://qwen.ai/blog?id=qwen3.5.

Nils Reimers and Iryna Gurevych. Sentence-BERT: Sentence embeddings using Siamese BERT-networks. In Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan (eds.), *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, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1410. URL https://aclanthology.org/D19-1410/.

Stephen Robertson and Hugo Zaragoza. The probabilistic relevance framework: Bm25 and beyond. *Foundations and Trends in Information Retrieval*, 4(1-2):1–174, 09 2009. ISSN 1554-0669. doi: 10.1561/1500000019. URL https://doi.org/10.1561/1500000019.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessi, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 68539–68551. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/d842425e4bf79ba039352da0f658a906-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. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/1b44b878bb782e6954cd888628510e90-Paper-Conference.pdf.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*, 2024.

Hongjin Su, Shizhe Diao, Ximing Lu, Mingjie Liu, Jiacheng Xu, Xin Dong, Yonggan Fu, Peter Belcak, Hanrong Ye, Hongxu Yin, Yi Dong, Evelina Bakhturina, Tao Yu, Yejin Choi, Jan Kautz, and Pavlo Molchanov. Toolorchestra: Elevating intelligence via efficient model and tool orchestration, 2025. URL https://arxiv.org/abs/2511.21689.

Gemma Team. Gemma 4 technical report, 2026. URL https://arxiv.org/abs/2607.02770.

Gemma Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, et al. Gemma 3 technical report. *arXiv preprint arXiv:2503.19786*, 2025.

Dat Tran and Douwe Kiela. Single-agent llms outperform multi-agent systems on multi-hop reasoning under equal thinking token budgets. *arXiv preprint arXiv:2604.02460*, 2026.

Karthik Valmeekam, Matthew Marquez, Sarath Sreedharan, and Subbarao Kambhampati. On the planning abilities of large language models - a critical investigation. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 75993–76005. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/efb2072a358cefb75886a315a6fcf880-Paper-Conference.pdf.

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. *Transactions on Machine Learning Research*, 2024a. ISSN 2835-8856. URL https://openreview.net/forum?id=ehfRiF0R3a.

Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, Wayne Xin Zhao, Zhewei Wei, and Jirong Wen. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6):186345, mar 2024b. ISSN 2095-2236. doi: 10.1007/s11704-024-40231-1. URL https://doi.org/10.1007/s11704-024-40231-1.

Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. Executable code actions elicit better LLM agents. In *Forty-first International Conference on Machine Learning*, 2024c. URL https://openreview.net/forum?id=jJ9BoXAfFa.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, Hoang Tran, Fuqiang Li, Ren Ma, Mingzhang Zheng, Bill Qian, Daniel Shao, Niklas Muennighoff, Yizhe Zhang, Binyuan Hui, Junyang Lin, Robert Brennan, Hao Peng, Heng Ji, and Graham Neubig. Openhands: An open platform for ai software developers as generalist agents. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 65882–65919, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/a4b6ad6b48850c0c331d1259fc66a69c-Paper-Conference.pdf.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 24824–24837. Curran Associates, Inc., 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/9d5609613524ecf4f15af0f7b31abca4-Paper-Conference.pdf.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, et al. Autogen: Enabling next-gen llm applications via multi-agent conversation. *arXiv preprint arXiv:2308.08155*, 2023.

Chunqiu Steven Xia, Yinlin Deng, Soren Dunn, and Lingming Zhang. Demystifying llm-based software engineering agents. *Proc. ACM Softw. Eng.*, 2(FSE), June 2025. doi: 10.1145/3715754. URL https://doi.org/10.1145/3715754.

Chengxing Xie, Bowen Li, Chang Gao, He Du, Wai Lam, Difan Zou, and Kai Chen. SWE-fixer: Training open-source LLMs for effective and efficient GitHub issue resolution. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 1123–1139, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-256-5. doi: 10.18653/v1/2025.findings-acl.62. URL https://aclanthology.org/2025.findings-acl.62/.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. 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. 52040–52094. Curran Associates, Inc., 2024. doi: 10.52202/079017-1650. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/5d413e48f84dc61244b6be550f1cd8f5-Paper-Datasets_and_Benchmarks_Track.pdf.

