$\Delta\tau$-BENCH: PUSHING AGENTS TOWARD A MOVING CAPABILITY FRONTIER

Anonymous authors
Paper under double-blind review

ABSTRACT

In this paper, we introduce $\Delta\tau$-Bench, an executable agent benchmark that models the structural evolution of large tool ecosystems. In $\Delta\tau$-Bench, independently evolving services continually reshape how tasks can be accomplished: familiar tools may disappear or be replaced, while more efficient alternatives emerge amid overlapping functionality and noisy documentation. Against this moving capability frontier, we evaluate whether agents can proactively discover efficient tool-use paths and promptly revise their accumulated procedural knowledge, bringing together demands that existing benchmarks fail to examine. $\Delta\tau$-Bench comprises 1,588 validated multi-turn tasks across 42 domains and six successive ecosystem states generated through capability-preserving transformations. Our evaluation shows that even the strongest evaluated model or harness achieves only 42.3% task success, while initial gains from existing skill-learning methods can not consistently translate into better adaptation as the ecosystem evolves. These findings motivate STEWARD, an ecosystem manager that jointly manages tools and procedural knowledge through incremental skill maintenance and dynamic provisioning during task execution. Experiments show that STEWARD sustains higher task success across ecosystem changes with lower skill-maintenance overhead, while improving both recovery from disrupted workflows and adoption of more efficient alternatives.

1 INTRODUCTION

Large Language Model (LLM) agents are reshaping how users interact with software, enabling them to delegate goals through natural language and have actions carried out on their behalf (Qin et al., 2025b; Wang et al., 2023; Xi et al., 2025; Dong et al., 2026b; Guo et al., 2024). Through tool interfaces, agents can access external services and coordinate multi-step operations to fulfill these goals (Qin et al., 2024; Su et al., 2025; Qiao et al., 2024b; Dong et al., 2026a; Fang et al., 2026; Qian et al., 2025a). This opens the prospect of a unified service interface: users specify what they need, while agents take responsibility for finding and orchestrating the capabilities required to deliver it.

As shown in Fig. 1, imagine a large company deploying an agent as a unified entry point to its services. Behind this single interface lies a large tool ecosystem, with individual departments independently maintaining and updating their own tools. Overlapping functionality and noisy or inaccurate descriptions make this catalog difficult to navigate, while ongoing updates continually reshape the available paths through it. As departments retire or replace tools, established workflows may break, forcing the agent to search the broader ecosystem for alternatives. Meanwhile, newly added tools may enable more efficient workflows, but these opportunities can remain hidden among the many services while familiar procedures continue to succeed. Keeping pace therefore requires the agent to actively discover how the company's changing capabilities can be used and revise its procedural knowledge accordingly. Otherwise, successful task execution can conceal a growing gap between what the ecosystem offers and what the agent has learned to accomplish with it.

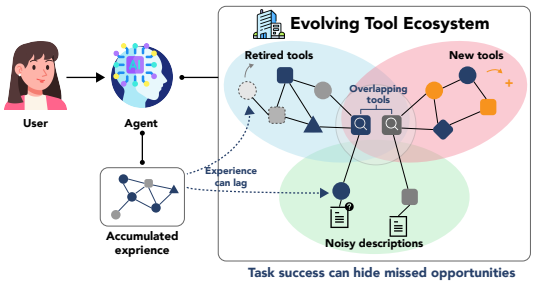


Figure 1: **Agent adaptation in evolving tool ecosystems.** Tool updates can invalidate learned workflows or introduce better alternatives, while overlapping tools and noisy descriptions hinder discovery.

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.

Evaluating this challenge requires comparing an agent's behavior with what the current ecosystem makes possible for the same task. Existing benchmarks assess large-scale tool retrieval and stateful execution (Shi et al., 2025; Qin et al., 2025a; Trivedi et al., 2024; Yao et al., 2024; Li et al., 2025), cost-aware replanning under changing execution conditions (Liu et al., 2026e; Valmeekam et al., 2023; Xie et al., 2024; Kokel et al., 2025), and robustness to tool evolution (Liu et al., 2026c; Chen et al., 2025b). However, these settings largely examine these abilities separately, generally keeping service structures fixed or constraining tool discovery when services change. *They leave open whether agents can proactively discover efficient tool-use paths and promptly update their procedural knowledge as a large ecosystem evolves.* Addressing this question calls for an evaluation of sustained agent competence against the moving capability frontier of an evolving tool ecosystem.

To address this gap, we introduce $\Delta\tau$**-Bench**, an executable benchmark spanning the full tool-use lifecycle in large, evolving ecosystems. It comprises stateful services across 42 domains and 1,588 validated tasks requiring multi-turn user interaction. Starting from these services, we generate six successive ecosystem states through capability-preserving transformations that refine, split, or integrate tool interfaces. Older and newer tools may coexist or replace one another, changing how tasks can be completed while preserving access to the underlying capabilities. For each state, we identify the most efficient task paths within the constructed solution space and use them as references for evaluating agent behavior. By assessing tool discovery alongside stateful execution against these evolving references, $\Delta\tau$-Bench measures whether agents recover from disrupted workflows and exploit efficiency gains that successful execution alone may fail to reveal.

Evaluation shows that current agents struggle to sustain reliable and efficient execution in evolving tool ecosystems, with even the strongest evaluated model or harness achieving only 42.3% task success. Existing skill-learning methods provide initial gains, but these do not consistently translate into better adaptation as the ecosystem evolves, and skill maintenance can also incur substantial overhead. These findings highlight the need to manage accumulated knowledge together with the changing tools through which it is applied. We therefore introduce **STEWARD**, an ecosystem manager that coordinates the full lifecycle of tools and skills as a coupled resource system, from knowledge updates to downstream use. By tracking skill–tool dependencies, STEWARD incrementally maintains the skill library as tools change, preserving reusable experience. During execution, it jointly supplies relevant tools and procedural guidance, adapting this resource selection as the task unfolds. This connects what the agent learns and retains with what it can effectively use in the current ecosystem.

Experiments show that STEWARD sustains stronger performance across ecosystem changes while reducing skill-maintenance overhead. Its benefits extend beyond keeping existing workflows functional to exploiting more efficient solutions as they become available. Further analyses show that preserving and selectively updating accumulated knowledge is more effective than repeatedly rebuilding it, while jointly providing tools and skills yields benefits that neither resource provides alone.

2 TASK FORMULATION

We model agentic interaction as a partially observable sequential decision process $\mathcal{M} = (\mathcal{U}, \mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{P}, \mathcal{R})$. Here, $\mathcal{U}$ is the space of user intents; each task typically has an intent $u \in \mathcal{U}$ not directly observable to the agent. The state space $\mathcal{S} = \mathcal{S}_E \times \mathcal{S}_H$ combines external environment states s^E and agent–user interaction states s^H, with $s = (s^E, s^H)$. The action space $\mathcal{A} = \mathcal{A}_{\text{tool}} \cup \mathcal{A}_{\text{resp}}$ comprises executable tool calls $a = (f, \mathbf{x}) \in \mathcal{A}_{\text{tool}}$, with tool f and arguments $\mathbf{x}$, and natural-language messages $a = y \in \mathcal{A}_{\text{resp}}$ to the user. The observation space $\mathcal{O} = \mathcal{O}_{\text{tool}} \cup \mathcal{O}_{\text{user}}$ comprises tool returns or errors and the user's initial request and subsequent replies, respectively. With observable history $h_t = (o_0, a_0, \ldots, a_{t-1}, o_t)$, action selection and environment transitions follow:

$$a_t \sim \pi_\theta(\cdot \mid h_t), \qquad (s_{t+1}, o_{t+1}) \sim \mathcal{P}(\cdot \mid s_t, a_t, u). \tag{1}$$

Along $\tau = (o_0, a_0, o_1, a_1, \ldots, o_T, a_T)$, tool calls may modify s_t^E and dialogue may modify s_t^H. At completion, reward $\mathcal{R}(\tau, u)$ scores the final environment state and satisfaction of user constraints.

Each episode has an exogenous tool-ecosystem version $v \in \mathcal{V}$, fixed throughout execution, with action space $\mathcal{A}^{(v)} = \mathcal{A}_{\text{tool}}^{(v)} \cup \mathcal{A}_{\text{resp}}$, where $\mathcal{A}_{\text{tool}}^{(v)}$ contains all tool calls executable at v. Across versions, changes to available functions and admissible arguments can alter feasible functions and successful action paths for the same intent u. The policy must consider h_t and $\mathcal{A}^{(v)}$ at each step t.

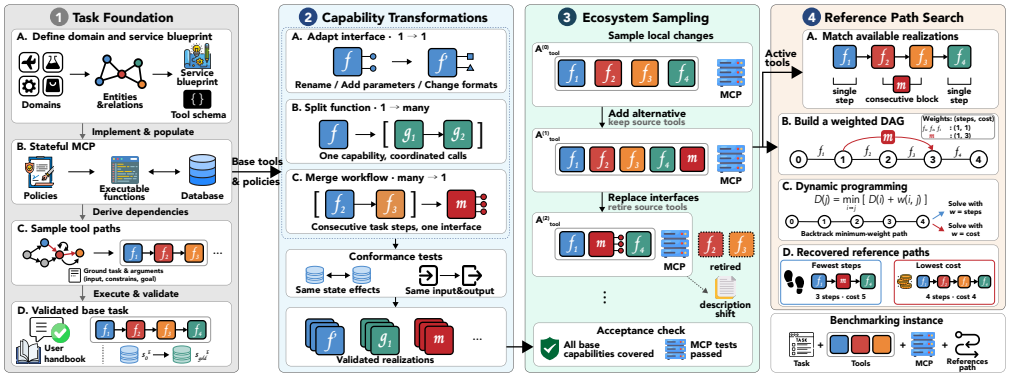


Figure 2: **Overview of $\Delta\tau$-Bench Construction.** We construct stateful environments and execution-validated tasks, then sample evolving tool ecosystems through validated capability-preserving transformations. Dynamic programming identifies step- and cost-minimal reference paths within the constructed realization space.

3 $\Delta\tau$-BENCH

We construct $\Delta\tau$-Bench (Fig. 2), an executable benchmark that systematically evaluates agent behavior in large and evolving tool ecosystems. It combines stateful multi-turn tasks with composable, capability-preserving transformations, enabling scalable ecosystem evolution without sacrificing executability and cross-ecosystem compatibility. $\Delta\tau$-Bench contains 1,588 tasks spanning 42 domains, with 1,191 tasks in the training set and 397 in the test set. Across six ecosystem states, this yields 9,528 task–ecosystem instances. Due to page limits, the detailed benchmark construction process and the statistics of $\Delta\tau$-Bench are described in Appx. A.

3.1 EXECUTABLE ENVIRONMENT AND TASK CONSTRUCTION

Domain-Grounded Stateful Environments. Following Sun et al. (2026), we expand seed domains into specialized, role-specific scenarios grouped by related activities. For each group, we construct entity schemas and an entity–relationship graph around a primary user role. These define coherent service blueprints specifying users, maintained information, and functions. Validated functions are deterministically converted into interfaces with natural-language descriptions and JSON Schemas, forming the tool-call action space $\mathcal{A}_{\text{tool}}$. We instantiate entity records and relationships in a mock database, validating schema and relational constraints to obtain the initial external state s_0^E. MCP services implement these interfaces under policies governing parameter semantics, invocation constraints, state changes, and dependencies, returning results or errors in declared formats.

Grounded Task Synthesis and Execution Validation. From the interfaces and policies, we derive a tool-dependency graph and sample paths under dependency and parameter-flow constraints. Sampling favors state-querying starts while allowing state-changing starts when their preconditions hold. To increase task difficulty, we join paths sharing a function and repeat intermediate subpaths, retaining only sequences with coherent function ordering, parameter flow, and task objectives. Conditioned on the service blueprints, interfaces, policies, and database summaries, we generate natural-language task templates and user intents requiring every sampled function for a single coherent objective. An independent evaluator screens templates for motivation clarity, coherence, function coverage, naturalness, and specificity. For retained templates, we ground arguments in user-known information or preceding tool outputs and replay instantiated sequences from s_0^E against the MCP service. Only instances where every action succeeds with the intended observations and state transitions are retained, yielding reference trajectories τ_{gold} and terminal external states s_{gold}^E.

Task-Conditioned User Simulation. An independent LLM-based user agent follows a general protocol and a private handbook specifying the complete intent u, motivation, grounded user information, and disclosure rules. We generate an initial request o_0 that expresses the user's immediate need while leaving some task details for subsequent dialogue. When the evaluated agent asks for missing task information, the user agent provides the corresponding details from the handbook in accordance with the task-specific disclosure rules. Throughout the dialogue, the user agent speaks from the user's perspective, progressively disclosing only grounded information without mentioning functions or execution plans. Responses are conditioned on the protocol, handbook, and visible dialogue, which contains only natural-language messages exchanged between the agents. Interaction ends when the user agent indicates that its intent is satisfied or the request is outside the service scope.

3.2 Evolving Tool Ecosystems Construction

Composable Capability Transformations. We construct each domain's space of alternative capability realizations from its executable base tool set $\mathcal{A}_{\text{tool}}^{(0)} = \mathcal{A}_{\text{tool}}$. A transformation $\phi : P \mapsto R$ maps a source block P to a target realization R, each containing one or more functions from the same domain, while preserving the source capabilities under a different interface structure. Function counts define three forms of transformation: 1) *one-to-one*, renaming functions or parameters, adding parameters, or changing formats; 2) *one-to-many*, decomposing a function into coordinated interfaces; and 3) *many-to-one*, integrating functions into a unified interface. Integration requires P to be a contiguous subsequence of a validated same-domain task trajectory, grounding R in a coherent, executable workflow. Given P, an LLM conditioned on the domain's service blueprint, function schemas, and operational policies proposes R and its executable implementation. Each candidate records explicit lineage to its source functions and is instantiated in the corresponding stateful environment for automated validation. Only transformations preserving required source capabilities, task-relevant state changes, and externally consumed outputs under equivalent inputs are retained. Local transformations incrementally expand this space without reconstructing domains or services.

Capability-Preserving Ecosystem Sampling. We organize validated realizations by source-to-target lineage and sample each version's active interface set $\mathcal{A}_{\text{tool}}^{(v)}$. Starting from all executable base interfaces in $\mathcal{A}_{\text{tool}}^{(0)}$, subsequent versions apply a bounded number of validated transformations to $\mathcal{A}_{\text{tool}}^{(v-1)}$, leaving other interfaces unchanged. For gradual updates and backward compatibility, we introduce R and either retain P, requiring selection among overlapping interfaces, or deactivate P, requiring agents to access and compose capabilities through R. We recursively verify coverage through lineage, rejecting transitions unless every base capability remains accessible through active interfaces, directly or through coordinated realizations. For accepted states, we adapt the domain's operational policies and assemble validated implementations into executable MCP services. We then optionally perturb selected function descriptions to simulate incomplete, compressed, or stale documentation, preserving available functions, parameter schemas, and execution semantics.

