AGGREGATE SCORES MASK GENERALIST AGENT'S SKILL USEFULNESS

Anonymous authors
Paper under double-blind review

ABSTRACT

Recent advances in AI agents are moving toward generalist agents that acquire new capabilities through modular skills—reusable capabilities, knowledge, and tools that let agents perform new tasks. This raises a key evaluation question: *when an agent already has many skills, what does adding another skill actually change?* Standard with- or without-skill benchmarks evaluate skills in isolation, measuring whether they help on target tasks while overlooking whether they duplicate existing skills or degrade capabilities the agent already has. We reframe skill evaluation as an incremental, system-level attribution problem: we compare the agent before and after a skill addition across a 2×2 grid of system state and test case context, and introduce a capability correctness (CC) metric that scores the capability atomic operations inside a skill, crediting exact and functionally equivalent invocations. Applied to a generalist agent extended with seven open-source domains as skills (320 released test cases), we find that a single skill addition produces both gains on its target tasks and regressions on previously supported ones—including redundancy and cross-skill interference that aggregate benchmarks hide. Because CC localizes these effects to specific capabilities, it also yields an actionable diagnostic signal for skill refinement, which we probe with a simple targeted pruning method. Therefore, we treat aggregate scores as insufficient to attribute the true usefulness of a skill. [1]

1 INTRODUCTION

Over the past year, the landscape for building AI agents has changed rapidly from building domain-specific or workflow-centric agents (Barres et al., 2026; Lu et al., 2025) to building general-purpose agents (Shlomov et al., 2026; Research, 2025) where they can acquire new capabilities through skills. Skills can encapsulate capabilities, domain knowledge, predefined workflows, and other resources that enable an agent to perform specific tasks (Zhang et al., 2025; Lisboa, 2026). Skills allow a common agent to be extended to new domains easily, and provide a natural abstraction for constructing agents that operate across heterogeneous tasks and environments.

However, as the number of skills available to an agent grows, a fundamental evaluation question arises. *What does adding a particular skill actually change?* Existing evaluations of agent skills typically adopt a naive with-skill versus without-skill evaluation setup (Li et al., 2026; Han et al., 2026b) to measure the impact of the added skill. When a skill is added to an agent, the performance is often measured on a set of tasks for which the skill is expected to be useful. An improvement in task-level performance is then taken as evidence that the skill provides value. More often than not, these agents are evaluated with individual skills or task-specific skill sets only on associated tasks (Li et al., 2026), leaving their effects on the broader capabilities of the agent unexamined. This is particularly important in enterprise settings, where a general purpose agent (Shlomov et al., 2026) may simultaneously have access to hundreds of skills and a wide range of capabilities. The effect of a newly introduced skill may extend beyond the tasks for which it was designed. A skill may provide substantial gains on tasks within its target domain while being redundant or negatively impact tasks outside of its target domain. Skills vary in scope and format, from single capabilities to extensive

[1] Code and dataset are available at `https://github.com/mask-generalist/aggregate_mask_generalist_agent_skill_usefulness`.

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.

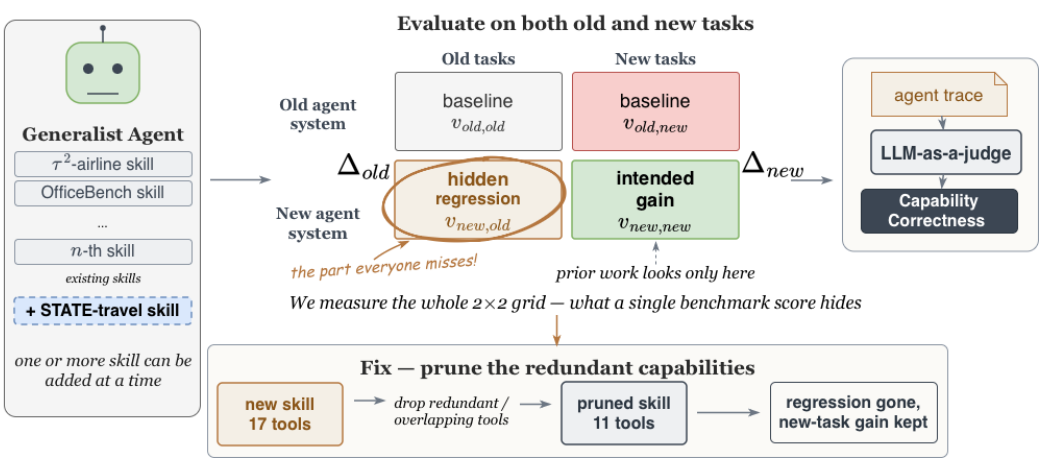


Figure 1: Overview of our incremental, system-level skill evaluation. A generalist agent with an existing skill library can be extended with one or multiple skills at a time. We evaluate the old system (existing skills) and the new system (+ one or many skills) on both old tasks and new tasks, forming a 2×2 grid evaluation. A single aggregate benchmark score reports only the intended gain on new tasks and hides regressions on previously supported tasks. Our capability correctness metric scores atomic operations within each skill from the agent trace, which in turn exposes redundant capabilities that can be pruned.

domain knowledge and workflows. This variability complicates whole-skill evaluation, as removing a skill may have negligible effects or disrupt multiple capabilities.