Zidi Xiong, Yuping Lin, Wenya Xie, Pengfei He, Zirui Liu, Jiliang Tang, Himabindu Lakkaraju, and Zhen Xiang. How memory management impacts LLM agents: An empirical study of experience-following behavior. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 623–645, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-390-6. URL https://aclanthology.org/2026.acl-long.27/.

Wei Xu, Zhaohui Yang, Derrick Wing Kwan Ng, Robert Schober, H. Vincent Poor, Zhaoyang Zhang, and Xiaohu You. A new path to integrated learning and communication (ilac): Large ai models leveraging hyperdimensional computing. *IEEE Transactions on Communications*, 74:4948–4973, 2026. doi: 10.1109/TCOMM.2026.3662396.

John Yang, Carlos Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated

software engineering. 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. 50528–50652. Curran Associates, Inc., 2024. doi: 10.52202/079017-1601. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/5a7c947568c1b1328ccc5230172e1e7c-Paper-Conference.pdf.

Liming Yang, Junyu Luo, Xuanzhe Liu, Yiling Lou, and Zhenpeng Chen. BAMAS: Structuring Budget-Aware Multi-Agent Systems. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 29802–29810. AAAI Press, 2026. doi: 10.1609/aaai.v40i35.40226.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 11809–11822. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/271db9922b8d1f4dd7aaef84ed5ac703-Paper-Conference.pdf.

Anbang Ye, Qianran Ma, Jia Chen, Muqi Li, Tong Li, Fujiao Liu, Siqi Mai, Meichen Lu, Haitao Bao, and Yang You. Sop-agent: Empower general purpose ai agent with domain-specific sops. *arXiv preprint arXiv:2501.09316*, 2025.

Ling Yue, Kushal Raj Bhandari, Ching-Yun Ko, Dhaval Patel, Shuxin Lin, Nianjun Zhou, Jianxi Gao, Pin-Yu Chen, and Shaowu Pan. From static templates to dynamic runtime graphs: A survey of workflow optimization for llm agents, 2026. URL https://arxiv.org/abs/2603.22386.

Yuntong Zhang, Haifeng Ruan, Zhiyu Fan, and Abhik Roychoudhury. Autocoderover: Autonomous program improvement. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*, ISSTA 2024, pp. 1592–1604, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400706127. doi: 10.1145/3650212.3680384. URL https://doi.org/10.1145/3650212.3680384.

Andrew Zhao, Daniel Huang, Quentin Xu, Maxwell Lin, Yong-Jin Liu, and Guibin Huang. ExpeL: LLM agents are experiential learners. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(17):19632–19642, 2024. doi: 10.1609/aaai.v38i17.29936.

Longtao Zheng, Rundong Wang, Xinrun Wang, and Bo An. Synapse: Trajectory-as-exemplar prompting with memory for computer control. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun (eds.), *International Conference on Learning Representations*, volume 2024, pp. 19036–19066, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/52f050499cf82fa8efb588e263f6f3a7-Paper-Conference.pdf.

Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. MemoryBank: Enhancing Large Language Models with Long-Term Memory. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 19724–19731. AAAI Press, 2024. doi: 10.1609/aaai.v38i17.29946.

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 B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun (eds.), *International Conference on Learning Representations*, volume 2024, 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 Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 8580–8622, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.421. URL https://aclanthology.org/2025.acl-long.421/.

A Appendix

A.1 Fine-Grained Performance Across Task Categories and Repositories

Table A.1: Fine-grained performance across task categories and repositories. Red superscripts on Ours denote absolute score improvements over Vanilla.