Task-Level Optimal Realization Search. Across versions, user intent and target state remain fixed, while task realizations may change. We extract each task's ordered base-function sequence $(f_1, \ldots, f_n)$ from its validated trajectory τ_{gold}. At version v, a task realization comprises active realizations whose source blocks exactly cover this sequence in order. The **reference path** minimizes the total value of a specified efficiency criterion over feasible task realizations within the constructed realization space. Step counts tool calls, while cost sums per-call scores (1–10) assigned by an LLM based on each tool's definition and implementation to reflect its potential execution cost. Let $\Phi^{(v)}(P)$ denote the realizations of P available in $\mathcal{A}_{\text{tool}}^{(v)}$. Partitioning the sequence into contiguous source blocks and selecting one active realization per block gives:$_m$

$$\rho_i^{(v,c)} = \underset{\substack{P_1 \oplus \cdots \oplus P_m = (f_1,\ldots,f_n) \\ R_j \in \Phi^{(v)}(P_j)}}{\arg\min} \sum_{j=1} c^{(v)}(R_j), \tag{2}$$

where $\oplus$ denotes sequence concatenation and $c^{(v)}(R_i)$ is the criterion value of R_i. Dynamic programming over active realizations in the task's domain separately minimizes function-call count and execution cost, yielding step-minimal and cost-minimal reference paths for each task and version. We replay each reference path in its corresponding MCP environment and retain only paths whose calls succeed and reproduce the required state changes.

3.3 Evaluation Protocol and Metrics

Outcome-based Task Success Rate. For task i at version v, the agent starts from $s_{0,i}^E$, and yields trajectory $\tau_i^{(v)}$ with terminal state $s_{T,i}^{E,(v)}$. Let $\Delta(s, s')$ denote database changes between external states. The required and agent-produced changes are:

$$\Delta_i^{\text{gold}} = \Delta(s_{0,i}^E, s_{\text{gold},i}^E), \qquad \Delta_i^{(v)} = \Delta(s_{0,i}^E, s_{T,i}^{E,(v)}). \tag{3}$$

A task succeeds when agent's changes cover all changes required by the validated gold execution:

$$S_i^{(v)} = \mathcal{R}(\tau_i^{(v)}, u_i) = \mathbb{I}\left[\Delta_i^{\text{gold}} \subseteq \Delta_i^{(v)}\right]. \tag{4}$$

Table 1: **Model evaluation on $\Delta\tau$-Bench.** The best results for each metric are highlighted in **bold**. Current models struggle to reach the capability frontier, with limited task success and inefficient execution.

Model	SR	Tool Retrieval			SPE		SPE/SR	
		P	R	F1	Step	Cost	Step	Cost
Proprietary Models								
Codex-GPT-5.5	**42.27**	19.90	**90.36**	**32.04**	**26.93**	**27.90**	63.71	66.00
Gemini-3.6-Flash	41.65	19.30	87.51	31.13	25.49	25.88	61.20	62.14
GPT-5.5	31.74	**20.21**	81.08	31.86	23.11	23.42	**72.81**	**73.79**
Claude-Sonnet-5	29.97	19.36	76.93	30.41	20.48	20.60	68.34	68.74
Open-source Models								
DeepSeek-V4-Pro	33.29	18.65	79.00	29.65	13.49	14.31	40.52	42.99
GLM-5.2	32.28	19.32	78.25	30.47	19.39	19.84	60.07	61.46
Qwen-3.8-Max	30.10	18.46	80.02	29.49	15.84	16.84	52.62	55.94
Kimi-K3	28.30	19.20	77.01	30.20	17.78	17.83	62.83	63.00
GPT-oss-120B	27.00	18.24	69.98	28.40	16.87	17.16	62.48	63.56

For N_v tasks evaluated at version v, the success rate $\mathrm{SR}^{(v)} = \frac{1}{N_v}\sum_{i=1}^{N_v} S_i^{(v)}$.

Tool Retrieval Quality. For task i at version v, let $\widehat{\mathcal{A}}_i^{(v)}$ and $\mathcal{A}_i^{(v)}$ denote the final tool set exposed to the agent after retrieval and the reference-path tools, respectively, both subsets of $\mathcal{A}_{\text{tool}}^{(v)}$. Task-level retrieval precision, recall, and F1 are:

$$\mathrm{P}_i^{(v)} = \frac{|\widehat{\mathcal{A}}_i^{(v)} \cap \mathcal{A}_i^{(v)}|}{|\widehat{\mathcal{A}}_i^{(v)}|}, \quad \mathrm{R}_i^{(v)} = \frac{|\widehat{\mathcal{A}}_i^{(v)} \cap \mathcal{A}_i^{(v)}|}{|\mathcal{A}_i^{(v)}|}, \quad \mathrm{F1}_i^{(v)} = \frac{2\mathrm{P}_i^{(v)}\mathrm{R}_i^{(v)}}{\mathrm{P}_i^{(v)}+\mathrm{R}_i^{(v)}}. \tag{5}$$

Success-weighted Path Efficiency. Success-weighted Path Efficiency (SPE) measures execution efficiency relative to reference paths. For task i at version v, let $\mathcal{W}_i^{(v)}$ and $\widehat{\mathcal{W}}_i^{(v)}$ denote the reference-path and agent tool-call trajectory weights under the same criterion. We define

$$\mathrm{SPE}^{(v)} = \frac{1}{N_v}\sum_{i=1}^{N_v} S_i^{(v)} \min\left(\frac{\mathcal{W}_i^{(v)}}{\widehat{\mathcal{W}}_i^{(v)}}, 1\right). \tag{6}$$

Using function-call count and execution cost yields Step-SPE and Cost-SPE, respectively.

Improvement Discovery and Degradation Adaptation. To evaluate frontier adaptation, we identify tasks whose reference-path weights decrease or increase across adjacent ecosystem versions:

$$\mathcal{D}_{\downarrow}^{(v)} = \left\{ i \mid \mathcal{W}_i^{(v)} < \mathcal{W}_i^{(v-1)} \right\}, \qquad \mathcal{D}_{\uparrow}^{(v)} = \left\{ i \mid \mathcal{W}_i^{(v)} > \mathcal{W}_i^{(v-1)} \right\}. \tag{7}$$

The Improvement Discovery Rate (IDR) and Degradation Adaptation Rate (DAR) are:

$$\mathrm{IDR}^{(v)} = \frac{1}{\left|\mathcal{D}_{\downarrow}^{(v)}\right|}\sum_{i\in\mathcal{D}_{\downarrow}^{(v)}} S_i^{(v)} \min\left(\frac{\mathcal{W}_i^{(v)}}{\widehat{\mathcal{W}}_i^{(v)}}, 1\right), \mathrm{DAR}^{(v)} = \frac{1}{\left|\mathcal{D}_{\uparrow}^{(v)}\right|}\sum_{i\in\mathcal{D}_{\uparrow}^{(v)}} S_i^{(v)} \min\left(\frac{\mathcal{W}_i^{(v)}}{\widehat{\mathcal{W}}_i^{(v)}}, 1\right). \tag{8}$$

IDR measures the discovery of newly shortened or cheaper paths. DAR measures effectiveness when optimal paths become longer or costlier. Steps and cost yield Step-IDR/DAR and Cost-IDR/DAR.

4 Evaluating Agents in Evolving Tool Ecosystems

Evaluation Setup. We evaluate whether current powerful models can reach the capability frontier of each of six tool-ecosystem versions and whether automated skill methods can track this frontier by accumulating experiences as the ecosystem evolves. For models that expose a configurable reasoning effort, we set it to `high` and retain the recommended defaults for all other inference parameters. The user model is fixed to DeepSeek-V3.2 throughout all experiments. All controlled model ablations and all comparisons in the skill-method study use DeepSeek-V3.2 as the task model.

4.1 Models Struggle to Reach the Current Frontier

We examine whether current models can independently reach the capability frontier of each tool-ecosystem version. As shown in Tab. 1, even the strongest model (Gemini-3.6-Flash) and harness

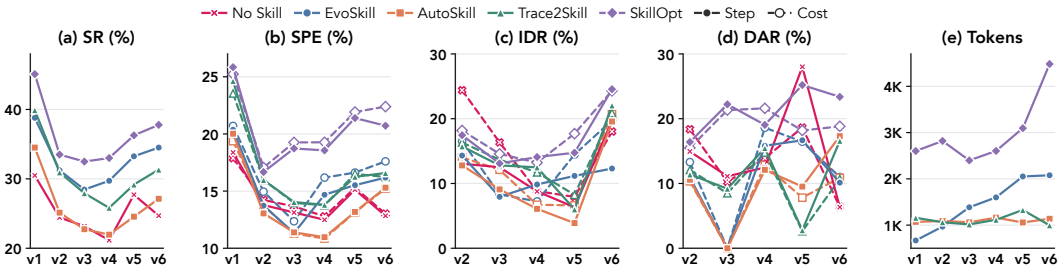


Figure 4: **Evaluation of automated skill methods across ecosystem versions.** Current skill evolution strategy can provide initial performance gains but cannot consistently adapt to the moving capability frontier.

(Codex-GPT-5.5) achieve SRs of only 41.65% and 42.27%, respectively, while neither exceeds 30% in Step- or Cost-SPE. Tool retrieval exhibits a pronounced imbalance: Codex-GPT-5.5 retrieves 90.36% of the relevant tools, yet attains only 19.90% precision and an F1 score of 32.04%, a pattern that persists across models, indicating that models often include relevant tools but struggle to exclude irrelevant candidates from the broader ecosystem. Moreover, success does not necessarily imply efficient execution. For example, although DeepSeek-V4-Pro slightly outperforms GLM-5.2 in SR, it trails in both Step- and Cost-SPE, and Step- and Cost-SPE/SR.

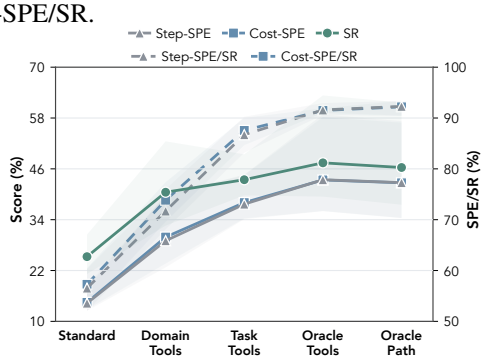


Figure 3: **Ablations on Tool Provision**.

We next conduct a controlled diagnostic to probe barriers to identifying and executing efficient tool-use paths. Specifically, we compare *Standard* with four increasingly informative conditions. *Domain Tools* restricts retrieval to the task domain, whereas *Task Tools* provides all tools used by the task across versions to test whether models can distinguish currently available and efficient options from unavailable or overlapping alternatives. *Oracle Tools* retains only relevant, current-version tools, while *Oracle Path* additionally provides the reference tool path to isolate path-planning difficulty. As shown in Fig. 3, SR and both SPE variants rise steadily from *Standard* through *Domain Tools* and *Task Tools* to *Oracle Tools*, whereas *Oracle Path* yields no further improvement. The gains from *Standard* to *Domain Tools* and *Task Tools* show that removing domain- and task-irrelevant candidates eases tool identification. More importantly, the pronounced SPE/SR gain from *Task Tools* to *Oracle Tools* shows that models struggle to distinguish efficient, version-valid options from unavailable or overlapping alternatives. The negligible change under *Oracle Path* instead indicates that the remaining gap lies less in high-level planning than in correctly and efficiently using available tools. Overall, current models struggle both to retrieve relevant tools precisely and to execute their efficient realizations in evolving ecosystems.

4.2 Skill Methods Struggle to Track the Moving Frontier

Our preceding diagnosis shows that models struggle primarily to identify and efficiently execute currently valid tool-use paths in an evolving ecosystem. So, we examine whether automated skill generation and optimization methods can overcome this limitation by accumulating experience. As shown in Fig. 4, we compare four automated skill methods: EvoSkill (Alzubi et al., 2026), AutoSkill (Yang et al., 2026b), Trace2Skill (Ni et al., 2026), SkillOpt (Yang et al., 2026a), against the *No Skill* baseline across six ecosystem versions (Please refer to Appx. B for baseline details). At *v1*, all four methods outperform *No Skill* in SR and both SPE variants, showing that accumulated experience provides useful initial guidance. However, these gains do not persist uniformly: AutoSkill falls below *No Skill* in SR at *v3* and *v5*, while AutoSkill and EvoSkill also underperform it in SPE at several intermediate versions. This limitation is more pronounced in DAR: at *v3*, both EvoSkill and AutoSkill drop to 0 in Step- and Cost-DAR, whereas *No Skill* retains 11.11% and 10.25%, respectively. Moreover, *No Skill* achieves the highest Cost-IDR at *v2* and both DAR variants at *v5*, indicating that accumulated skills do not consistently improve adaptation to ecosystem changes. Accordingly, although SkillOpt leads in SR and SPE across versions, its advantage does not extend consistently to IDR and DAR, showing that stronger current-version performance does not necessarily imply better frontier tracking. This stronger current-version performance also incurs substantially higher maintenance costs, with SkillOpt averaging about 3,000 K/task for skill generation and optimization,

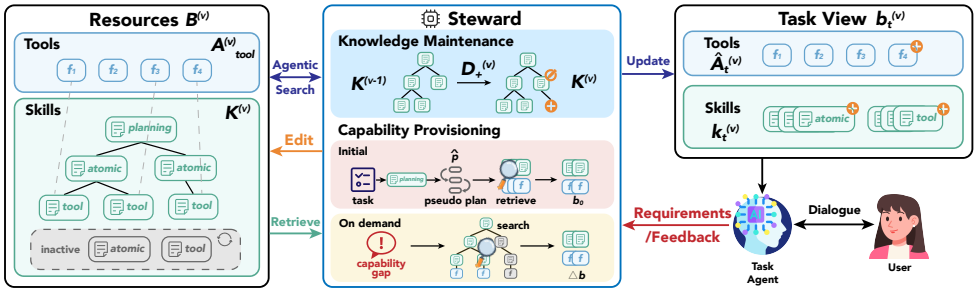


Figure 5: **Overview of STEWARD.** As an ecosystem manager, STEWARD maintains shared tool and skill resources, updating hierarchical skills using verified experience and current tool availability. It initializes the task agent's view through planning and retrieval, then dynamically updates it as capability gaps emerge.

roughly two to three times the other methods. These findings show that generating and optimizing skills in isolation is insufficient for sustained and efficient frontier tracking. Reliable adaptation requires tools and skills to be jointly maintained and provisioned as a unified, evolving ecosystem.