We define a *capability* as an atomic operation specified within a skill, which may be implemented as a Python function, a shell command, such as `cat` or `ls`, etc. Each skill comprises one or more such capabilities. Existing standard benchmarks (Li et al., 2026; Zhong et al., 2026) and conventional aggregate metrics (Chong et al., 2026; Ma et al., 2024) are unable to capture how adding a skill changes an agent that already possesses a diverse set of capabilities. This reveals a gap in existing skill evaluation: current approaches measure whether a skill improves performance, but do not characterize its *incremental* effect on the agent's broader set of capabilities.

We address this gap by approaching skill evaluation for generalist agent as an incremental system-level attribution problem. Rather than asking only whether an individual skill can improve performance, we ask how introducing a skill changes the behavior of an *already capable* agent. To address this gap, we evaluate new skills within an agent's existing skill library. Our framework compares the agent before and after skill addition on new and previously supported tasks, using systematically generated general, single-domain, and cross-domain test cases with ground truth outcomes. Additionally, we introduce a capability correctness metric that, together with the commonly used skill precision and recall metrics (Li et al., 2026), attributes a skill's utility at the fine-grained capability level. Our evaluation reveals how skill additions affect individual capabilities within arbitrarily authored skills, highlighting potential redundancy with existing capabilities and regressions on previously supported tasks.

We summarize our contributions as follows:

i) We formulate skill evaluation as an incremental, system-level attribution problem, and introduce a 2×2 grid evaluation spanning system state (existing skills vs. existing skills and new skill) and test cases context (new-skill vs. existing), capturing both newly enabled and existing capabilities.

ii) We introduce a fine-grained capability correctness metric that evaluates atomic operations within arbitrarily authored skills, revealing effects obscured by whole-skill scores.

iii) We provide an extensive set of experiments on a unified environment with seven open-source agent domains integrated as skills into two generalist agents and an automated test-generation pipeline for general, single-domain, and cross-domain tasks, releasing 320 test cases.

iv) We show in a case study how our capability correctness metric guides skill refinement by identifying potentially redundant or interfering capabilities, assessed through targeted pruning compared against a random removal control.

2 Related Works

Skill Evaluation. SkillsBench (Li et al., 2026), SkillLearnBench (Zhong et al., 2026), SWE-Skills-Bench (Han et al., 2026b), and Shaposhnikov et al. (2026) primarily compare skills against no-skill baselines on associated tasks, without directly evaluating additions to existing libraries or retention of supported skill capabilities. Separate-domain evaluations of CUGA (Shlomov et al., 2026; Marreed et al., 2025) likewise do not isolate skill interference. A related line of work measures regressions from an empty skill library (Tank & Nama, 2026), estimates skill effects across randomized selection of skill libraries for curation (Wang et al., 2026c), or generates capability-aligned tasks for only the newly supplied skills (Yu et al., 2026). Skill-Use (Han et al., 2026a) and Skill-Coach (Zhu et al., 2026) have experimented with distractor-augmented skill libraries. Although their experiments resemble the presence of other skills in our setting, the added skills serve as distractors for the evaluated tasks; their intended capabilities are not jointly assessed in those experiments. In our setting, existing skills have dedicated task suites retained from earlier skill additions, allowing subsequent skill library updates to be assessed against established skill capabilities. Therefore our evaluation paradigm assess how each skill addition changes performance on new and previously supported tasks, revealing gains and losses at the level of skill atomic capabilities as the skill library of the agent continue to evolve.

Test Case Generation. Skill evaluation heavily depends on the test cases, which raises the challenge of generating tests that exercise the capabilities encoded by an agent skill. TaskCraft (Shi et al., 2026) builds complex tasks by chaining or combining atomic tool-use tasks, AgentSynth (Xie et al., 2026) composes longer tasks from executed and verified subtasks, and GTA (Huang et al., 2026) generates solvable web tasks from crawled site graphs. MCP-Bench (Wang et al., 2026d) constructs tasks from tool-dependency chains, while MCP-Persona (Wang et al., 2026b) generates multi-step tasks over simulated applications. These methods derive tasks primarily from tools, environments, or workflows rather than skill artifacts. More directly related frameworks generate tests from the skills themselves. Shaposhnikov et al. (2026) derive tasks, execution environments, and evaluation rubrics from real-world skills. SkillTester (Wang et al., 2026a) and SkillAudit (Yu et al., 2026) generate capability-aligned tasks with explicit evaluation criteria, while SKT (Tan et al., 2026) and Terminal-World (Cheng et al., 2026) extend this to tasks involving multiple skills. STEPS (Wei et al., 2026) instead generates tasks by combining related capabilities from a categorized list rather than skill packages. Following these established approaches, we generate tests by sampling within- and cross-skill capability chains from documented workflows and LLM-proposed dependencies, with cross-domain cases combining capabilities from existing and newly added skills. We ground each task in database or workspace data and verify its expected outputs and subgoal assertions against it.