Benchmark	Category / Repository	Vanilla	Skills	Ours
AgentSkillOS	Data Computation	32.11	42.13	**61.78** $^{+29.67}$
	Document Creation	34.99	44.52	**56.88** $^{+21.89}$
	Motion & Video	23.63	35.42	**48.31** $^{+24.68}$
	Visual Creation	27.78	39.60	**54.70** $^{+26.92}$
	Web Interaction	35.64	42.48	**61.68** $^{+26.04}$
SkillsBench	Software Engineering	6.15	12.47	**20.52** $^{+14.37}$
	Office & White Collar	4.51	10.86	**13.93** $^{+9.42}$
	Natural Science	3.24	9.29	**12.15** $^{+8.91}$
	Media & Content Production	4.12	10.73	**15.51** $^{+11.39}$
	Cybersecurity	5.68	13.51	**19.09** $^{+13.41}$
	Finance	5.20	11.62	**15.33** $^{+10.13}$
	Robotics	3.03	9.28	**17.12** $^{+14.09}$
	Manufacturing	2.41	8.83	**15.88** $^{+13.47}$
	Energy	3.08	9.08	**15.15** $^{+12.07}$
	Mathematics	5.36	**10.80**	9.90 $^{+4.54}$
	Healthcare	4.38	**9.24**	8.90 $^{+4.52}$
SWE Verified	django/django	24.63	27.95	**30.86** $^{+6.23}$
	sympy/sympy	18.34	22.42	**30.93** $^{+12.59}$
	sphinx-doc/sphinx	20.18	24.77	**29.82** $^{+9.64}$
	matplotlib/matplotlib	22.90	26.35	**34.02** $^{+11.12}$
	scikit-learn/scikit-learn	21.05	24.48	**28.39** $^{+7.34}$
	astropy/astropy	19.37	23.97	**32.35** $^{+12.98}$
	pydata/xarray	23.60	26.72	**32.12** $^{+8.52}$
	pytest-dev/pytest	20.12	23.44	**30.38** $^{+10.26}$
	pylint-dev/pylint	18.54	20.83	**23.46** $^{+4.92}$
	Other repositories	16.53	20.37	**31.29** $^{+14.76}$

Observation A.1: SOP guidance provides broad but task-dependent gains. As shown in Table A.1, the improvements from SOP guidance are distributed across task categories and repositories rather than being driven by a single task group. Ours improves over Vanilla in all 26 fine-grained groups, with absolute gains ranging from 4.52 to 29.67 percentage points, and outperforms Skills in 24 of them. The magnitude of the improvement varies substantially across tasks. AgentSkillOS shows consistently large gains of 21.89–29.67 points, while SkillsBench and SWE-bench Verified exhibit broader variation, with gains of 4.52–14.37 and 4.92–14.76 points, respectively. Mathematics and Healthcare show the smallest gains and are the only categories where Skills slightly outperforms Ours. These results indicate that SOP guidance generalizes across domains, while the magnitude of its benefit remains dependent on the underlying task structure.

A.2 Held-Out Task Transfer with a Frozen SOP Codebook

Observation A.2: Frozen structured SOPs transfer effectively to held-out tasks. As shown in Table A.2, the frozen SOP codebook continues to provide effective execution guidance on tasks excluded from codebook construction. Structured SOP improves the macro-average score from 18.37 with no memory to 30.57, with consistent gains across AgentSkillOS, SkillsBench, and SWE-bench. More importantly, Structured SOP outperforms Raw Trajectory Retrieval under the same historical experience, improving the three benchmarks by 11.20, 4.50, and 3.50 points, respectively, and raising the macro average from 24.17 to 30.57. This result indicates that the improvement does not depend solely on access to prior trajectories; organizing historical experience into reusable SOPs provides more effective guidance for previously unseen evaluation tasks.

Table A.2: Performance on held-out evaluation tasks that are excluded from codebook construction, with the SOP codebook frozen during evaluation. Raw Trajectory Retrieval and Structured SOP use the same historical experience, while Structured SOP replaces direct trajectory retrieval with the frozen SOP codebook.

Representation	AgentSkillOS	SkillsBench	SWE-bench	Macro Avg.
No Memory	29.20	4.30	21.60	18.37
Traj. Retrieval	38.50	8.90	25.10	24.17
Ours	**49.70**	**13.40**	**28.60**	**30.57**

A.3 Transfer Across Executor Model Families

Table A.3: Performance across different executor model families. Scores report macro-average performance across the three benchmarks. Red superscripts on Ours denote absolute score changes relative to Vanilla, and GPU-s/Task reports the execution cost per task.