5 STEWARD: MANAGING TOOLS AND SKILLS IN EVOLVING ECOSYSTEMS

The Ecosystem Manager. We introduce STEWARD, a model-driven ecosystem manager that coordinates the maintenance of tool and skill resources and their provision to agents for task execution. At version v, STEWARD maintains a shared resource state $\mathcal{B}^{(v)} = (\mathcal{A}^{(v)}_{\text{tool}}, \mathcal{K}^{(v)})$, where $\mathcal{K}^{(v)}$ is a persistent skill library that records procedural knowledge, tool dependencies, and availability status. Through dedicated management interfaces, STEWARD updates skills, retrieves relevant tools and knowledge, and inspects tool schemas to configure the resources available to the task agent. Conditioned on the interaction history h_t, STEWARD constructs a task-specific view $b_t^{(v)} = (\widehat{\mathcal{A}}_t^{(v)}, k_t^{(v)})$ of these resources, comprising the exposed tools $\widehat{\mathcal{A}}_t^{(v)} \subseteq \mathcal{A}^{(v)}_{\text{tool}}$ and the accompanying procedural context $k_t^{(v)}$. The task agent then acts according to $a_t \sim \pi_\theta(\cdot \mid h_t, b_t^{(v)})$, while STEWARD maintains $\mathcal{K}^{(v)}$ across versions and adapts $b_t^{(v)}$ to evolving task requirements. Due to page limits, please refer to Appx. C for more method details and Appx. D for prompts used in STEWARD.

Knowledge Maintenance. We initialize skills at $v = 1$ by following (Wang et al., 2026). As tools evolve, STEWARD incrementally updates $\mathcal{K}^{(v-1)}$ using the current tool catalog and newly verified execution experience while preserving reusable knowledge. The library follows a three-level hierarchy: *planning skills* capture task-level workflows, *atomic skills* encode the reusable operations within these workflows, and *tool skills* specify the invocation requirements of individual tools and are linked to the atomic skills that use them. At version v, the manager takes as input $\mathcal{K}^{(v-1)}$, $\mathcal{A}^{(v)}_{\text{tool}}$, and a set $\mathcal{D}_+^{(v)}$ of successful training trajectories derived from the task agent and verified through environment-state evaluation. Conditioned on $\mathcal{K}^{(v-1)}$, the manager distills $\mathcal{D}_+^{(v)}$ into local edits to the existing skill hierarchy, adding or revising skills at the appropriate levels and updating their links. The manager then marks skills as active or inactive according to the availability of their tool dependencies in $\mathcal{A}^{(v)}_{\text{tool}}$, retaining inactive skills for reactivation when their dependencies become available again. The resulting library $\mathcal{K}^{(v)}$ preserves both partitions, with retrieval indices and the skill graph rebuilt over its active subset $\mathcal{K}^{(v)}_{\text{act}}$ for capability provisioning.

Capability Provisioning. For each task, STEWARD jointly selects tools from $\mathcal{A}^{(v)}_{\text{tool}}$ and procedural knowledge from $\mathcal{K}^{(v)}_{\text{act}}$ to construct $b_t^{(v)}$ according to the task's current requirements. At the start of an episode, the manager retrieves planning skills relevant to the initial user request and uses them to construct a task-specific provisional plan $\hat{p}$. For each step of $\hat{p}$, the manager retrieves atomic skills and combines their referenced tools with directly retrieved tools. Selected atomic skills and their linked tool skills form $k_0^{(v)}$, while candidate tools available in the current version form $\widehat{\mathcal{A}}_0^{(v)}$, together constituting $b_0^{(v)}$. When new user information or execution feedback reveals capability gaps, the manager supplements the task view through lightweight retrieval or agentic search and inspection across $\mathcal{A}^{(v)}_{\text{tool}}$ and $\mathcal{K}^{(v)}_{\text{act}}$. The manager adds verified tools to $\widehat{\mathcal{A}}_t^{(v)}$ and updates $k_t^{(v)}$ with procedural guidance and, when needed, the full text of selected skills. By coupling incremental knowledge maintenance with on-demand capability provisioning, STEWARD aims to reduce the costs of skill updates and task execution while preserving task success as the tool ecosystem evolves.

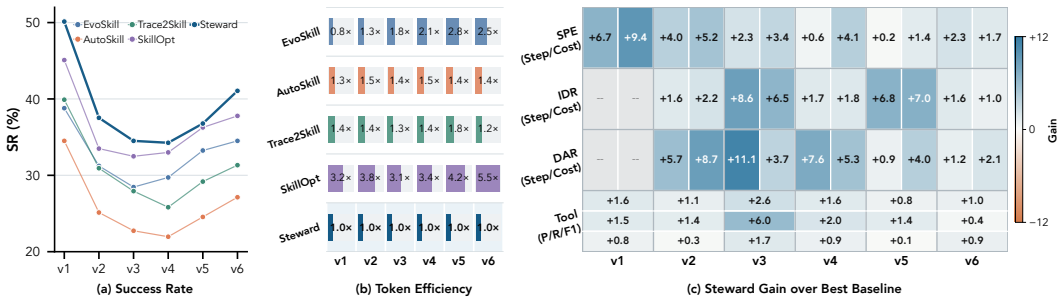


Figure 6: **Evaluation of STEWARD in evolving tool ecosystems. (a) Task success.** STEWARD achieves the highest SR across all six ecosystem versions. **(b) Relative maintenance cost.** Baselines require 1.2–5.5× as many maintenance tokens as STEWARD from v_2 onward. **(c) Percentage-point gains over the best baseline per metric and version.** STEWARD consistently improves execution efficiency, frontier tracking, and tool retrieval.

6 EXPERIMENTAL RESULTS FOR STEWARD

STEWARD Tracks the moving frontier efficiently. We evaluate whether the joint management of tools and skills enables STEWARD to sustain task success and efficiently track the moving capability frontier using DeepSeek-V3.2. As shown in Fig. 6, we compare STEWARD with four automated skill-based methods across six tool-ecosystem versions under the same task model, user model, and evaluation protocol. In Fig. 6(a), STEWARD achieves the highest SR at every version, averaging 39.04% versus 36.36% for SkillOpt, showing that its advantage persists beyond initial skill construction as the ecosystem evolves. Fig. 6(b) further shows that STEWARD incurs the lowest maintenance cost from *v2* onward, with the baselines consuming 1.2–5.5× as many skill-maintenance tokens. Thus, the higher task success is achieved without increasing skill-update expenditure, distinguishing STEWARD from methods that rely on increasingly costly skill generation or optimization. To examine whether this advantage extends beyond SR, Fig. 6(c) reports the percentage-point gain of STEWARD over the strongest baseline for each version and metric. Most notably, STEWARD improves Step-/Cost-IDR by 4.04%/3.69% and Step-/Cost-DAR by 5.28%/4.76% on average, showing that its advantage extends from current-version success to discovering improved paths and recovering from degraded ones across ecosystem changes. These dynamic gains are accompanied by average Step-/Cost-SPE improvements of 2.68%/4.21% and consistent gains in tool precision, recall, and F1, indicating more efficient execution and more effective capability provisioning. Together, these results establish that STEWARD combines stronger frontier tracking with lower maintenance overhead, motivating the following analyses of which management mechanisms produce these gains.

Table 2: **Ablations on cross-version knowledge maintenance.** Across both models, incremental maintenance leads on all performance metrics while requiring fewer maintenance tokens than full reconstruction.

Model	Strategy	Avg. Results (%)					Tokens (K/task)
		SR	SPE	IDR	DAR	Tool F1	
DeepSeek-V3.2	Frozen Library	33.90	20.34/21.18	17.96/19.56	24.29/22.16	35.96	**0.00**
	Full Reconstruction	34.61	21.06/21.95	17.62/18.80	24.24/21.37	33.56	1,271.10
	Incremental Maintenance	**36.83**	**21.10/23.15**	**20.83/21.29**	**26.53/23.84**	**39.37**	769.55
GPT-5.5	Frozen Library	39.60	28.23/29.26	25.65/28.11	28.69/26.17	27.20	**0.00**
	Full Reconstruction	38.99	27.75/28.69	27.58/28.96	26.85/24.48	26.50	1,449.60
	Incremental Maintenance	**40.81**	**30.67/31.62**	**29.11/31.27**	**31.23/29.62**	**32.54**	1,009.02

Incremental maintenance balances knowledge reuse and adaptation. To isolate cross-version maintenance, strategies within each model start from the same skill library at *v1* and differ only in their maintenance from *v2* onward. As shown in Table 2, incremental maintenance achieves the highest average SR for both DeepSeek-V3.2 and GPT-5.5 (36.83% and 40.81%), together with the best SPE, IDR, DAR, and tool F1. Compared with full reconstruction, it improves SR by 2.22 and 1.82 percentage points while reducing maintenance tokens by 39.5% and 30.4%, respectively. Freezing avoids update costs but yields lower SR than incremental maintenance; full reconstruction even underperforms freezing on GPT-5.5 (38.99% vs. 39.60%) despite its update cost. These results suggest that targeted updates better balance the reuse of accumulated knowledge with adaptation to ecosystem changes.

Dynamic tool and skill search are complementary. We ablate capability provisioning on DeepSeek-V3.2 by taking initial task view construction as the baseline and adding dynamic skill search, tool search, or both. As shown in Fig. 7, skill search alone yields little improvement in SR, SPE, IDR, or Tool F1, indicating that procedural knowledge is difficult to operationalize when the corresponding tools are not dynamically exposed. Tool search alone improves SR and Tool F1 by 8.65% and 4.14% percentage points, respectively, but provides limited SPE and IDR gains and

reduces Step-DAR by 2.76%, showing that finding available tools does not ensure their efficient or adaptive use. Combining both modules improves every metric; relative to tool search alone, *Full Provisioning* raises Step-/Cost-DAR by 11.41%/9.30% percentage points and Tool F1 by 4.90% points. These results show that tool search exposes valid implementations while skill search supplies the procedural guidance needed to use them, making their joint management essential for capability provisioning.

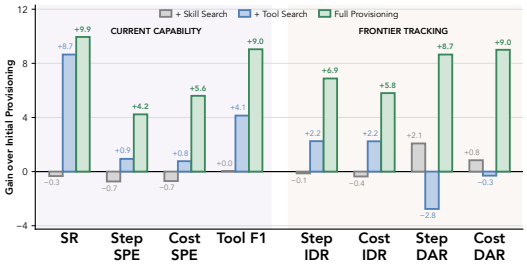


Figure 7: **Ablations on capability provisioning.** Combining dynamic tool and skill search yields the largest gains over initial provisioning across all metrics, demonstrating their complementary roles in task execution.

7 Related Work

Benchmarks for Tool-Using Agents. As LLMs become autonomous agents, tool ecosystems mediate discovery and use of external capabilities, motivating benchmarks spanning a broader tool-use lifecycle (Qu et al., 2025; Qin et al., 2025b). Early benchmarks assess API selection, argument generation, multi-tool composition, and planning (Patil et al., 2024; Qin et al., 2024; Li et al., 2023; Tang et al., 2023; Patil et al., 2025; Chen et al., 2024b; 2025a). Stateful successors evaluate multi-turn interaction in environments with persistent tool effects, where success requires satisfying user constraints (Trivedi et al., 2024; Yao et al., 2024; Barrès et al., 2025; Lu et al., 2025; Li et al., 2025). Complementary benchmarks study retrieval from large, heterogeneous tool repositories (Xu et al., 2024; Shi et al., 2025; Qin et al., 2025a; Wang et al., 2025a; Liu et al., 2026d), while efficiency-oriented benchmarks evaluate resource constraints, planning or operational costs, and replanning under changing costs, preferences, or tool availability (Valmeekam et al., 2023; Hazra et al., 2024; Kokel et al., 2025; Xie et al., 2024; Qian et al., 2025b; Liu et al., 2026e). Recent work probes robustness through API-level perturbations or controlled mutations of MCP servers (Chen et al., 2025b; Liu et al., 2026c). Yet these capabilities remain largely evaluated in isolation, leaving the end-to-end tool-use lifecycle in evolving ecosystems underexplored. $\Delta\tau$-Bench addresses this gap as the first benchmark unifying large-scale tool retrieval, multi-turn stateful execution, and efficiency-aware adaptation within a controlled, evolving tool ecosystem.

Skill Learning and Evolution for LLM Agents. Agents should convert interaction experience into reusable procedural knowledge for continual improvement. Early work distills trajectories into natural-language guidance or reusable workflows, while later methods represent skills as executable or compositional routines (Zhao et al., 2024; Fu et al., 2024; Chen et al., 2024a; Wang et al., 2025b; Qiao et al., 2024a; Wang et al., 2024; Liu et al., 2026a; Zheng et al., 2025; Chen et al., 2026). Recent approaches automate skill construction and refinement through trajectory consolidation, optimization, reinforcement learning, verification, and persistent accumulation (Yang et al., 2026b; Ni et al., 2026; Alzubi et al., 2026; Yang et al., 2026a; Zheng et al., 2026; He & Yang, 2026; Tang et al., 2026), with complementary work organizing growing libraries through hierarchies, dependencies, graphs, and lifecycle management (Wang et al., 2026; Liu et al., 2026b; Pu et al., 2026; Bai et al., 2026; Liang et al., 2026). Yet these methods largely evolve skills over a stable operational substrate and emphasize eventual task success, leaving open whether procedural knowledge can adapt quickly enough to sustain efficient execution as the capability frontier moves. STEWARD recasts skill evolution as ecosystem-level adaptation, coupling tools with procedural knowledge to preserve reusable experience and track the evolving efficiency frontier.

8 Conclusion

We introduce $\Delta\tau$-Bench, an executable benchmark evaluating sustained agent competence against the moving capability frontier of large, evolving tool ecosystems. The evaluated agents struggle to sustain reliable and efficient execution, while initial gains from accumulated skills do not consistently translate into better adaptation as tools change. STEWARD couples incremental skill maintenance with dynamic provision of tools and procedural guidance, improving task performance and adaptation while reducing maintenance overhead. These results support managing tools and procedural knowledge as interdependent resources, keeping accumulated experience useful as the ecosystem evolves.

AI USE STATEMENT

We affirm that Large Language Models are employed solely as an assistive tool to refine wording and sentence structure and to design the display format of some figures during our paper writing process. Their use in the experiments is strictly for scientific research purposes, and all such usage has been explicitly documented in our Experimental Settings and Reproducibility Statement. We have reviewed all AI-assisted work. We take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of LLMs.