3 Methodology

3.1 Problem Formulation

Many generalist agents (Shlomov et al., 2026; Research, 2025) are commonly equipped with built-in tools such as *file-write*, *file-read*, *bash* commands, *web-search*, etc. A generalist agent's capability can be further extended with a set of curated skills. Let $\mathcal{A}$ denote a generalist agent. The agent's behavior is determined by the underlying large language model (LLM) M, the set S of skills that is available to the agent, and the environment ϵ in which the agent operates. The environment consist of the execution backend, databases, and the states from which all skills read and write.

We define the existing agent system $\mathcal{A}_{old}$ and the updated new system $\mathcal{A}_{new}$ as follows, where the latter adds s_{new} to the skill library S while keeping the model M and environment ϵ fixed:

$$\mathcal{A}_{old} = \mathcal{A}(\epsilon, M, S), \qquad \mathcal{A}_{new} = \mathcal{A}(\epsilon, M, S \cup \{s_{new}\}). \tag{1}$$

To evaluate the impact of adding s_{new}, we evaluate both old and new systems against the same task. Let P_D denote the distribution of evaluation tasks, where $x \sim P_D$ is a task sampled from the target

task distribution. We define $\mathrm{Eval}(\mathcal{A}, x)$ as the evaluation performance of the agent $\mathcal{A}$ on task x. The contribution of a new skill depends on the set of skills already available to the agent. We therefore define the marginal contribution of the new skill relative to the existing library skill S as follows:

$$\Delta(s_{\text{new}} \mid S; P_D) = \mathbb{E}_{x \sim P_D} \left[\mathrm{Eval}(\mathcal{A}_{new}, x) - \mathrm{Eval}(\mathcal{A}_{old}, x) \right]. \tag{2}$$

A key question we want to address in this work is *how should the evaluation task distribution P_D be constructed for a meaningful assessment of a skill's usefulness in an existing system?* Existing evaluations differ in both the agent configuration and the task distribution used to assess a new skill. Specifically, some work evaluates both old and new systems on a task distribution specifically constructed for only the new skill, without testing for regression on old, unrelated tasks. In addition, most of the literature compares agents equipped with task-specific skill sets against skill-free baselines (Li et al., 2026; Zhong et al., 2026), rather than agents with existing skill libraries:

$$\Delta_{task} = \mathbb{E}_{x \sim P_{task}} \left[\mathrm{Eval}(\mathcal{A}(\epsilon, M, S_{\text{task}}), x) - \mathrm{Eval}(\mathcal{A}(\epsilon, M, \emptyset), x) \right], \tag{3}$$

where S_{task} denotes the curated skills supplied for the task, which may include multiple skills. While this comparison measures the benefit of the supplied skill set, it does not isolate the effect of adding a new skill to an existing skill library and leaves regressions on other previously supported tasks unmeasured. Therefore in this work, we evaluate incremental skill additions for gains in new capabilities and retention of existing ones.

3.2 From Skill Gains to System-Wide Effects

To meaningfully understand the impact of a newly added skill to the existing system, we need to report evaluation scores separately for test cases drawn from the old and new distributions. Aggregating these evaluation scores on new and old tasks such as in Equation 2, can allow gains on new tasks to offset regressions on existing ones. Meanwhile evaluating only on new test cases relevant to the new skill, mask regression on old test cases. Hence, we propose to evaluate the existing agent system $\mathcal{A}_{old}$ that already has a set of existing skills S, and the new agent system A_{new} that acquires a new skill s_{new}, both on previously supported tasks drawn from P_{old} and new skill tasks drawn from P_{new}. We compute the following set of evaluation scores:

$$v_{i,j} = \mathbb{E}_{x \sim P_j} \left[\mathrm{Eval}(\mathcal{A}_i, x) \right], \quad \text{where } i, j \in \{old, new\}. \tag{4}$$

The subscript i indexes the agent system and the subscript j indexes the task distribution. Both agent systems are evaluated on the same sampled task within each distribution. Using the computed set of four scores $\{v_{i,j}\}$ and substituting it into Equation 2, we quantify the newly added skill's contribution on the new-skill tasks as:

$$\Delta_{new} := \Delta(s_{\text{new}} \mid S; P_{new}) = v_{new,new} - v_{old,new}, \tag{5}$$

and the change in previously supported capabilities is defined as:

$$\Delta_{old} := \Delta(s_{\text{new}} \mid S; P_{old}) = v_{new,old} - v_{old,old}. \tag{6}$$

A positive Δ_{new} indicates an improvement on new-skill tasks beyond the existing agent's performance showing positive value in adding the new skill. On the other hand, a negative Δ_{old} indicates a regression on previously supported tasks. Computing the two values separately allow us to understand the system-wide effects of adding the skill.