Executor	Vanilla	Skills	Ours	GPU-s/Task
DeepSeek-R1-Distill-Llama-8B	19.36	22.71	**26.84** $^{+7.48}$	281.64
DeepSeek-R1-Distill-Qwen-14B	25.17	28.62	**32.91** $^{+7.74}$	407.28
gemma-3-12b-it	21.63	24.38	**28.76** $^{+7.13}$	351.42
gemma-3-27b-it	28.54	30.96	**33.87** $^{+5.33}$	692.75
NVIDIA-Nemotron-3-Nano-30B-A3B-FP8	23.82	27.11	**30.46** $^{+6.64}$	324.18
Nemotron-Orchestrator-8B	29.76	31.84	**35.12** $^{+5.36}$	294.67
gpt-oss-20b	30.88	32.41	**35.36** $^{+4.48}$	368.94
gpt-oss-120b	36.72	36.31	**39.86** $^{+3.14}$	668.53
gemma-4-E4B-it	14.83	**15.21**	14.19 $^{-0.64}$	247.86
gemma-4-31B-it	30.17	32.06	**34.92** $^{+4.75}$	738.21

Observation A.3: SOP guidance transfers across executor families, but its benefit remains executor-dependent. As shown in Table A.3, SOP guidance produces positive gains over Vanilla for 9 of the 10 evaluated executors while keeping the procedural knowledge and execution mechanism unchanged. The improvements span multiple model families, including DeepSeek (Guo et al., 2025), Gemma3 (Team et al., 2025),Gemma4 (Team, 2026), Nemotron (Su et al., 2025),Nemotron 3 Nano (NVIDIA, 2025), and GPT-OSS (Agarwal et al., 2025), with absolute gains ranging from 3.14 to 7.74 points among the improved executors. Ours also outperforms the Skills Baseline on the same nine executors. The only exception is gemma-4-E4B-it, where Ours decreases by 0.64 points relative to Vanilla. These results show that the benefit of SOP guidance is not tied to the executor family used in the main experiments, while the magnitude and even presence of the gain remain dependent on the underlying executor.

A.4 Effect of Dependency-Guided SOP Composition

Table A.4: Effect of explicit SOP composition under matched procedural knowledge and execution budgets. SOP Set Prompting uses the same SOP set without explicit dependency organization, whereas Dependency-Guided Composition structures procedures by input-output dependencies before execution. Higher Score and Task Success are better; lower Invalid-Input Calls and GPU-s/Task are better.

Task	Organization	Score	Task Success (%)	Invalid-Input Calls (%)	GPU-s/Task
SOP	SOP Set Prompting	31.6	27.5	11.8	**268.7**
	Dependency-Guided Composition	**36.9**	**33.3**	**7.2**	272.4
Complex	SOP Set Prompting	22.4	17.0	19.6	**296.8**
	Dependency-Guided Composition	**31.7**	**26.0**	**10.4**	301.5

Observation A.4: Explicit dependency composition improves multi-procedure execution with little additional cost. As shown in Table A.4, explicitly organizing the same SOP knowledge by input–output dependencies improves execution over simply providing the SOP set. On SOP tasks, Dependency-Guided Composition increases the score from 31.6 to 36.9 and task success from 27.5% to 33.3%, while reducing invalid-input calls from 11.8% to 7.2%. The advantage becomes larger on Complex tasks, where the score increases from 22.4 to 31.7, task success rises from 17.0% to 26.0%, and invalid-input calls decrease from 19.6% to 10.4%. These gains require only small increases in execution cost, from 268.7 to 272.4 GPU-s/Task and from 296.8 to 301.5 GPU-s/Task, respectively. The larger improvement on Complex tasks indicates that explicit dependency organization becomes more valuable when execution involves multiple interdependent procedures, with fewer unmet input requirements consistent with more reliable coordination across dependent steps.

A.5 Scaling with SOP Construction Data

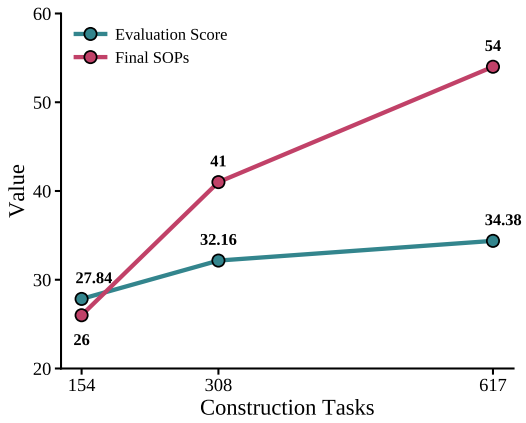
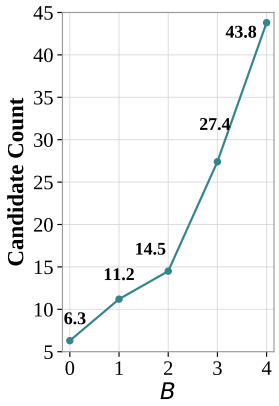