ETHICS STATEMENT

This study was conducted in full compliance with established ethical standards and research best practices. All data employed are derived and synthesized exclusively from publicly available sources; no proprietary or confidential information was used. Every reference to these data sources is accurately and appropriately cited throughout the paper. We strongly encourage all users of our benchmark to uphold the highest ethical standards, ensuring fairness, transparency, and responsibility in their research. Any use that could cause harm or negatively impact society is strictly prohibited.

REPRODUCIBILITY STATEMENT

The benchmark and the code of this paper are included in the Supplementary Material. The construction of executable environments and tasks is described in §3.1 and Appx. A.1, while ecosystem evolution and reference-path search are detailed in §3.2 and Appx. A.2. Dataset statistics are reported in Appx. A.4, and the evaluation protocol and metrics are defined in §3.3. The design of STEWARD, including incremental knowledge maintenance and dynamic provisioning of tools and skills, is presented in §5. The settings for baseline comparisons and controlled ablations are reported in §4, §6, and Appx. B. The prompts used in our study are provided in Appx. D.

REFERENCES

Salaheddin Alzubi, Noah Provenzano, Jaydon Bingham, Weiyuan Chen, and Tu Vu. Evoskill: Automated skill discovery for multi-agent systems. *CoRR*, abs/2603.02766, 2026. doi: 10.48550/ARXIV.2603.02766. URL https://doi.org/10.48550/arXiv.2603.02766.

Tong Bai, Zhenglin Wan, Pengfei Zhou, Xingrui Yu, Wangbo Zhao, Yang You, and Ivor W. Tsang. Skilldag: Self-evolving typed skill graphs for LLM skill selection at scale. *CoRR*, abs/2606.03056, 2026. doi: 10.48550/ARXIV.2606.03056. URL https://doi.org/10.48550/arXiv.2606.03056.

Victor Barrès, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. τ^2-bench: Evaluating conversational agents in a dual-control environment. *CoRR*, abs/2506.07982, 2025. doi: 10.48550/ARXIV.2506.07982. URL https://doi.org/10.48550/arXiv.2506.07982.

Chen Chen, Xinlong Hao, Weiwen Liu, Xu Huang, Xingshan Zeng, Shuai Yu, Dexun Li, Yuefeng Huang, Xiangcheng Liu, Xinzhi Wang, and Wu Liu. Acebench: A comprehensive evaluation of LLM tool usage. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2025, Suzhou, China, November 4-9, 2025*, pp. 12970–12998. Association for Computational Linguistics, 2025a. doi: 10.18653/V1/2025.FINDINGS-EMNLP.697. URL https://doi.org/10.18653/v1/2025.findings-emnlp.697.

Guoxin Chen, Zhong Zhang, Xin Cong, Fangda Guo, Yesai Wu, Yankai Lin, Wenzheng Feng, and Yasheng Wang. Learning evolving tools for large language models. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025b. URL https://openreview.net/forum?id=wtrDLMFU9v.

Minghao Chen, Yihang Li, Yanting Yang, Shiyu Yu, Binbin Lin, and Xiaofei He. Automanual: Constructing instruction manuals by LLM agents via interactive environmental learning. In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela

Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (eds.), *Advances in Neural Information Processing Systems 37: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024a. URL http://papers.nips.cc/paper_files/paper/2024/hash/0142921fad7ef9192bd87229cdafa9d4-Abstract-Conference.html.

Tianyi Chen, Yinheng Li, Michael Solodko, Sen Wang, Nan Jiang, Tingyuan Cui, Junheng Hao, Jongwoo Ko, Sara Abdali, Suzhen Zheng, Leon Xu, Hao Fan, Pashmina Cameron, Justin Wagle, and Kazuhito Koishida. Cua-skill: Develop skills for computer using agent. *CoRR*, abs/2601.21123, 2026. doi: 10.48550/ARXIV.2601.21123. URL https://doi.org/10.48550/arXiv.2601.21123.

Zehui Chen, Weihua Du, Wenwei Zhang, Kuikun Liu, Jiangning Liu, Miao Zheng, Jingming Zhuo, Songyang Zhang, Dahua Lin, Kai Chen, and Feng Zhao. T-eval: Evaluating the tool utilization capability of large language models step by step. 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), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pp. 9510–9529. Association for Computational Linguistics, 2024b. doi: 10.18653/V1/2024.ACL-LONG.515. URL https://doi.org/10.18653/v1/2024.acl-long.515.

Guanting Dong, Junting Lu, Junjie Huang, Wanjun Zhong, Longxiang Liu, Shijue Huang, Zhenyu Li, Yang Zhao, Xiaoshuai Song, Xiaoxi Li, et al. Agent-world: Scaling real-world environment synthesis for evolving general agent intelligence. *arXiv preprint arXiv:2604.18292*, 2026a.

Guanting Dong, Xiaoshuai Song, Yuyang Hu, Jiajie Jin, Chenghao Zhang, Yifei Chen, Xiaoxi Li, Huaying Yuan, Xinyu Yang, Tongyu Wen, et al. Towards long-horizon agents: A survey. 2026b.

Runnan Fang, Shihao Cai, Baixuan Li, Jialong Wu, Guangyu Li, Wenbiao Yin, Xinyu Wang, Xiaobin Wang, Liangcai Su, Zhen Zhang, Shibin Wu, Zhengwei Tao, Yong Jiang, Pengjun Xie, Ningyu Zhang, Fei Huang, Wentao Zhang, and Jingren Zhou. Towards general agentic intelligence via environment scaling. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Findings of the Association for Computational Linguistics, ACL 2026, San Diego, California, United States, July 2-7, 2026*, pp. 17610–17621. Association for Computational Linguistics, 2026. doi: 10.18653/V1/2026.FINDINGS-ACL.872. URL https://doi.org/10.18653/v1/2026.findings-acl.872.

Yao Fu, Dong-Ki Kim, Jaekyeom Kim, Sungryull Sohn, Lajanugen Logeswaran, Kyunghoon Bae, and Honglak Lee. Autoguide: Automated generation and selection of context-aware guidelines for large language model agents. In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (eds.), *Advances in Neural Information Processing Systems 37: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024. URL http://papers.nips.cc/paper_files/paper/2024/hash/d8efbb5dd415974eb095c3f06bff1f48-Abstract-Conference.html.

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. *CoRR*, abs/2402.01680, 2024. doi: 10.48550/ARXIV.2402.01680. URL https://doi.org/10.48550/arXiv.2402.01680.

Rishi Hazra, Pedro Zuidberg Dos Martires, and Luc De Raedt. Saycanpay: Heuristic planning with large language models using learnable domain knowledge. In Michael J. Wooldridge, Jennifer G. Dy, and Sriraam Natarajan (eds.), *Thirty-Eighth AAAI Conference on Artificial Intelligence, AAAI 2024, Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence, IAAI 2024, Fourteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2024, February 20-27, 2024, Vancouver, Canada*, pp. 20123–20133. AAAI Press, 2024. doi: 10.1609/AAAI.V38I18.29991. URL https://doi.org/10.1609/aaai.v38i18.29991.

Yu He and Weikai Yang. Skillcommit: Evolving agent skills through behaviorally validated scope expansion. *CoRR*, abs/2608.15165, 2026. doi: 10.48550/ARXIV.2608.15165. URL https://doi.org/10.48550/arXiv.2608.15165.

Harsha Kokel, Michael Katz, Kavitha Srinivas, and Shirin Sohrabi. Acpbench: Reasoning about action, change, and planning. In Toby Walsh, Julie Shah, and Zico Kolter (eds.), *Thirty-Ninth AAAI Conference on Artificial Intelligence, Thirty-Seventh Conference on Innovative Applications of Artificial Intelligence, Fifteenth Symposium on Educational Advances in Artificial Intelligence, AAAI 2025, Philadelphia, PA, USA, February 25 - March 4, 2025*, pp. 26559–26568. AAAI Press, 2025. doi: 10.1609/AAAI.V39I25.34857. URL https://doi.org/10.1609/aaai.v39i25.34857.

Junlong Li, Wenshuo Zhao, Jian Zhao, Weihao Zeng, Haoze Wu, Xiaochen Wang, Rui Ge, Yuxuan Cao, Yuzhen Huang, Wei Liu, Junteng Liu, Zhaochen Su, Yiyang Guo, Fan Zhou, Lueyang Zhang, Juan Michelini, Xingyao Wang, Xiang Yue, Shuyan Zhou, Graham Neubig, and Junxian He. The tool decathlon: Benchmarking language agents for diverse, realistic, and long-horizon task execution. *CoRR*, abs/2510.25726, 2025. doi: 10.48550/ARXIV.2510.25726. URL https://doi.org/10.48550/arXiv.2510.25726.

Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. Api-bank: A comprehensive benchmark for tool-augmented llms. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, pp. 3102–3116. Association for Computational Linguistics, 2023. doi: 10.18653/V1/2023.EMNLP-MAIN.187. URL https://doi.org/10.18653/v1/2023.emnlp-main.187.

Yuan Liang, Ruobin Zhong, Haoming Xu, Chen Jiang, Yi Zhong, Runnan Fang, Jia-Chen Gu, Shumin Deng, Yunzhi Yao, Mengru Wang, Shuofei Qiao, Xin Xu, Tongtong Wu, Kun Wang, Yang Liu, Zhen Bi, Jungang Lou, Yuchen Eleanor Jiang, Hangcheng Zhu, Gang Yu, Haiwen Hong, Longtao Huang, Hui Xue, Chenxi Wang, Yijun Wang, Zifei Shan, Xi Chen, Zhaopeng Tu, Feiyu Xiong, Xin Xie, Peng Zhang, Zhengke Gui, Lei Liang, Jun Zhou, Chiyu Wu, Jin Shang, Yu Gong, Junyu Lin, Changliang Xu, Hongjie Deng, Wen Zhang, Keyan Ding, Qiang Zhang, Fei Huang, Ningyu Zhang, Jeff Z. Pan, Guilin Qi, Haofen Wang, and Huajun Chen. Skillnet: Create, evaluate, and connect AI skills. *CoRR*, abs/2603.04448, 2026. doi: 10.48550/ARXIV.2603.04448. URL https://doi.org/10.48550/arXiv.2603.04448.

Guangyi Liu, Pengxiang Zhao, Liang Liu, Zhiming Chen, Yuxiang Chai, Yaozhen Liang, Wenhao Wang, Siheng Chen, Zhengxi Lu, Shuai Ren, Hao Wang, Shibo He, Yong Liu, and Wenchao Meng. Learnact: Few-shot mobile GUI agent with a unified demonstration benchmark. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Findings of the Association for Computational Linguistics, ACL 2026, San Diego, California, United States, July 2-7, 2026*, pp. 29820–29843. Association for Computational Linguistics, 2026a. doi: 10.18653/V1/2026.FINDINGS-ACL.1491. URL https://doi.org/10.18653/v1/2026.findings-acl.1491.

Hongyi Liu, Haoyan Yang, Tao Jiang, Bo Tang, Feiyu Xiong, and Zhiyu Li. Skillsvote: Lifecycle governance of agent skills from collection, recommendation to evolution. *CoRR*, abs/2605.18401, 2026b. doi: 10.48550/ARXIV.2605.18401. URL https://doi.org/10.48550/arXiv.2605.18401.

Huanxi Liu, Kun Hu, Jiaqi Liao, Qiang Wang, Pengfei Qian, Yuanzhao Zhai, Dawei Feng, Bo Ding, and Huaimin Wang. Mcpevol-bench: Benchmarking LLM agent performance across dynamic evolutions of MCP servers. *CoRR*, abs/2607.14642, 2026c. doi: 10.48550/ARXIV.2607.14642. URL https://doi.org/10.48550/arXiv.2607.14642.

Jiayu Liu, Qihan Lin, Cheng Qian, Rui Wang, Emre Can Acikgoz, Xiaocheng Yang, Jiateng Liu, Zhenhailong Wang, Xiusi Chen, Heng Ji, and Dilek Hakkani-Tür. Planbench-xl: Evaluating long-horizon planning of LLM tool-use agents in large-scale tool ecosystems. *CoRR*, abs/2606.22388, 2026d. doi: 10.48550/ARXIV.2606.22388. URL https://doi.org/10.48550/arXiv.2606.22388.

Jiayu Liu, Cheng Qian, Zhaochen Su, Qing Zong, Shijue Huang, Bingxiang He, and Yi R. Fung. Costbench: Evaluating multi-turn cost-optimal planning and adaptation in dynamic environments for LLM tool-use agents. 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), ACL 2026, San Diego, California, United States, July 2-7, 2026*, pp. 12826–12858. Association for Computational Linguistics, 2026e. doi: 10.18653/V1/2026.ACL-LONG.584. URL https://doi.org/10.18653/v1/2026.acl-long.584.

Jiarui Lu, Thomas Holleis, Yizhe Zhang, Bernhard Aumayer, Feng Nan, Haoping Bai, Shuang Ma, Shen Ma, Mengyu Li, Guoli Yin, Zirui Wang, and Ruoming Pang. Toolsandbox: A stateful, conversational, interactive evaluation benchmark for LLM tool use capabilities. In Luis Chiruzzo, Alan Ritter, and Lu Wang (eds.), *Findings of the Association for Computational Linguistics: NAACL 2025, Albuquerque, New Mexico, USA, April 29 - May 4, 2025*, volume NAACL 2025 of *Findings of ACL*, pp. 1160–1183. Association for Computational Linguistics, 2025. doi: 10.18653/V1/2025.FINDINGS-NAACL.65. URL https://doi.org/10.18653/v1/2025.findings-naacl.65.

Jingwei Ni, Yihao Liu, Xinpeng Liu, Yutao Sun, Mengyu Zhou, Pengyu Cheng, Dexin Wang, Erchao Zhao, Xiaoxi Jiang, and Guanjun Jiang. Trace2skill: Distill trajectory-local lessons into transferable agent skills. *CoRR*, abs/2603.25158, 2026. doi: 10.48550/ARXIV.2603.25158. URL https://doi.org/10.48550/arXiv.2603.25158.

Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. Gorilla: Large language model connected with massive apis. In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (eds.), *Advances in Neural Information Processing Systems 37: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024. URL http://papers.nips.cc/paper_files/paper/2024/hash/e4c61f578ff07830f5c37378dd3ecb0d-Abstract-Conference.html.

Shishir G. Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E. Gonzalez. The berkeley function calling leaderboard (BFCL): from tool use to agentic evaluation of large language models. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Forty-second International Conference on Machine Learning, ICML 2025, Vancouver, BC, Canada, July 13-19, 2025*, volume 267 of *Proceedings of Machine Learning Research*. PMLR / OpenReview.net, 2025. URL https://proceedings.mlr.press/v267/patil25a.html.