3.3 Skill Capability Correctness Metric

Skills authored by different users may vary in format and structure (Lisboa, 2026). We decompose each skill into capabilities, each represented by an atomic function. This provides a common basis for evaluating agent capabilities across different skill implementations. Refer to Appendix A.13 for test case examples of expected capabilities.

For each task x, there is a corresponding list of ground truth capabilities:

$$C^{\text{GT}}(x) = \left[c_1^{GT}(x), \ \ldots, \ c_{M(x)}^{GT}(x) \right], \tag{7}$$

where $M(x)$ is the number of ground truth capabilities for task x, and each $c_m^{GT}(x)$ is an atomic function comprising of the function name (if available), a natural-language description of the capability, and the expected outcome from executing the capability.

Given an agent execution trajectory τ, we utilize an LLM-as-a-judge to assess capability equivalence. It accepts either an exact function-name-and-output match or a functionally equivalent invocation (i.e., fuzzy match), assigning both the same score. A task capability correctness (CC) is:

$$\mathrm{CC}(x, \tau) = \frac{1}{M(x)} \sum_{m=1}^{M(x)} \text{LLM_judge}(c_m^{\mathrm{GT}}(x), \tau), \tag{8}$$

where LLM_judge returns 1 for either match type and 0 otherwise. Using the capability correctness metric to instantiate our $\mathrm{Eval}(\cdot, \cdot)$ function, we obtain the following score:

$$\begin{aligned} v_{i,j}^{\mathrm{CC}} &= \mathbb{E}_{x \sim P_j} \left[\mathrm{Eval}_{\mathrm{CC}}(\mathcal{A}_i, x) \right] \\ &= \mathbb{E}_{x \sim P_j} \left[\mathbb{E}_{\tau \sim p(\cdot \mid \mathcal{A}_i, x)} \left[\mathrm{CC}(x, \tau) \right] \right], \end{aligned} \qquad \text{where } i, j \in \{old, new\}. \tag{9}$$

Note that, $p(\tau \mid \mathcal{A}_i, x)$ denotes the trajectory distribution induced by agent $\mathcal{A}_i$ on task x. We report the resulting scores for the four corresponding agent configurations and task distributions. Besides the capability correctness metric, we also reported in our experiments other metrics such as skill recall and precision (formula in Appendix A.9.1), subgoal-based progress rate (Chong et al., 2026), $pass@k$, `pass^k` (Barres et al., 2026), latency, and tokens.

Proposition (Capability Correctness Is Invariant to Skill Grouping)**.** *Fix a task x, trajectory τ, and ground-truth capabilities $C^{\mathrm{GT}}(x)$ with $M(x) > 0$. Let S and S' group these capabilities such that each is assigned to exactly one skill in each library. If LLM_judge(c, τ) is independent of skill grouping, then*

$$\mathrm{CC}_S(x, \tau) = \mathrm{CC}_{S'}(x, \tau) = \mathrm{CC}(x, \tau), \tag{10}$$

where subscripts denote the grouping used. The proof is provided in Appendix A.8.

3.4 Test Case Generation Approach

We construct general, single-, and cross-domain tests for existing and newly added skills, retaining tests for the existing library S across skill additions. When s_{new} is introduced, we generate additional test cases that test its capabilities independently or in combination with existing skills. To generate these test cases, we follow prior dependency-based methods (Shen et al., 2024; Wang et al., 2026d; Abichandani et al., 2026) and build a capability graph from skill documentation, linking capabilities within and across skills through documented workflows and LLM-proposed dependencies. We sample capability chains for new-skill test cases, each containing at least one capability from s_{new}. For each sampled chain, we generate a task x, capability ground truth $C^{\mathrm{GT}}(x)$ specifying required capabilities and expected outcomes (Section 3.3), and subgoal assertions for progress rate metric (Chong et al., 2026). Outcomes and assertions are grounded in and verified against database or workspace data in the environment ϵ.

4 Datasets and Experimental Settings

Agent and Dataset. We augment two popular generalist agents, CUGA (Shlomov et al., 2026) and Pi (Zechner, 2025) with skills created using `agent-skill-creator` (Lisboa, 2026) in Claude Code (Anthropic) across seven domains: τ^2-retail and τ^2-airline from τ^2-Bench (Barres et al., 2026); OfficeBench (Wang et al., 2024); ToolSandbox (Lu et al., 2025); and STATE-travel, STATE-customer, and STATE-shopping from STATE-Bench (Microsoft, 2026). The skills are selected to study the overlapping capabilities of the agent on new and old test cases. In total, we release 320 test cases: 42 general, 103 single-domain, and 175 cross-domain testcases, together with our evaluation setup and environment testbed. Our setup provides more comprehensive coverage than some of the original domain-specific agents (e.g., τ^2-Bench and ToolSandbox) encompassing both individual domains and their combinations. The seven domain skills operate on shared database records in a unified environment ϵ. Further details are in Appendix A.10.