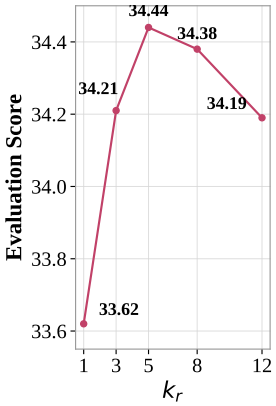
Figure A.1: Scaling behavior with increasing construction tasks. Evaluation performance improves as the construction set grows, while the resulting SOP codebook expands from 26 to 54 procedures.

Observation A.5: Additional construction experience improves transfer performance with diminishing returns. As shown in Figure A.1, increasing the number of independent construction tasks consistently improves performance on the fixed held-out evaluation set. The evaluation score increases from 27.84 to 32.16 and 34.38 as the number of construction tasks grows from 154 to 308 and 617, while the resulting SOP codebook expands from 26 to 41 and 54 procedures. This trend shows that additional construction experience can be converted into reusable procedural knowledge that benefits unseen tasks. The improvement, however, becomes progressively smaller: the first increase in construction experience yields a 4.32-point gain, whereas the subsequent larger increase yields only 2.22 points. This diminishing gain indicates that the benefit of additional experience gradually saturates as procedural knowledge accumulates.

A.6 Sensitivity of SOP Grounding to Retrieval Parameters



(a) Candidate-set size under different Hamming-radius budgets B. Larger radii progressively expand the number of SOP candidates considered during grounding.



(b) Evaluation performance under different retrieval sizes k_r. Performance improves as k_r increases to 5 and remains relatively stable for larger candidate sets.

Figure A.2: Sensitivity of latent SOP grounding to retrieval parameters. Increasing B expands the candidate pool, while evaluation performance peaks at a moderate retrieval size k_r and changes only slightly thereafter.

Observation A.6: SOP grounding is stable within a moderate retrieval range, while larger candidate sets provide limited additional benefit. As shown in Fig. A.2, increasing the grounding radius substantially expands the candidate pool, from 6.3 candidates at $B = 0$ to 43.8 at $B = 4$. Evaluation performance improves as the retrieval size increases from $k_r = 1$ to $k_r = 5$, reaching 34.44, but remains nearly unchanged or slightly decreases for larger retrieval sets, with scores of 34.38 and 34.19 at $k_r = 8$ and $k_r = 12$, respectively. These results show that grounding remains relatively stable over a moderate retrieval range, while continually enlarging the candidate set does not necessarily improve execution quality and increases the amount of candidate processing required during grounding.

A.7 Runtime Recovery Case Studies

Table A.5: Two logged execution cases illustrating successful runtime recovery and an unresolved failure. Case A is resolved through local repair without escalation, whereas Case B remains unsuccessful after local recovery and subsequent escalation.

Case	Failure Trigger	Recovery Action	Escalation	Final Outcome
A: data_computation_task6 (seed 42)	Native checks detect that *integrals.json* and five PNG files are missing; fallback still fails to produce all five PNGs.	Retry and fallback, followed by three ordered whole-script repair and rerun attempts: legend argument, bar-axis length, and the remaining legend argument.	Not performed in the recorded case (count = 0).	**100/100**; all 10 configured checks pass and strict success = true.
B: document_creation_task4 (seed 43)	Saved native checks fail the minimum 12-page requirement and the landscape-orientation requirement.	Retry and fallback; a local rewrite is attempted, followed by a width change from 960 → 69120, but the result is still rejected. The teacher later attempts a ReportLab replacement.	Actual Qwen3.5-9B continuation with four actions; the replacement is not subsequently executed.	**10.53/100**; passed = false and strict success = false. Page count and orientation still fail, and the reported page metadata remains unconfirmed.