Hongji Pu, Xinyuan Song, and Liang Zhao. Skillops: Managing LLM agent skill libraries as self-maintaining software ecosystems. *CoRR*, abs/2605.13716, 2026. doi: 10.48550/ARXIV.2605.13716. URL https://doi.org/10.48550/arXiv.2605.13716.

Cheng Qian, Emre Can Acikgoz, Qi He, Hongru Wang, Xiusi Chen, Dilek Hakkani-Tur, Gokhan Tur, and Heng Ji. Toolrl: Reward is all tool learning needs. In Danielle Belgrave, Cheng Zhang, Laura N. Montoya, Hsuan-Tien Lin, Razvan Pascanu, Piotr Koniusz, Marzyeh Ghassemi, Nancy Chen, Iván Vladimir Meza Ruíz, and Arturo Loaiza-Bonilla (eds.), *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2025, NeurIPS 2025, San Diego, CA, USA, December 2-7, 2025 / Mexico City, Mexico, November 30 - December 5, 2025*, 2025a. URL http://papers.nips.cc/paper_files/paper/2025/hash/97c5b2707228e7e3fb67e4ecc2e0e607-Abstract-Conference.html.

Cheng Qian, Zuxin Liu, Akshara Prabhakar, Zhiwei Liu, Jianguo Zhang, Haolin Chen, Heng Ji, Weiran Yao, Shelby Heinecke, Silvio Savarese, Caiming Xiong, and Huan Wang. Userbench: An interactive gym environment for user-centric agents. *CoRR*, abs/2507.22034, 2025b. doi: 10.48550/ARXIV.2507.22034. URL https://doi.org/10.48550/arXiv.2507.22034.

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 Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (eds.), *Advances in Neural Information Processing Systems 37: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024a. URL http://papers.nips.cc/paper_files/paper/2024/hash/d032263772946dd5026e7f3cd22bce5b-Abstract-Conference.html.

Shuofei Qiao, Ningyu Zhang, Runnan Fang, Yujie Luo, Wangchunshu Zhou, Yuchen Eleanor Jiang, Chengfei Lv, and Huajun Chen. Autoact: Automatic agent learning from scratch for QA via self-planning. 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), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pp. 3003–3021. Association for Computational Linguistics, 2024b. doi: 10.18653/V1/2024.ACL-LONG.165. URL https://doi.org/10.18653/v1/2024.acl-long.165.

Shengqian Qin, Yakun Zhu, Linjie Mu, Shaoting Zhang, and Xiaofan Zhang. Meta-tool: Unleash open-world function calling capabilities of general-purpose large language models. 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), ACL 2025, Vienna, Austria, July 27 - August 1, 2025*, pp. 30653–30677. Association for Computational Linguistics, 2025a. doi: 10.18653/V1/2025.ACL-LONG.1481. URL https://doi.org/10.18653/v1/2025.acl-long.1481.

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 *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=dHng2O0Jjr.

Yujia Qin, Shengding Hu, Yankai Lin, Weize Chen, Ning Ding, Ganqu Cui, Zheni Zeng, Xuanhe Zhou, Yufei Huang, Chaojun Xiao, Chi Han, Yi R. Fung, Yusheng Su, Huadong Wang, Cheng Qian, Runchu Tian, Kunlun Zhu, Shihao Liang, Xingyu Shen, Bokai Xu, Zhen Zhang, Yining Ye, Bowen Li, Ziwei Tang, Jing Yi, Yuzhang Zhu, Zhenning Dai, Lan Yan, Xin Cong, Yaxi Lu, Weilin Zhao, Yuxiang Huang, Junxi Yan, Xu Han, Xian Sun, Dahai Li, Jason Phang, Cheng Yang, Tongshuang Wu, Heng Ji, Guoliang Li, Zhiyuan Liu, and Maosong Sun. Tool learning with foundation models. *ACM Comput. Surv.*, 57(4):101:1–101:40, 2025b. doi: 10.1145/3704435. URL https://doi.org/10.1145/3704435.

Changle Qu, Sunhao Dai, Xiaochi Wei, Hengyi Cai, Shuaiqiang Wang, Dawei Yin, Jun Xu, and Ji-Rong Wen. Tool learning with large language models: a survey. *Frontiers Comput. Sci.*, 19 (8):198343, 2025. doi: 10.1007/S11704-024-40678-2. URL https://doi.org/10.1007/s11704-024-40678-2.

Zhengliang Shi, Yuhan Wang, Lingyong Yan, Pengjie Ren, Shuaiqiang Wang, Dawei Yin, and Zhaochun Ren. Retrieval models aren't tool-savvy: Benchmarking tool retrieval for large language models. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Findings of the Association for Computational Linguistics, ACL 2025, Vienna, Austria, July 27 - August 1, 2025*, volume ACL 2025 of *Findings of ACL*, pp. 24497–24524. Association for Computational Linguistics, 2025. doi: 10.18653/V1/2025.FINDINGS-ACL.1258. URL https://doi.org/10.18653/v1/2025.findings-acl.1258.

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. *CoRR*, abs/2511.21689, 2025. doi: 10.48550/ARXIV.2511.21689. URL https://doi.org/10.48550/arXiv.2511.21689.

Zexu Sun, Bokai Ji, Hengyi Cai, Shuaiqiang Wang, Lei Wang, Guangxia Li, and Xu Chen. Agentskiller: Scaling generalist agent intelligence through semantically integrated cross-domain data synthesis. *CoRR*, abs/2602.09372, 2026. doi: 10.48550/ARXIV.2602.09372. URL https://doi.org/10.48550/arXiv.2602.09372.

Liyan Tang, Cyrus Rashtchian, Chun-Sung Ferng, Andrew Tomkins, Da-Cheng Juan, and Tu Vu. Wikiskill: Compiling agent experience into persistent knowledge for skill evolution. *CoRR*, abs/2608.27454, 2026. doi: 10.48550/ARXIV.2608.27454. URL https://doi.org/10.48550/arXiv.2608.27454.

Qiaoyu Tang, Ziliang Deng, Hongyu Lin, Xianpei Han, Qiao Liang, and Le Sun. Toolalpaca: Generalized tool learning for language models with 3000 simulated cases. *CoRR*, abs/2306.05301, 2023. doi: 10.48550/ARXIV.2306.05301. URL https://doi.org/10.48550/arXiv.2306.05301.

Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. Appworld: A controllable world of apps and people for benchmarking interactive coding agents. 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), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pp. 16022–16076. Association for Computational Linguistics, 2024. doi: 10.18653/V1/2024.ACL-LONG.850. URL https://doi.org/10.18653/v1/2024.acl-long.850.

Karthik Valmeekam, Matthew Marquez, Alberto Olmo Hernandez, Sarath Sreedharan, and Subbarao Kambhampati. Planbench: An extensible benchmark for evaluating large language models on planning and reasoning about change. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (eds.), *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023. URL http://papers.nips.cc/paper_files/paper/2023/hash/7a92bcdede88c7afd108072faf5485c8-Abstract-Datasets_and_Benchmarks.html.

Chenxi Wang, Zhuoyun Yu, Xin Xie, Wuguannan Yao, Runnan Fang, Shuofei Qiao, Kexin Cao, Guozhou Zheng, Xiang Qi, Peng Zhang, and Shumin Deng. Skillx: Automatically constructing skill knowledge bases for agents. *CoRR*, abs/2604.04804, 2026. doi: 10.48550/ARXIV.2604.04804. URL https://doi.org/10.48550/arXiv.2604.04804.

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. *Trans. Mach. Learn. Res.*, 2024, 2024. 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 Ji-Rong Wen. A survey on large language model based autonomous agents. *CoRR*, abs/2308.11432, 2023. doi: 10.48550/ARXIV.2308.11432. URL https://doi.org/10.48550/arXiv.2308.11432.

Renxi Wang, Xudong Han, Lei Ji, Shu Wang, Timothy Baldwin, and Haonan Li. Toolgen: Unified tool retrieval and calling via generation. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025a. URL https://openreview.net/forum?id=XLMAMmowdY.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Forty-second International Conference on Machine Learning, ICML 2025, Vancouver, BC, Canada, July 13-19, 2025*, volume 267 of *Proceedings of Machine Learning Research*. PMLR / OpenReview.net, 2025b. URL https://proceedings.mlr.press/v267/wang25bx.html.

Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, Rui Zheng, Xiaoran Fan, Xiao Wang, Limao Xiong, Yuhao Zhou, Weiran Wang, Changhao Jiang, Yicheng Zou, Xiangyang Liu, Zhangyue Yin, Shihan Dou, Rongxiang Weng, Wenjuan Qin, Yongyan Zheng, Xipeng Qiu, Xuanjing Huang, Qi Zhang, and Tao Gui. The rise and potential of large language model based agents: a survey. *Sci. China Inf. Sci.*, 68(2), 2025. doi: 10.1007/S11432-024-4222-0. URL https://doi.org/10.1007/s11432-024-4222-0.

Jian Xie, Kai Zhang, Jiangjie Chen, Tinghui Zhu, Renze Lou, Yuandong Tian, Yanghua Xiao, and Yu Su. Travelplanner: A benchmark for real-world planning with language agents. In Ruslan Salakhutdinov, Zico Kolter, Katherine A. Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Forty-first International Conference on Machine Learning, ICML 2024,*

Vienna, Austria, July 21-27, 2024, volume 235 of *Proceedings of Machine Learning Research*, pp. 54590–54613. PMLR / OpenReview.net, 2024. URL https://proceedings.mlr.press/v235/xie24j.html.

Qiancheng Xu, Yongqi Li, Heming Xia, and Wenjie Li. Enhancing tool retrieval with iterative feedback from large language models. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2024, Miami, Florida, USA, November 12-16, 2024*, volume EMNLP 2024 of *Findings of ACL*, pp. 9609–9619. Association for Computational Linguistics, 2024. doi: 10.18653/V1/2024.FINDINGS-EMNLP.561. URL https://doi.org/10.18653/v1/2024.findings-emnlp.561.

Yifan Yang, Ziyang Gong, Weiquan Huang, Qihao Yang, Ziwei Zhou, Zisu Huang, Yan Li, Xuemei Gao, Qi Dai, Bei Liu, Kai Qiu, Yuqing Yang, Dongdong Chen, Xue Yang, and Chong Luo. Skillopt: Executive strategy for self-evolving agent skills. *CoRR*, abs/2605.23904, 2026a. doi: 10.48550/ARXIV.2605.23904. URL https://doi.org/10.48550/arXiv.2605.23904.

Yutao Yang, Junsong Li, Qianjun Pan, Bihao Zhan, Yuxuan Cai, Lin Du, Jie Zhou, Kai Chen, Qin Chen, Xin Li, Bo Zhang, and Liang He. Autoskill: Experience-driven lifelong learning via skill self-evolution. *CoRR*, abs/2603.01145, 2026b. doi: 10.48550/ARXIV.2603.01145. URL https://doi.org/10.48550/arXiv.2603.01145.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. *CoRR*, abs/2406.12045, 2024. doi: 10.48550/ARXIV.2406.12045. URL https://doi.org/10.48550/arXiv.2406.12045.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. Expel: LLM agents are experiential learners. In Michael J. Wooldridge, Jennifer G. Dy, and Sriraam Natarajan (eds.), *Thirty-Eighth AAAI Conference on Artificial Intelligence, AAAI 2024, Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence, IAAI 2024, Fourteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2024, February 20-27, 2024, Vancouver, Canada*, pp. 19632–19642. AAAI Press, 2024. doi: 10.1609/AAAI.V38I17.29936. URL https://doi.org/10.1609/aaai.v38i17.29936.

Boyuan Zheng, Michael Y. Fatemi, Xiaolong Jin, Zora Zhiruo Wang, Apurva Gandhi, Yueqi Song, Yu Gu, Jayanth Srinivasa, Gaowen Liu, Graham Neubig, and Yu Su. Skillweaver: Web agents can self-improve by discovering and honing skills. *CoRR*, abs/2504.07079, 2025. doi: 10.48550/ARXIV.2504.07079. URL https://doi.org/10.48550/arXiv.2504.07079.

Mingxuan Zheng, Yujin Zhou, Chuxue Cao, Boqin Yin, Yuyao Zhang, Jiapeng Sun, Shuaishuai Gong, Sirui Han, and Yike Guo. Skillprox: Self-evolving agent skills via proximal textual gradient descent. *CoRR*, abs/2608.07449, 2026. doi: 10.48550/ARXIV.2608.07449. URL https://doi.org/10.48550/arXiv.2608.07449.

A Details of $\Delta\tau$-Bench Construction

A.1 Executable Environment and Task Construction

Domain-Grounded Service Modeling. Following Sun et al. (2026), we use application domains as semantic anchors for subsequent service and environment generation. Each domain specifies a coherent scope of user goals and activities, thereby preventing the generated components from becoming an arbitrary collection of unrelated concepts and operations. Starting from a set of seed domains, we expand the domain pool into more specialized and role-specific scenarios and group related domains according to the activities they cover. For example, the *artist-booking* domain may cover activities such as identifying suitable artists, checking their availability, and arranging bookings for events. Given the resulting domain group, we identify a primary user role and represent the information involved in the corresponding activities as entity schemas, each of which defines a type of object and its attributes. We then construct an entity–relationship graph whose nodes represent entity types and whose edges encode the semantic relationships between them. Based on coherent subsets of entities and relationships in this graph, we derive service blueprints that define the intended users, maintained information, and available functions of each service. Each

function is further described by its functional purpose, required and optional parameters, and returned information. In the *artist-booking* example, a *booking coordinator* serves as the primary user, while *artist*, *client*, *event*, and *booking* constitute the main entity types. The entity–relationship graph may associate each *booking* with a *client*, an *artist*, and an *event*, providing the semantic basis for functions that *retrieve candidate artists*, *check their availability*, and *create bookings*. After validation, each function is deterministically converted into a function-calling interface containing a function name and natural-language description, together with a JSON Schema object that specifies the names, types, descriptions, required fields, and enumeration constraints of its input parameters. The resulting interfaces form the tool-call action space $\mathcal{A}_{\text{tool}}$ for each service.

From Service Blueprints to Stateful Environments. The service blueprints and function interfaces specify what each service should support, but they do not yet provide an environment in which the functions can be executed and their effects can persist. We first instantiate each entity schema with concrete records and materialize the relationships between these records in a mock database. The resulting records and relationships collectively define the initial external state s_0^E, after being checked against the corresponding schema and relational constraints. In the *artist-booking* example, the stateful database s_0^E may contain concrete *artists*, *clients*, *events*, and *bookings*, with each *booking* record referring to a specific *client*, *artist*, and *event*. A database alone, however, does not determine when a function may be invoked or how it changes the environment s_t^E. We then define operational policies that specify each function's preconditions, parameter semantics, execution constraints, state changes, and cross-function dependencies. Using the function interfaces as callable contracts and the operational policies as behavioral specifications, we implement the function as executable server-side code in the MCP service. Concretely, each implementation is a Python function registered under the name and input schema declared by the corresponding interface. When invoked, it receives the supplied arguments, validates them against the schema, resolves the referenced records, enforces the applicable operational constraints, and performs the required read or write operations on the mock database. It then converts the execution result or encountered error into the return format declared by the interface. At runtime, the MCP server routes a tool action $a_t = (f_t, \mathbf{x}_t) \in \mathcal{A}_{\text{tool}}$ to the server-side implementation associated with f_t. The function processes $\mathbf{x}_t$ according to the operational policy, updates s_t^E when necessary, and returns an observation $o_{t+1} \in \mathcal{O}_{\text{tool}}$.