Experimental Settings. We construct three sets of experimental setups to assess the existing agent system $\mathcal{A}_{old}$ and new system $\mathcal{A}_{new}$ performance on both previously supported test cases $x \sim P_{old}$ and new test cases $x \sim P_{new}$. The existing skill libraries and newly added skills are listed in Table 1 for each setting, where Setting 3 is the most complex, followed by Setting 1, and finally Setting 2 is our base case. We build on the TED evaluation framework (Chong et al., 2026), extending it with

Table 1: Skills and test case domains per setting. Each setting comprises *general*, single-domain, and cross-domain test cases. See Table 11 in Appendix A.11 for the full enumeration.

Setting	$\mathcal{A}_{old}$ skills	+ New skill(s)	Old test cases ($x \sim P_{old}$)	New test cases ($x \sim P_{new}$)
S1	τ^2-*airline*, *OfficeBench*	*STATE-travel*	*general*, single-, and cross-domain over old skills	*STATE-travel* single-domain, and cross-domain with *OfficeBench* and *general*
S2	*(built-in tools only)*	*OfficeBench*	*general* only	*OfficeBench* single-domain, and cross-domain with *general*
S3	τ^2-*retail*, *ToolSandbox*	*STATE-customer*, *STATE-shopping*	*general*, single-, and cross-domain over the old skills	New skills single-domain, and cross-domain with the existing skills

our proposed CC metric, skill precision, and skill recall alongside the framework's progress rate and other available metrics. We run five trials per test case, using GPT-4.1 as the user proxy and GPT-5.4 as the LLM-as-a-judge for the CC and progress rate metrics.

5 Results and Discussion

5.1 Aggregate scores mask agent's skill performance

Table 2: Union of all old and new test cases spanning general, single-domain and cross-domain task types. Both agents use Claude Opus 4.6. Δ columns report the change in metric value of the new agent system relative to old system. Values annotated with blue indicate improvement, red indicate regression. The full table with other metrics can be found in Appendix A.12.

CUGA agent						**Pi agent**					
		Cap. corr.		Mean Prog.				Cap. corr.		Mean Prog.	
Set.	Config	Val.	Δ_{union}	Val.	Δ_{union}	Set.	Config	Val.	Δ_{union}	Val.	Δ_{union}
S1	Old sys	0.795		0.608		S1	Old sys	0.545		0.447	
	New sys	0.801	↑0.007	0.619	↑0.010		New sys	0.545	0.000	0.463	↑0.015
S2	Old sys	0.657		0.661		S2	Old sys	0.619		0.584	
	New sys	0.832	↑0.174	0.681	↑0.020		New sys	0.716	↑0.098	0.616	↑0.032
S3	Old sys	0.468		0.347		S3	Old sys	0.486		0.373	
	New sys	0.636	↑0.168	0.464	↑0.117		New sys	0.566	↑0.080	0.428	↑0.055

In Table 2, we show the performance of the CUGA and Pi agent with Opus 4.6 model on the union of old and new test cases spanning the general, single-domain, and cross-domain task types. Adding STATE-travel skill to a system with existing skills (Setting 1, Table 1), shows a marginal change in the capability correctness (CC) and progress rate metrics for either agent.

In Setting 2, both agents achieved better performance, in particular on the CC metric, when the OfficeBench skills are added to the new system. However, these aggregate improvements do not reveal whether the benefits extend to previously supported tasks or whether gains on new tasks mask regressions on existing tasks. Since Table 2 combines scores of existing test cases and new test cases across varying task types, we cannot properly identify where the regression or improvement comes from. Evaluating existing and new test cases separately is therefore necessary to understand the effects of skill addition which is the main motivation of our paper.

5.2 Measuring system-wide effects of skill addition

We separate new-skill test cases from those already supported by the existing agent. In addition, we examine the effects of skill addition across different task types. We analyze general and single-domain tasks in Table 3, cross-domain tasks in Table 4, and all tasks types in Table 15.