Observation A.7: Runtime validation enables targeted recovery from intermediate execution failures. As shown in Table A.5, runtime checks expose intermediate outputs that violate task requirements and provide concrete failure signals for subsequent recovery. In Case A, missing artifacts remain detectable after fallback, after which the executor performs targeted repairs to the script arguments and plotting configuration. These repairs eventually satisfy all 10 configured checks and achieve strict task success. Case B similarly triggers local rewriting and parameter adjustment after page-count and orientation violations are detected, although the recorded recovery does not resolve the task. These cases show that validation feedback can guide localized changes to actions, parameters, and execution choices rather than treating a failed attempt as an opaque outcome, while also illustrating that local recovery is not guaranteed to succeed for every violation.

A.8 Limitations and Discussion

Procedural transfer remains dependent on the task distribution and the underlying Executor. Although our evaluation includes held-out tasks and multiple model families, it remains bounded by three benchmarks and their configured tool environments. The negative gains observed for some Executors also indicate that procedural guidance does not benefit every model equally. **Discussion: Structured SOPs support procedural reuse beyond individual task instances and the Executor family used in the main study.** A frozen SOP codebook outperforms raw trajectory retrieval on held-out tasks under the same historical experience, while cross-family experiments demonstrate benefits beyond the Qwen3.5 Executors. Together, these findings support structured procedural knowledge as a transferable interface between planning and execution, rather than merely a record of previously successful actions.

Procedural coverage depends on construction experience, while scalability to substantially larger and evolving codebooks remains unverified. The framework derives its procedures from teacher-generated trajectories, and missing procedural knowledge is harder to compensate for than locally detectable execution errors in the tested perturbations. Moreover, the construction and grounding studies evaluate a codebook of 54 SOPs rather than a large, continuously updated repos-

itory. **Discussion: Abstraction and dependency-guided composition transform historical experience into compact procedural knowledge that supports reuse and coordinated execution.** The compact codebook retains broad task coverage and high lightweight executability within the evaluated settings, demonstrating that heterogeneous trajectories can be consolidated into reusable procedural units. Furthermore, explicit composition improves execution over simply providing the same SOP set, with larger benefits on complex tasks. The contribution therefore lies not only in retaining historical knowledge, but also in organizing it into executable structures that coordinate interdependent procedures.

Execution reliability remains conditional on contract completeness, verifier correctness, and the recoverability of detected failures. Passing encoded checks does not establish correctness beyond their scope, and the unresolved document-creation case illustrates that recovery attempts may leave task requirements unsatisfied. Controlled perturbations likewise cover selected failure modes rather than all failures encountered in changing environments. **Discussion: Contract-guided verification and local recovery make execution failures diagnosable and actionable.** The framework connects explicit requirements, observable violations, and corrective actions in a closed-loop execution process. Ablations support the importance of verification and recovery, while the successful recovery case demonstrates how concrete failure signals guide targeted repairs without escalation. Local recovery also contributes the largest additional success share across the tested perturbations. These findings establish the practical value of using procedural constraints to guide corrective execution rather than treating unsuccessful attempts as opaque outcomes.

Communication compression does not by itself establish lower end-to-end cost or lifecycle efficiency. Latent dispatch relies on a compatible codebook and task state, while grounding, verification, and recovery introduce additional work. Compute savings vary across Executors and workloads, and reduced communication can coexist with increased executor-token consumption and latency. Offline construction and codebook synchronization must also be considered when assessing deployment costs. **Discussion: Procedural reuse can improve the performance–compute tradeoff, while compact latent dispatch reduces communication without sacrificing task quality in the evaluated setting.** At the 9B Executor scale, SOP guidance improves task performance while reducing allocated GPU-seconds. In the separate dispatch comparison, latent instructions substantially reduce serialized payload while preserving execution quality. These findings demonstrate complementary benefits from organizing execution around reusable procedures and avoiding repeated transmission of full procedural descriptions, without requiring every component of execution cost to decrease simultaneously.

Future Work. We will evaluate broader task distributions, tool environments, and Executor families to characterize when procedural transfer remains effective. We will also diversify construction experience and investigate incremental codebook maintenance to address coverage gaps and assess retrieval and composition at larger scales. Stronger contracts, more comprehensive verifiers, and evaluation under naturally occurring and combined failures will further test and improve recovery reliability. Finally, lifecycle efficiency studies will account for offline construction, synchronization, and online execution overheads under different reuse and update frequencies, clarifying when the observed performance and communication benefits translate into sustained deployment savings.