Grounded Task Synthesis and Execution Validation. Given a validated stateful environment, we next construct multi-step tasks whose required operations and information dependencies are grounded in the environment. We first derive a tool-dependency graph from the function interfaces and operational policies. Each node represents a function, while a directed edge indicates an execution dependency between two functions. A valid path sampled on this graph naturally provides an executable structure that can be transformed into a multiple coordinated tool-call task. To obtain diverse candidate structures, we sample paths from the graph by first drawing a starting node from a weighted probability distribution over eligible functions. Because many user tasks follow a read-before-write pattern, first locating or inspecting target entities and then modifying them, we assign higher starting probabilities to query-type functions that retrieve information without changing s_t^E. This preference is probabilistic rather than a hard constraint, allowing state-changing functions to remain possible starting points when their preconditions are already satisfied.

From the starting node, we extend the path on the graph, preserving dependency and parameter-flow constraints. To improve complexity, we further construct tasks requiring longer or repeated tool use through two graph-grounded augmentation operations. First, we join two paths when they share at least one function node. The shared function provides an explicit semantic connection between the two sequences and serves as an anchor for organizing them under a unified task objective. Second, we select an intermediate subpath and repeat it one or more times to model tasks that require iterating the same group of operations. We retain the resulting sequence only if its function ordering, parameter flow, and overall task objective remain coherent.

Because graph connectivity alone does not guarantee that a function sequence corresponds to a real-world user request, we ground task synthesis in the domain semantics. Specifically, we generate a natural-language task template and a unified user intent conditioned on the service blueprint, function interfaces, operational policies, and database summary. The template must describe a single coherent user objective whose completion requires every function in the sampled sequence. An independent

evaluator then assesses each template in terms of motivation clarity, logical coherence, function coverage, naturalness, and specificity. Templates that do not meet the quality threshold are discarded.

For each retained template, we determine the provenance of every required argument. Arguments corresponding to facts or constraints known by the user are assigned to the user side of the task. Identifiers and values produced during execution are linked to the preceding tool outputs. We then instantiate the template with concrete records and parameter values consistent with the initial external state s_0^E. Starting from s_0^E, we replay the grounded function sequence against the MCP service under $\mathcal{P}$, supplying user-known arguments from the instantiated task and propagating intermediate tool outputs to subsequent calls when required. We retain only instances for which every action executes successfully and produces the intended observations and state transitions. The validated execution is recorded as the reference tool trajectory τ_{gold}, and its terminal external state is stored as s_{gold}^E.

Task-Conditioned User Simulation. To instantiate the user side of multi-turn interaction, we introduce an independent LLM-based user agent that collaborates with the evaluated agent throughout task execution. The user agent is governed by a general interaction protocol and a private task handbook. The interaction protocol requires the user agent to preserve the assigned intent, disclose information progressively, use only grounded values, and communicate from the user's perspective without referring to functions or execution plans. For each task, the private task handbook specifies the complete user intent u, the underlying motivation, the grounded information assigned to the user side of the task, and task-specific instructions governing when that information may be disclosed. We additionally generate a concise initial request o_0 for the evaluated agent that expresses the user's immediate need while leaving information intended for later clarification undisclosed. During rollout, when the evaluated agent produces a natural-language response $y_t \in \mathcal{A}_{\text{resp}}$, the user agent generates the next observation $o_{t+1} \in \mathcal{O}_{\text{user}}$ conditioned on the interaction protocol, the private task handbook, and the visible dialogue. When clarification is requested, the user agent provides the corresponding value from the task handbook. Its visible dialogue contains only the natural-language messages exchanged between the two agents. The generated response advances the interaction state s_t^H. The interaction continues until the user agent indicates that its intent has been satisfied or identifies the request as outside the service scope.

A.2 Evolving Tool Ecosystems Construction

Composable Capability Transformations. For each domain, we start from its executable base tool set $\mathcal{A}_{\text{tool}}^{(0)} = \mathcal{A}_{\text{tool}}$ and construct a domain-specific space of alternative capability realizations. We represent each transformation as $\phi : P \mapsto R$, where both the source block P and the target realization R can contain one or more functions. Both P and R are restricted to the same domain, and R must preserve the capabilities represented by P under a different interface structure. The numbers of functions contained in P and R induce three general forms of transformation: 1) A one-to-one transformation refines access to an existing capability by renaming the function or parameters, adding parameters, or changing their formats; 2) A one-to-many transformation decomposes one function into several coordinated interfaces; 3) A many-to-one transformation integrates multiple functions into a unified interface. For integration, the source block is further restricted to a contiguous subsequence observed in a validated task trajectory from the same domain. This restriction grounds the merged interface in a coherent and executable workflow rather than an arbitrary combination of functions. Given a source block P, we use an LLM conditioned on the domain's service blueprint, function schemas, and operational policies to propose the target realization R and its executable implementation. Each candidate maintains explicit lineage links to the source functions it covers and is instantiated in the corresponding stateful environment for automated validation. We retain a transformation only when its target realization R preserves the required source capabilities P, task-relevant state changes, and externally consumed outputs under equivalent inputs. Because transformations modify only local capability blocks, the evolution space can be expanded incrementally without reconstructing domains or services.

Capability-Preserving Ecosystem Sampling. The validated transformations define the alternative capability realizations available within each domain, but they do not determine which interfaces are exposed at each ecosystem version. We organize these realizations through their source-to-target lineage and denote the active interface set at version v by $\mathcal{A}_{\text{tool}}^{(v)}$. The initial state $\mathcal{A}_{\text{tool}}^{(0)}$ contains all

executable base interfaces, from which subsequent states are generated incrementally. To construct $\mathcal{A}_{\text{tool}}^{(v)}$, we start from $\mathcal{A}_{\text{tool}}^{(v-1)}$ and apply a bounded number of validated transformations while leaving the remaining interfaces unchanged. To emulate gradual interface update and backward compatibility, we sometimes introduce R while retaining P, creating overlapping realizations that require agents to select among redundant interfaces. In other cases, we deactivate the interfaces in P after introducing R, requiring agents to access and compose the underlying capabilities through the new realization. We accept a sampled state only when every capability in the base tool set remains accessible through its active interfaces, either directly or through a coordinated realization. Using the lineage relations established above, we recursively verify this coverage and reject any transition that leaves a source capability without a valid realization. For each accepted state, we adapt the domain's operational policies and assemble the validated interface implementations into an executable MCP service. To further simulate incomplete, compressed, or stale documentation in real-world tool ecosystem evolution, we optionally perturb selected function descriptions after constructing the executable service. This produces controlled documentation shifts while keeping the available functions, parameter schemas, and execution semantics unchanged.

Task-Level Optimal Realization Search. Although the user intent and target state remain fixed, different ecosystem versions may provide different realizations for completing the same task. For each task, we extract the ordered base-function sequence $(f_1, \ldots, f_n)$ from its validated trajectory τ_{gold}. At version v, a task realization is an ordered sequence of active realizations whose associated source blocks exactly cover this base-function sequence. Under a specified efficiency criterion, we define the **reference path** as the feasible task realization with the minimum total criterion value within the constructed realization space. Step counts tool calls, while cost sums per-call scores (1–10) assigned by an LLM based on each tool's definition and implementation to reflect its potential execution cost. Let $\Phi^{(v)}(P)$ denote the realizations of source block P available in $\mathcal{A}_{\text{tool}}^{(v)}$. We search for the reference path by partitioning the base-function sequence into contiguous source blocks and selecting one active realization for each block:

$$\rho_i^{(v,c)} = \underset{\substack{P_1 \oplus \cdots \oplus P_m = (f_1, \ldots, f_n) \\ R_j \in \Phi^{(v)}(P_j)}}{\arg\min} \sum_{j=1}^{m} c^{(v)}(R_j), \tag{9}$$

where $\oplus$ denotes sequence concatenation and $c^{(v)}(R_i)$ specifies the efficiency criterion for realization R_i. We solve this sequence-covering problem through dynamic programming over the active realizations in the task's domain. In this paper, we use function-call count and execution cost as separate criteria to produce a step-minimal reference path and a cost-minimal reference path for each task and version. These paths are stored as benchmark annotations for the evaluation protocol and are not prescribed as trajectories that the evaluated agent must reproduce.

A.3 Evaluation Protocol and Metrics

Outcome-based Task Success. We first evaluate task completion from environment outcomes. For each task i and version v, the corresponding MCP service is initialized with the task's external state $s_{0,i}^{E}$, and the agent trajectory $\tau_i^{(v)}$ produces a terminal state $s_{T,i}^{E,(v)}$. Let $\Delta(s, s')$ denote the set of database changes between two external states, including inserted, updated, and deleted records. We obtain the required and agent-produced changes as:

$$\Delta_i^{\text{gold}} = \Delta(s_{0,i}^{E}, s_{\text{gold},i}^{E}), \qquad \Delta_i^{(v)} = \Delta(s_{0,i}^{E}, s_{T,i}^{E,(v)}). \tag{10}$$

A task is considered successful when the agent-produced changes cover all changes required by the validated gold execution:

$$S_i^{(v)} = \mathcal{R}(\tau_i^{(v)}, u_i) = \mathbb{I}\left[\Delta_i^{\text{gold}} \subseteq \Delta_i^{(v)}\right]. \tag{11}$$

For the N_v tasks evaluated at version v, we report the success rate:

$$\text{SR}^{(v)} = \frac{1}{N_v} \sum_{i=1}^{N_v} S_i^{(v)}. \tag{12}$$

Table 3: **Statistics of evolving tool ecosystems.** Tool scale across six ecosystem states and the corresponding interface changes.

Ecosystem	Tool Scale		Interface Changes		
	# Tools	Avg.	Added	Retired	Modified
v_1	892	21.24	–	–	–
v_2	841	20.02	110	161	176
v_3	837	19.93	196	200	147
v_4	799	19.02	125	163	150
v_5	858	20.43	179	120	155
v_6	890	21.19	166	134	165

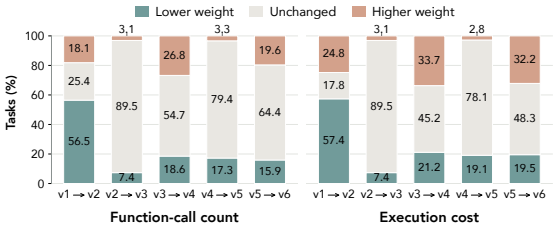


Figure 8: **Reference-path shifts across transitions.** Task distributions by function-call count and execution cost.

Tool Retrieval Quality. Beyond task completion, we evaluate whether the agent can accurately identify tools required by the current task from the complete action space. For task i at version v, let $\widehat{\mathcal{A}}_i^{(v)} \subseteq \mathcal{A}_{\text{tool}}^{(v)}$ denote the final tool set exposed to the agent after retrieval. We use $\mathcal{A}_i^{(v)} \subseteq \mathcal{A}_{\text{tool}}^{(v)}$ to denote the tools contained in the reference path. The task-level retrieval P, R, and F1 are defined as:

$$\mathrm{P}_i^{(v)} = \frac{\left|\widehat{\mathcal{A}}_i^{(v)} \cap \mathcal{A}_i^{(v)}\right|}{\left|\widehat{\mathcal{A}}_i^{(v)}\right|}, \qquad \mathrm{R}_i^{(v)} = \frac{\left|\widehat{\mathcal{A}}_i^{(v)} \cap \mathcal{A}_i^{(v)}\right|}{\left|\mathcal{A}_i^{(v)}\right|}, \qquad \mathrm{F1}_i^{(v)} = \frac{2\mathrm{P}_i^{(v)}\mathrm{R}_i^{(v)}}{\mathrm{P}_i^{(v)} + \mathrm{R}_i^{(v)}}. \tag{13}$$

Success-weighted Path Efficiency. We define Success-weighted Path Efficiency (SPE) to measure how efficiently an agent completes tasks relative to the reference paths. For task i at version v, let $\mathcal{W}_i^{(v)}$ denote the reference-path weight and $\widehat{\mathcal{W}}_i^{(v)}$ the weight of the agent's tool-call trajectory under the same criterion. SPE is defined as

$$\mathrm{SPE}^{(v)} = \frac{1}{N_v} \sum_{i=1}^{N_v} S_i^{(v)} \min\left(\frac{\mathcal{W}_i^{(v)}}{\widehat{\mathcal{W}}_i^{(v)}}, 1\right). \tag{14}$$

Using function-call count and execution cost as the path weight yields Step-SPE and Cost-SPE.

Improvement Discovery and Degradation Adaptation. To evaluate whether the agent can adapt to the evolving frontier, we compare the reference-path weight of each task across adjacent ecosystem versions. We define the tasks whose optimal execution decreases or increases as:

$$\mathcal{D}_{\downarrow}^{(v)} = \left\{ i \mid \mathcal{W}_i^{(v)} < \mathcal{W}_i^{(v-1)} \right\}, \qquad \mathcal{D}_{\uparrow}^{(v)} = \left\{ i \mid \mathcal{W}_i^{(v)} > \mathcal{W}_i^{(v-1)} \right\}. \tag{15}$$

The Improvement Discovery Rate (IDR) and Degradation Adaptation Rate (DAR) are defined as:

$$\mathrm{IDR}^{(v)} = \frac{1}{\left|\mathcal{D}_{\downarrow}^{(v)}\right|} \sum_{i \in \mathcal{D}_{\downarrow}^{(v)}} S_i^{(v)} \min\left(\frac{\mathcal{W}_i^{(v)}}{\widehat{\mathcal{W}}_i^{(v)}}, 1\right), \mathrm{DAR}^{(v)} = \frac{1}{\left|\mathcal{D}_{\uparrow}^{(v)}\right|} \sum_{i \in \mathcal{D}_{\uparrow}^{(v)}} S_i^{(v)} \min\left(\frac{\mathcal{W}_i^{(v)}}{\widehat{\mathcal{W}}_i^{(v)}}, 1\right). \tag{16}$$

IDR measures whether the agent discovers newly shortened or cheaper execution paths, whereas DAR measures whether it remains effective when the optimal path becomes longer or more costly. Using function-call count or execution cost as the path weight yields Step-IDR/DAR and Cost-IDR/DAR.