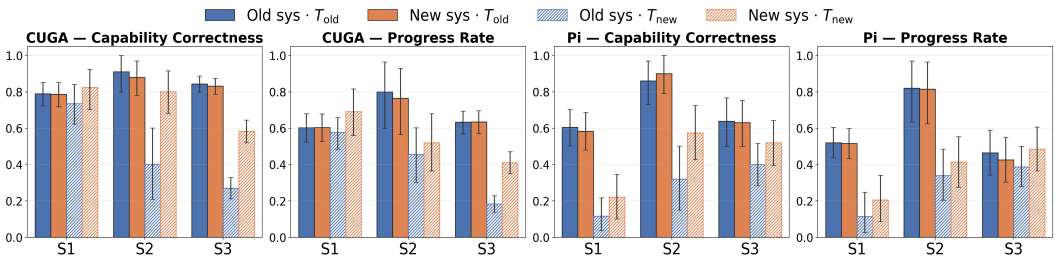


Figure 2: Bootstrap 95% confidence intervals (CI) for capability correctness and progress rate metric on the general + single-domain tasks, for Opus 4.6 CUGA and Pi agent across Settings 1-3. For other metrics, the CI plots are in Appendix A.15.

Evaluation on general and single-domain tasks. Table 3 shows consistent gains in the CC metric on new test cases for both CUGA and Pi, ranging from 9.0 to 39.9 percentage points (pp). However, adding new skills reduces CC score on previously supported test cases in five of the six reported settings, suggesting interference with existing capabilities that new-task evaluation alone would miss. In contrast, the aggregation in Table 2 dilutes these gains and masks any form of regression.

Table 3: **General+single-domain** test cases (Claude Opus 4.6). Δ columns report the change in metric value of the new agent system relative to old system. T_{old} and T_{new} denote old and new test cases, respectively. The full table with other metrics can be found in Appendix A.12.

		Capability correctness				Mean progress rate				Skill precision			Skill recall		
Setting	Config	T_{old}	Δ_{old}	T_{new}	Δ_{new}	T_{old}	Δ_{old}	T_{new}	Δ_{new}	T_{old}	Δ_{old}	T_{new}	T_{old}	Δ_{old}	T_{new}
CUGA S1	Old sys	0.789		0.735		0.603		0.577		0.961		–	0.971		–
	New sys	0.786	↓0.003	0.825	↑0.090	0.604	↑0.001	0.691	↑0.114	0.916	↓0.045	0.733	0.926	↓0.045	0.800
CUGA S2	Old sys	0.910		0.402		0.800		0.456		1.000		–	1.000		–
	New sys	0.880	↓0.030	0.801	↑0.399	0.765	↓0.035	0.520	↑0.064	0.800	↓0.200	1.000	0.800	↓0.200	1.000
CUGA S3	Old sys	0.844		0.270		0.633		0.183		0.913		–	0.917		–
	New sys	0.832	↓0.012	0.583	↑0.313	0.634	↑0.001	0.411	↑0.228	0.888	↓0.025	0.568	0.890	↓0.027	0.608
Pi S1	Old sys	0.605		0.117		0.520		0.114		0.526		–	0.526		–
	New sys	0.583	↓0.022	0.222	↑0.105	0.516	↓0.004	0.205	↑0.091	0.498	↓0.028	0.117	0.502	↓0.024	0.167
Pi S2	Old sys	0.860		0.320		0.820		0.339		1.000		–	1.000		–
	New sys	0.900	↑0.040	0.574	↑0.254	0.815	↓0.005	0.414	↑0.075	1.000	0.000	0.080	1.000	0.000	0.080
Pi S3	Old sys	0.639		0.400		0.465		0.387		0.597		–	0.600		–
	New sys	0.630	↓0.008	0.520	↑0.120	0.426	↓0.039	0.486	↑0.099	0.486	↓0.111	0.253	0.503	↓0.097	0.279

In Setting 3, the new CUGA system achieves 58.3% CC on new test cases, satisfying just over half of the expected capabilities on average. Skill recall is also limited to 60.8%, indicating incomplete invocation of the ground truth skills, which may have contributed to the low CC score.

On previously supported tasks in this setting, the new system shows slightly lower CC metric, skill precision, and skill recall than the old system, revealing a slight deterioration in both capability retention and skill selection as the new skill interferes with the existing skills.

This observation again raises the broader question of whether our capability metric can help identify parts of a skill that are redundant given the agent system's existing capabilities, which we further investigate later in Section 5.3. Meanwhile, Pi shows low skill recall on new test cases across all settings, with trajectory analysis revealing frequent reliance on built-in tools rather than the supplied skills. Figure 2 shows the confidence intervals of CUGA and PI agents for the CC and progress rate metrics. Appendix A.12 contains the full table with other metrics and CUGA GPT-5.6-sol performance.

Evaluation on cross-domain tasks. In Table 4, we show the evaluation results on cross-domain tasks. Cross-domain test cases are a special case: the new test cases are constructed from the newly added skill *combined with* the old skills. The combinations used for the different settings are shown in Table A.11. For CUGA the CC metric on new cross-domain tasks improves by 15.4 and 20.3

Table 4: **Cross-domain** test cases (Claude Opus 4.6). Δ columns report the change in metric value of the new agent system relative to old system. T_{old} and T_{new} denote old and new test cases, respectively. The full table with other metrics can be found in Appendix A.12.

		Capability correctness				Mean progress rate				Skill precision			Skill recall		
Setting	Config	T_{old}	Δ_{old}	T_{new}	Δ_{new}	T_{old}	Δ_{old}	T_{new}	Δ_{new}	T_{old}	Δ_{old}	T_{new}	T_{old}	Δ_{old}	T_{new}
CUGA S1	Old sys	0.845		0.805		0.585		0.687		0.908		–	0.947		–
	New sys	0.837	↓0.008	0.790	↓0.015	0.541	↓0.044	0.709	↑0.022	0.588	↓0.320	0.408	0.571	↓0.377	0.408
CUGA S2	Old sys	–		0.660		–		0.727		–		–	–		–
	New sys	–		0.814	↑0.154	–		0.758	↑0.031	–		1.000	–		1.000
CUGA S3	Old sys	0.629		0.278		0.518		0.206		1.000		–	0.800		–
	New sys	0.652	↑0.023	0.481	↑0.203	0.499	↓0.019	0.334	↑0.128	0.985	↓0.015	0.728	0.731	↓0.069	0.553
Pi S1	Old sys	0.671		0.548		0.503		0.408		0.406		–	0.394		–
	New sys	0.654	↓0.017	0.556	↑0.008	0.494	↓0.009	0.455	↑0.048	0.265	↓0.141	0.062	0.377	↓0.017	0.054
Pi S2	Old sys	–		0.676		–		0.593		–		–	–		–
	New sys	–		0.675	↓0.001	–		0.620	↑0.027	–		0.000	–		0.000
Pi S3	Old sys	0.631		0.401		0.474		0.292		0.250		–	0.236		–
	New sys	0.633	↑0.002	0.531	↑0.131	0.488	↑0.014	0.381	↑0.089	0.176	↓0.074	0.272	0.162	↓0.074	0.237

pp in Settings 2 and 3, respectively, but declines in Setting 1, where the old system already performs well. When new skill is introduced, Setting 1 also show declines in CC, progress rate, and in particular skill recall, on previously supported old cross-domain tasks, exposing disruption to existing workflows. The significant drop (37.7 pp) in the skill recall metric suggests that the new skill, i.e., STATE-travel, is likely interfering with the existing τ^2-airline skill for old workflow tasks that partially require τ^2-airline capabilities.

We see similar interference in Setting 3. The existing CUGA agent system has τ^2-retail and Toolsandbox skills, while the new system adds STATE-customer and STATE-shopping skills. On new test cases, the added skills improve CC by 20.3 pp. Meanwhile, on old supported test cases, skill recall drops by 6.9 pp when the skills are added. This suggests that the new skills have capabilities similar to the old skill library in the existing system that caused the agent to mistakenly use the new skill to solve old tasks.

For new test cases, one recurring source of interference we observed is the overlap between τ^2-retail and STATE-shopping skills. Both skills support product and order operations, but use different format and refund processes. Early routing errors can disrupt subsequent capability execution. For a task that requires product retrieval through STATE-shopping followed by a workspace file update, CUGA first invokes τ^2-retail and fails, then switches to STATE-shopping but reaches the turn limit. Together with regressions on previously supported tasks, this illustrates interference in both directions: old skills can hinder new tasks, while new skills can disrupt existing workflows, underscoring the need to evaluate both old and new test cases. Unlike single-domain evaluation, cross-domain test cases are designed to assess skill integration across existing and old domains, and their effects on existing capability combinations.

Evaluation on all tasks type. For completeness, we provide Table 15 combining all task types, i.e., general, single-domain, and cross-domain in Appendix A.12.

5.3 Pruning Redundant Skill Capabilities

Beyond measurement, CC provides an actionable diagnostic: it localizes which atomic capabilities in a newly added skill are redundant or interfere with the existing capabilities of the agent. We use Setting 1 (CUGA with Opus 4.6) as a case study of diagnostic-guided skill refinement, holding the existing skill library $S_{old} = \{\tau^2\text{-airline, OfficeBench}\}$ fixed and revising only STATE-travel,

Table 5: Performance after skill pruning on Setting 1. General + single-domain number of samples: $n_{\text{old}} = 49$, $n_{\text{new}} = 12$; cross-domain number of samples: $n_{\text{old}} = 17$, $n_{\text{new}} = 13$.

	General + single-domain				**Cross-domain**			
	CC		Mean progress		CC		Mean progress	
System	T_{old}	T_{new}	T_{old}	T_{new}	T_{old}	T_{new}	T_{old}	T_{new}
Old system	**0.789**	0.735	0.603	0.577	0.845	0.805	**0.585**	0.687
Unpruned	0.786	0.825	0.604	0.691	0.837	0.790	0.541	0.709
Random prune	0.771	0.736	0.617	0.623	0.838	**0.860**	0.535	**0.725**
CC-guided	0.780	**0.843**	**0.641**	**0.748**	**0.873**	0.838	0.524	0.693

simulating onboarding a new skill into an established system. CC credits both exact function-name matches and functionally equivalent invocations, and separating these reveals how each capability is fulfilled. For `search_flights`, only 3.3% of judgments recorded exact tool use, while 64.7% recorded a functionally equivalent invocation, often a direct database query. After adding STATE-travel, the agent also called its `search` tool _flights on old tasks, although these require the existing τ^2-airline skill. This reveals both redundancy on new tasks and cross-skill confusion on existing ones.