A.4 Benchmark Statistics and Characteristics

Overall, $\Delta\tau$-Bench contains 1,588 tasks spanning 42 domains, with 1,191 tasks in the training set and 397 in the test set; across six ecosystem states, this yields 9,528 task–ecosystem instances. As shown in Tab. 3, the aggregate tool scale remains comparable across states, whereas Fig. 8 reveals non-monotonic changes in reference-path weights across transitions. The different distributions under function-call count and execution cost further indicate that interaction length and execution expense capture complementary dimensions of efficiency. Tab. 4 further details the task counts and mean tool-call lengths of the original gold (reference) trajectories for each domain and split.

Table 4: **Domain-level task statistics.** Mean calls are computed from the original gold trajectories, before ecosystem-specific interface transformations.

Domain	Tasks			Mean gold calls	
	Train	Test	Total	Train	Test
AgriculturalEngineeringTrainingServer	34	11	45	4.32	4.36
AgriculturalSkillsTrainingHub	32	10	42	4.69	5.00
ArtistBookingAgency	13	4	17	4.54	4.75
ArtistCareerTracker	62	20	82	5.45	5.55
ArtistContractComplianceAdvisor	53	18	71	5.60	5.50
ArtistTeachingStudio	42	14	56	4.67	4.79
ClassroomSafetyAnalyzer	26	9	35	4.92	4.89
ClinicalEfficacyAnalyzer	70	24	94	4.93	5.04
ClinicalProcedureManager	10	4	14	4.30	4.50
CommunityFacilitySchedulingCoordinator	5	2	7	4.80	5.00
ConstructionProjectRiskServer	43	14	57	5.16	5.50
CourseComplianceManager	11	4	15	5.18	4.50
CurriculumResourceIntegrator	58	19	77	5.14	4.89
EngineeringSpecificationComplianceServer	50	16	66	5.74	5.56
FarmEquipmentServiceNetwork	37	12	49	5.43	5.58
FirstResponderCommunityResourceCoordination	17	6	23	4.88	5.33
FirstResponderFieldOpsCommunicationAndMonitoring	14	5	19	4.43	4.40
FirstResponderIncidentReportingAndSceneMgmt	7	2	9	6.43	6.50
FirstResponderTrainingAndEquipmentCertification	2	0	2	4.00	–
FleetServiceCoordinator	33	11	44	5.18	5.27
GrantAndResourceComplianceTracker	1	0	1	4.00	–
InfrastructureComplianceManager	62	20	82	5.03	5.25
InfrastructureMonitoringHub	46	16	62	5.35	5.25
LivestockHealthMonitor	20	7	27	4.80	5.00
MedicalDeviceEngineeringServer	84	28	112	5.65	5.79
MobileServiceDeploymentOptimizer	13	4	17	4.38	4.00
MobilityPlatformIntegrator	34	13	47	5.12	5.08
PharmaceuticalTreatmentOptimizer	12	4	16	5.42	5.25
PolicyImpactReviewer	28	10	38	4.71	4.70
PolicyImplementationAndRiskAdvisor	16	5	21	4.56	4.40
PrecisionCropManagementAdvisor	56	19	75	5.14	5.05
PublicAssistanceWorkflowManager	3	1	4	4.33	4.00
PublicEmployeeSupportPortal	14	5	19	5.07	5.00
PublicInfrastructurePlanner	78	26	104	4.87	4.85
RiskMonitorCoordinator	49	16	65	5.16	4.94
ServiceProviderCompensationAndSupportManager	18	6	24	5.00	5.33
ServiceProviderManager	4	1	5	6.00	5.00
ServiceQualityAndInsightGenerator	19	6	25	4.16	4.00
SmartInfrastructureMonitoringServer	3	1	4	4.00	4.00
StudentSupportProgramConnector	4	1	5	5.75	5.00
SupplyChainSpecificationVerifier	4	2	6	5.75	5.00
TeacherManager	4	1	5	4.75	4.00
Overall	**1,191**	**397**	**1,588**	**5.10**	**5.11**

B Details of Skill-based Baselines

- **AutoSkill** (Yang et al., 2026b) extracts reusable skills from user inputs to support continual improvement without updating model parameters. It maintains a skill library by adding distinct capabilities and merging refinements into existing skills, while filtering unsuitable or redundant candidates. For subsequent requests, relevant skills are retrieved and injected into the agent's context to guide its responses.
- **Trace2Skill** (Ni et al., 2026) distills experience from successful and failed execution trajectories into a reusable skill directory. Separate analysts process trajectories in parallel to propose skill patches, which are hierarchically consolidated to resolve conflicts and retain generalizable procedures. The resulting skill guides subsequent execution without parameter updates or retrieval of individual trajectory memories.
- **EvoSkill** (Alzubi et al., 2026) discovers and refines reusable skills through iterative failure analysis. A proposer diagnoses unsuccessful executions and suggests new or revised skills, which a skill-builder turns into structured skill folders containing procedural instructions and optional supporting resources. Candidate agent programs are evaluated on held-out validation data, and the highest-scoring programs are retained for further evolution while the underlying model remains frozen.
- **SkillOpt** (Yang et al., 2026a) optimizes a natural-language skill document while keeping the target model fixed. A separate optimizer model converts scored execution trajectories into textual edits under a limited edit budget, and each update is accepted only if it improves held-out validation performance. Rejected edits provide feedback for future proposals, while consolidation across epochs preserves longer-term lessons.

For all baselines, we directly adapt their official implementations to $\Delta\tau$-Bench without revising any modules and use DeepSeek-V3.2 as the common backbone.

C Details of Steward

Skill Structure and Initialization. Steward organizes procedural knowledge into a three-level skill library. Planning skills describe task-level workflows, while atomic skills encode reusable operations with explicit tool dependencies. Tool skills provide invocation guidance for individual tools. Each atomic skill stores a name, a description, reusable usage guidance, and its required tools. The skill graph links planning skills to associated atomic skills and atomic skills to the tool skills corresponding to their dependencies. At $v = 1$, we initialize the library following (Wang et al., 2026). Planning and atomic skills are extracted from training trajectories that pass environment-state evaluation. Planning extraction groups tool calls by their underlying subgoals, removing exploration, debugging, and failed attempts while preserving execution order. Atomic extraction focuses on one specified tool at a time, replacing task-specific values with reusable parameters and retaining relevant coordination with other tools. Tool-skill summarization additionally combines tool schemas with repeated tool-call failure evidence to identify invocation constraints and recovery procedures. The corresponding prompts used in this paragraph are detailed in Appx. D.1.

Incremental Knowledge Maintenance. After initialization, planning skills remain fixed, while Steward incrementally maintains atomic and tool skills using newly verified training trajectories. Given task metadata, the evaluation outcome, a trajectory summary, and retrieved skills, the manager proposes `add`, `modify`, `keep`, or `skip` decisions. Updates prioritize reusable workflow guidance, including tool dependencies and recovery procedures, while avoiding duplicate skills or changes unsupported by new experience. Cost-aware guidance preserves all task-required actions and favors cheaper alternatives only when trajectory evidence establishes the same preconditions and state effects. It may also identify redundant calls or known-invalid retries. The manager inherits both active and inactive records from the preceding ecosystem state and merges updates by skill name, with newer records replacing older ones. A tool skill is active when its associated tool exists in the current catalog, whereas an atomic skill is active only when all its declared tool dependencies are available. Missing dependencies are recorded, and inactive skills are retained for reactivation when their required tools become available again. Retrieval indices and the skill graph are rebuilt over the active subset. An auxiliary distillation step learns retrieval-policy skills from successful episodes in

which newly retrieved tools were subsequently invoked. These skills provide conditional guidance for retrieval timing, query formulation, or result assessment. They remain outside the hierarchical skill graph and do not directly change tool visibility. Prompts are detailed in Appx. D.2.

Initial Capability Provisioning. At the start of each episode, STEWARD retrieves the top five planning skills using the initial user request and asks an LLM to rewrite them into a task-specific provisional plan $\hat{p}$. This produces a structured list of actionable steps without modifying the stored planning skills. For each step, the manager retrieves up to five atomic skills subject to a similarity threshold. Candidates linked to the retrieved planning skills receive priority, followed by deduplication and maximal marginal relevance (MMR) to balance relevance and diversity. An LLM then selects at most 8 atomic skills based on the request and provisional plan. Their linked tool skills are attached to provide detailed invocation guidance, together forming the initial procedural context $k_0^{(v)}$. To complement skill-based retrieval, the manager directly retrieves the top three tools for each plan step. The union of these tools and the selected skills' declared dependencies is intersected with the current tool catalog to obtain the initial visible set $\widehat{\mathcal{A}}_0^{(v)}$. The visible tools and procedural context constitute the initial task-specific view $b_0^{(v)}$. The skill context is supplied as a system message at each turn without appending repeated copies to the dialogue history. It instructs the task agent to apply skills only when relevant and to rely on the actual session tool list when deciding which tools can be called. Prompts for this process are shown in Appx. D.3.

On-Demand Capability Provisioning. During execution, the task agent can call `retrieve_tools` with a focused capability query to retrieve up to five additional tools when its current tool set is insufficient. For gaps requiring further investigation, it can call `explore_capabilities` to delegate a focused search to the manager, with at most three delegations per episode. Each request specifies the task goal and missing capability, optionally including previously failed attempts. The manager also receives recent user messages and tool errors, the provisional plan, and the names of already visible tools and injected skills. It searches the active skill library and full tool catalog, using skill bodies and graph links to recover relevant procedural guidance. During this investigation, the manager cannot execute service tools or communicate with the user, and it must inspect each candidate tool's schema before recommending it. If a referenced tool no longer exists, it searches for a semantically equivalent replacement and explicitly reports the substitution. Among equivalent options, the prompt favors lower cost or latency. The investigation operates within an action budget and ends as soon as the delegated request can be answered. The manager returns a compact report specifying recommended tools and an actionable workflow, together with relevant pitfalls and rejected alternatives. At most 8 tools are returned per investigation, and a programmatic check admits only tools present in the current catalog into the task agent's visible set. When summarization would lose parameter-level or ordering details, the manager can additionally provide the full text of at most two skills. Skill-body delivery is limited to 4,000 characters cumulatively per episode, and each new added-skills block replaces the previous block. Providing a skill does not automatically expose its referenced tools, which must be inspected and admitted separately. These outputs jointly update the available tools $\widehat{\mathcal{A}}_t^{(v)}$ and procedural context $k_t^{(v)}$ as execution reveals new capability requirements. Please refer to Appx. D.4 for prompts used in this paragraph.

D PROMPTS

D.1 SKILL STRUCTURE AND INITIALIZATION

D.1.1 PLANNING SKILL EXTRACTION

Planning Skill Extraction

System Prompt:
You are a **Planning Expert.** Your job is to analyze an agent's API interaction history and the user's task, then distill them into a concise, reusable plan. This plan should serve as a reference for handling similar tasks more effectively in the future.

Objectives:

1. **Understand Capabilities.** Analyze the recorded API calls to identify the actual functional capabilities demonstrated.
2. **Abstract into a Plan.** For each feasible task supported by those capabilities, produce a concise, reusable step-by-step plan that can be applied to similar tasks.

Planning Creation Rules:

1. **Focus.** Do not simply restate each API function step-by-step using technical jargon. Instead, describe the underlying sub-goal behind each action segment.
2. **Remove Non-Essential Steps.** Exclude capability exploration, debugging, and failed steps.
3. **Reusability.** The plan must be precise enough for other models to reuse.
4. **Conciseness.** Merge steps from the interaction history that share the same objective into a single sub-step in the plan. Use a compact writing style for each sub-step, while listing the key APIs involved in that step (one or more). Do not omit any critical, potentially required API keys.

Output Format:
For each task, output exactly one plan and follow this format strictly:

```
<plan>
# step 1: A natural, specific, concise sub-task goal; key APIs used
(one or more).
# step 2: ...
...
</plan>
```

Good Examples:
{PLAN_EXAMPLE}

Checklist Before Finalizing:

- **Reusability.** Ensure no critical steps are missing and the step order is correct.
- **Conciseness.** Confirm there are no redundant or unnecessary steps.
- **Agent-centered.** Make sure the plan reads like actionable instructions that other models can reliably follow.

User Prompt:
User Task:
{USER_TASK}

Trajectory:
{TRAJECTORY}

D.1.2 Atomic Skill Extraction

Atomic Skill Extraction

System Prompt:
An agent system is provided with a **Skill Library** and has tried to solve the task multiple times with a successful solution. Review the task-solving attempt and extract generalizable skills.

Inputs Description:

- **User Task**
- **Trajectory:** A record of an agent's interactions successfully with the environment as it attempts to complete a user task.

- **Skill Library**: A collection of all currently available skills that can be directly reused.
- **Specific-Tool**: Given a specific tool, extract only one reusable skill for the specified tool.

Skill Definition Rule:
Skill is a dictionary with four keys: name, document, content, and tools.

1. name: the specific tool's name.
2. document: the tool's functionality, the key parameters, the final output of the skill and any important notes.
3. content: the tool's usage examples, and examples of combining it with other tools (if applicable).
4. tools: the key tools used in the content (list).

- The skill is centered around a specific tool, describing its core functionality, important notes, and common usage examples.
- Explicitly declare the key parameters and the final output data types using type hints. Example: Parameters: param: str; Outputs: output: dict:
- Include detailed description of the skill with input and output explanation.
- The skill should not be similar to the existing skills in the skill library.
- The parameters used in content must be reusable instead of hard-coded values (e.g., specific email address "jay@gmail.com").
- The usage examples of content may involve one or more tool uses.
- The document must clearly and thoroughly document all relevant details of the specific tool use.
- Never call other skills from the skill library or any previously defined skills.
- Do not import any Python packages.
- Avoid a functional style and Python code style; there's no need to use return.

Update Existing Skills:
Your goal is to ensure the system retains actionable skill that helps it behave correctly in the future.
You have three options: **[modify, add, keep]**

- **modify**: Revise an existing skill to make it more effective (e.g., improving documents). Only change content when necessary, and ensure the resulting skill remains broadly general-purpose.
- **add**: Introduce a new skill only when existing skill library missing the the specified tool.
- **keep**: Preserve the skill unchanged when there are no clear issues.
- If the **Skill Library** is empty, you may only use **add**.
- **modify** and **keep** are only valid for skills that already exist in the provided **Skill Library**.
- For **modify**, modified_from must exactly match an existing skill name from the provided **Skill Library**.
- For **keep**, skill_name must exactly match an existing skill name from the provided **Skill Library**.

Common actions:

- add a new skill
- update a skill's usage instructions/documentation
- revise a skill's variable/parameter definitions to make it more generalizable
- keep a skill unchanged
- ...

Requirements for Each Skill That Is Modified or Added:

- **Avoid duplication:** If a skill library is provided, do not add new skills that are similar to existing ones--use **keep** or **modify** instead.
- **Ensure domain specificity:** The skill must contain domain-specific tool.
- **Specific-Tool guided extraction:** Only focus on the specified tool in the trajectory when extracting skills.

Good Skill Example:
{GOOD_SKILL_EXAMPLE}

Output Format:
you will finish by returning in this JSON format as follows:

{OUTPUT_FORMAT_EXAMPLE}

Note that your updated skills may not need to cover all the options. You can only use one type of updates or choose to remain all skills unchanged.
Return a JSON array only. Each array item must be an object. option must be one of modify, add, or keep. For modify and add, skill must be an object with keys name, document, content, and tools, and tools must be a list of strings. If the provided **Skill Library** is empty, do not output any modify or keep item.

Checklist Before Finalizing:

- **Reusability.** Ensure no critical steps are missing, each skill is modular, all parameters are abstract rather than specific.
- **Optimality.** Ensure each skill meets the required definition standards.
- **Agent-centered.** Add helpful notes in each skill to guide other models in using it correctly.
- **Specific-Tool focus.** Whether the extracted skill doesn't center around this Tool?

User Prompt:
User Task:
{USER_TASK}

Trajectory:
{TRAJECTORY}

Skill Library:
{SKILL_LIBRARY}

Specific-Tool:
{TARGET_TOOL_NAME}

D.1.3 TOOL SKILL EXTRACTION

Tool Skill Extraction

System Prompt:
You are constructing a reusable tool skill for an agent skill library. Your task is to read the tool schema and repeated rollout evidence about a single tool-use failure pattern, then write ONE reusable tool skill that helps future agents avoid or recover from that failure.

Goal:
Produce a tool-centric skill that captures:

1. what the tool is for,
2. the important input/output details,
3. the specific failure pattern seen in the evidence,
4. the preconditions or parameter constraints needed to avoid that failure,
5. a generic recovery pattern.

Constraints:

- Output exactly one JSON object inside <skill>...</skill>.
- The JSON object must contain exactly these keys: name, document, content.
- name must be the target tool name, not a new abstract name.
- document should describe the tool, key parameters, outputs, and important notes from both schema and evidence.
- content should be reusable natural-language usage examples, not Python code.
- If the rollout evidence shows the tool is actually used in a way that differs from the schema, explicitly clarify that usage pattern in the skill instead of assuming the schema is the full story.
- Keep the skill generic and reusable; do not narrate a specific user story.

Output Format:

```
<skill>
{
  "name": "tool_name",
  "document": "...",
  "content": "..."
}
</skill>
```

User Prompt:
Tool Schema:
{TOOL_SCHEMA}

Repeated Failure Evidence:
{FAILURE_EVIDENCE}

D.2 Incremental Knowledge Maintenance

D.2.1 Incremental Skill Maintenance

Incremental Skill Maintenance

System Prompt:
You are updating a graph-backed skill library from one rollout session.
You will be given:

- task metadata and reward outcome
- a summarized trajectory
- planning skills retrieved into context
- atomic skills retrieved into context
- tool skills retrieved/attached into context

Your job is to decide which tool skills and atomic skills should be kept, modified, or newly added. The resulting skills are for a future agent to reuse while solving user tasks. They should describe agent-side workflow guidance, tool usage constraints, recovery behavior, and reusable execution patterns. Do not turn user behavior or conversation style into a skill unless it directly constrains how the agent should act.

Prioritization:

- Prefer updates that capture non-trivial, reusable agent guidance.
- Prefer sessions that reveal complex workflow, multi-step coordination, hidden dependency, recovery behavior, or easy-to-miss constraints.
- Prefer guidance that would help a future agent avoid mistakes, avoid dead ends, or handle a subtask that is somewhat hard rather than obvious.
- A successful session can still justify an update if it demonstrates a non-obvious workflow or a subtle but important control rule.
- If a subtask was solved in a simple, direct, obvious way and the session adds no meaningful new guidance, prefer "keep" or "skip".

High-Value Signals:

- lookup -> act, validate -> execute, preview -> confirm, or retrieve ID -> perform update patterns
- tool errors, retries, repair steps, argument correction, format correction, or fallback handling
- multi-tool coordination, order dependence, or context-building before action
- cases where the correct action is not just "call the tool directly"
- reusable guidance for difficult or failure-prone subtasks

Low-Value Signals:

- merely restating what a tool does
- retelling the task or conversation
- documenting a one-step happy-path call that is already obvious from the tool itself
- adding generic advice with no concrete workflow value

- overfitting to task-specific facts instead of extracting reusable guidance

Rules:

- Tool skills are one-tool-one-skill.
- If a tool already has a tool skill in context, you may only use option "modify" or "keep" for that tool.
- If a tool does not yet have a tool skill in context, you may only use option "add" or "skip".
- For tool skills, do NOT output a tools field; the system will set it automatically.
- Tool skill updates should focus on agent-facing pitfalls, prerequisites, verification steps, confirmation semantics, argument traps, and common recovery patterns for that tool.
- Tool skills should NOT mainly be API summaries, generic capability descriptions, or obvious parameter recitations.
- For existing atomic skills in context:
  - use "modify" only if the skill needs a real update
  - use "keep" if no update is needed
- You may use atomic "add" only for a genuinely new reusable capability that is not just a clarification of an existing atomic skill.
- Atomic skill updates should focus on reusable multi-step subtask workflow: sequencing, branching, validation, retry, recovery, and tool coordination.
- Atomic skills should NOT mainly capture trivial single-call behavior or task retelling.
- Prioritize reusable agent workflow guidance over task-specific narration.
- Do NOT force updates. If the session does not contribute meaningful new reusable guidance, choose keep/skip.
- Do NOT propose any planning skill updates here.

Output Format:
Output exactly one JSON object inside <proposal>...</proposal> with these keys:

- confidence: number between 0 and 1
- reasoning_summary: string
- tool_skill_updates: list[object]
- atomic_skill_updates: list[object]

Each tool update object must have:

- option: "add" | "modify" | "keep" | "skip"
- tool_name: string
- rationale: string
- skill: object or null

Each atomic update object must have:

- option: "add" | "modify" | "keep" | "skip"
- skill_name: string
- modified_from: string or ""
- rationale: string
- skill: object or null

For add/modify skill objects:

- atomic skill must contain name/document/content/tools
- tool skill must contain name/document/content

Cost-Aware Skill Guidance:
Correctness is the primary objective. Preserve every necessary state change, required precondition, confirmation, and task-required verification. Only when rollout evidence shows that candidates are semantically equivalent with the same preconditions and state effects may you prefer the lower-cost tool. Never describe a cheaper, merely similar tool as a substitute. Capture cost guidance only when evidence demonstrates an avoidable redundant call, an error/known-invalid retry, or an equivalent alternative. Prefer the minimal correct workflow: authenticate once, obtain required IDs once, perform required writes, and verify or report only when the task requires it.

User Prompt:
Task Metadata:
{TASK_METADATA}

Reward Outcome:
{REWARD}

Summarized Trajectory:
{TRAJECTORY_SUMMARY}

Planning Skills in Context:
{PLANNING_SKILLS}

Atomic Skills in Context:
{ATOMIC_SKILLS}

Tool Skills in Context:
{TOOL_SKILLS}

D.2.2 RETRIEVAL-POLICY SKILL DISTILLATION

Retrieval-Policy Skill Distillation

System Prompt:
You create concise retrieval-policy skills for an agent with a virtual retrieve_tools(query) tool.
Use only the supplied evidence. Write one conditional, reusable rule about when to retrieve, how to phrase a focused query, or how to assess retrieved schemas. Do not name task-specific IDs, invent tools, or describe business-tool APIs. Do not claim retrieval itself executes an action.

Output Format:
Return exactly:

```
<policy>
{
  "name": "...",
  "document": "...",
  "content": "...",
  "policy_type":
    "query_reformulation|retrieval_control|result_assessment"
}
</policy>
```

Evidence:

User Prompt:
{RETRIEVAL_EVIDENCE}

D.3 Initial Capability Provisioning

D.3.1 Task-Specific Pseudo Plan Construction

Task-Specific Pseudo Plan Construction

System Prompt:
You are rewriting retrieved planning skills into a task-specific pseudo-plan. Use the current request and the retrieved plans.

Output Format:
After thinking, return strict JSON only with the shape:

{"steps":["step 1","step 2"]}

Each step must be a short actionable string. Do not include commentary or any text outside the JSON.

User Prompt:
Current Request:
{QUERY_TEXT}

Retrieved Planning Skills:
{RETRIEVED_PLANS}

D.3.2 Skill Selection

Skill Selection

System Prompt:
Select the most applicable reusable skills for solving the current request. Consider the request and pseudo-plan.

Output Format:
Return strict JSON with the shape:

{"selected_indices":[1,2,...]}

Select at most {FINAL_MAX_SKILLS} skills.

User Prompt:
Current Request:
{QUERY_TEXT}

Pseudo-plan:
{PSEUDO_PLAN}

Candidate Skills:
{CANDIDATE_SKILLS}

D.3.3 Skill Context Injection

Skill Context Injection

System Context Template:
Reusable reference skills. Use them only when applicable to the current request. Any tools mentioned in these skills are only references and may not all be available in the current environment. You must rely on the actual tool list provided in this session when deciding which tools can be called.

Selected Atomic Skills:
{ATOMIC_SKILL_BLOCKS}

Tool Clarifications:
{TOOL_CLARIFICATION_BLOCKS}

D.4 On-Demand Capability Provisioning

D.4.1 Capability Investigation Interface

Capability Investigation Interface

Tool Name:
explore_capabilities

Tool Description:
Delegate a focused investigation to the manager. Use this when the currently available tools are clearly insufficient, when you cannot tell which tool expresses a specific constraint the user mentioned, or when a tool call failed because a required or unsupported parameter. The manager searches the reusable skill library and the full cross-domain tool catalog, inspects tool schemas (up to {MAX_ACTIONS} lookups), and returns a compact report plus a set of tools that become available to you. When a stored procedure contains parameter-level or ordering detail, it can also add that procedure verbatim to your context under # Added skills. It never executes any tool, so it has no side effects and does not change any data. Prefer retrieve_tools for a cheap single keyword lookup; use this only for harder cases.

Parameters:

- goal (string, required):
  What you are trying to accomplish, in one or two sentences. Include the concrete user constraints you already know.
- missing_capability (string, required):
  The specific capability or parameter you could not find with the tools you currently have.
- known_failures (string, optional):
  Tools you already tried and why they did not work.

D.4.2 Tool Retrieval Interface

Tool Retrieval Interface

Tool Name:
retrieve_tools

Tool Description:
Retrieve additional tools when the current tool list is insufficient. Pass one short, specific natural-language query describing the tool capability you still need. Use this only when the currently available tools do not cover the task.

Parameters:

- query (string, required):
  A short, specific description of the tool capability you want to retrieve.

D.4.3 On-Demand Capability Investigation

On-Demand Capability Investigation

System Prompt:
You are the manager. A main agent is solving a multi-turn task with tools, ran out of usable tools, and delegated a focused investigation to you.
You can search a reusable skill library, read skill bodies and walk the skill graph, search the full cross-domain tool catalog, inspect tool JSON schemas. You CANNOT execute any real tool and you never

talk to the user. Your only deliverable is a short report plus the list of tools the main agent should be given.

Rules:

1. Be economical. You have at most {MAX_ACTIONS} action calls in total. Start broad (search_tools / search_skills), then verify with inspect_tool. You MAY request several independent actions in the same turn (e.g. inspect three candidate tools at once)--doing so is cheaper and strongly preferred over one action per turn.
2. ALWAYS verify a tool with inspect_tool before recommending it. Never invent a tool name or a parameter name: only names returned by the backend exist.
3. Tools the main agent can already see are listed in the request. Do not recommend them again unless a parameter detail was the blocker.
4. Tool names change across catalog versions. If a skill references a tool that no longer exists, search for a semantically equivalent replacement and say so explicitly.
5. Prefer tools with lower cost / latency when several options are equivalent.
6. Call finish as soon as you can answer the request--do not keep exploring just because you still have budget left.

Addressing Skills (Important):

7. Every skill card carries a ref such as atomic:verify_engineer_credentials, tool_skill:record_reading, or planning:1a2b3c4d. read_skill and expand_graph accept ONLY that ref. Never pass a display name, a tool name, or a sentence copied from a card--it will not resolve.
8. If a lookup fails you get error plus did_you_mean. Use one of those suggestions instead of retrying the same string. An empty neighbors list with node_found: true means the node genuinely has no such edges--move on.
9. When you need to know HOW to call a tool (required parameters, valid enum values, preconditions, ordering), use search_tool_skills. inspect_tool gives you the schema; tool skills give you the learned usage rules.

Reporting:

10. Compress aggressively: the report goes into the main agent's context, so it must be short and actionable. Distill skills into imperative steps rather than pasting their bodies.
11. If a skill body contains parameter-level or ordering detail that would be lost by paraphrasing, put its ref in commit_skills (max {MAX_COMMIT_SKILLS})--the main agent will receive that procedure verbatim. Use it sparingly; it costs the main agent context.
12. commit_skills does NOT give the main agent the skill's tools. Any tool you want it to use must be inspected and listed in commit_tools.

Report Format:

finish(report=...) must follow this exact layout (omit a section if empty):

```
GOAL: <one sentence>
TOOLS:
  - <tool_name>: <why it is needed> | required: <comma separated required params> | cost=<c> latency=<l>
```

```
WORKFLOW:
  1.  <one line per step>
PITFALLS:
  - <one line per pitfall>
RULED_OUT:
  - <tool_name>:  <why it does not work>
```

Hard Limits:
At most {MAX_COMMIT_TOOLS} tools in TOOLS, at most 5 WORKFLOW steps, at most 3 PITFALLS, at most 3 RULED_OUT entries, and at most {MAX_REPORT_CHARS} characters overall.

User Prompt:
GOAL:
{GOAL}

MISSING CAPABILITY:
{MISSING_CAPABILITY}

KNOWN FAILURES:
{KNOWN_FAILURES}

RECENT USER MESSAGES:
{RECENT_USER_MESSAGES}

RECENT TOOL ERRORS:
{RECENT_TOOL_ERRORS}

PSEUDO-PLAN:
{PSEUDO_PLAN}

TOOLS ALREADY VISIBLE TO THE MAIN AGENT:
{VISIBLE_TOOL_NAMES}

SKILLS ALREADY INJECTED:
{INJECTED_SKILL_NAMES}