We calculate the proportions of exact matches, functional equivalences and no match, and then—guided by these statistical signals, we apply a simple pruning edit: we remove six capabilities with lowest exact-match rates while retaining frequently used ones, clarify the description to distinguish STATE-travel from airline ticketing, and remove duplicate text in `SKILL.md` (reduction in capabilities from 14 $\rightarrow$ 8; SKILL.md 106 $\rightarrow$ 72 lines). As a control, we remove six capabilities randomly. Table 5 shows point estimates favor CC-guided pruning on both T_{old} and T_{new}, but the 95% bootstrap intervals overlap across conditions, so we do not claim a significant gain and instead report pruning *as a diagnostic use of CC*. Full pruning results with confidence intervals for both Setting 1 and Setting 3 are in A.3, and the STATE-travel revision walkthrough are in Appendix A.4.

5.4 Human Validation

To validate our proposed capability score metric, we randomly selected 40 trials, covering 118 capabilities in total. Results show that the LLM judge agrees with human annotations on 94.1% of 118 randomly selected capabilities. This shows a high agreement rate between LLM-as-a-judge with human judgements. Detailed validation results and disagreement analysis are provided in Appendix A.14.

6 Conclusion

We reframed skill evaluation as an incremental, system-level attribution across a 2×2 grid of agent system state and test case context, and introduced a capability-correctness metric that attributes utility to atomic operations inside a skill. Our CC metric exposes changes that aggregate scores can obscure. Pruning experiments further illustrate how these insights can help identify redundant skill components. Together, these contributions support the evaluation of skill additions both by what they enable and how they affect capabilities already in place. We discuss limitations and directions for future work in Appendix A.2.

AI USE STATEMENT

In this work, we used generative AI tools for refining prompt templates; converting domain-specific agents into reusable agent skills; generating synthetic datasets and environments; and refining and polishing the paper's text to improve clarity and presentation. We also used generative AI tools to improve the layout and visual presentation of selected diagrams, figures, and tables.

We have not used generative AI tools to develop theoretical models or conceptual frameworks, formulate mathematical claims, provide critical ingredients for proving mathematical claims, assist in writing proofs, propose or refine hypotheses, or design or provide feedback on research methodology or experiments. Assistance with translation and qualitative or thematic data analysis was not applicable to this work, and generative AI tools were not used to interpret research results.

Additionally, generative AI tools were used to create or modify selected scientific figures and images. All AI-assisted content was reviewed by the authors. In particular, AI-generated or AI-assisted code was manually verified and tested for correctness by two authors, and all AI-assisted tables and figures were manually reviewed and verified before being included in the paper. For LLM-generated judgments, we also manually reviewed the results and compared them against human judgments, recording the alignment between the LLM and human evaluations. The authors take full responsibility for the final content of this work, including all text, claims, code, figures, tables, judgments, and other artifacts produced with the assistance of generative AI.

ETHICS STATEMENT

This work deals with generalist AI agents and skill evaluation and does not involve private user data. Experiments are conducted in controlled benchmark settings using public benchmarks and synthetic data. We therefore do not identify any specific ethical concerns.

REPRODUCIBILITY STATEMENT

We provide an anonymous public GitHub repository containing all code, skill implementations, evaluation configurations, testcases and detailed instructions to run our experiments. The paper and appendix includes results with additional standard metrics, test generation prompts and confidence interval plots. In addition, the code repository contains the raw experiment logs and aggregated evaluation scores so that researchers can independently inspect the agent trajectories and regenerate our results.

REFERENCES

Harshavardhan Abichandani, Penny Chong, Jiyuan Shen, Gunraj Singh, Ashutosh Hathidara, Marcus Duigan Xing Yu, Jane Lo, Atin Ghosh, Yipeng Li, and Daniel Dahlmeier. Edgegen: Improving tool-calling agents beyond happy paths with synthetic edge case generation, 2026. URL https://arxiv.org/abs/2609.24115.

Anthropic. Claude code. URL https://github.com/anthropics/claude-code.

Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik R Narasimhan. τ^2-bench: Evaluating conversational agents in a dual-control environment. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=OC2z7iSQKa.

Zihao Cheng, Hongru Wang, Zeming Liu, Xinyi Wang, Xiangrong Zhu, Yuhang Guo, Wei Lin, Jeff Z. Pan, and Yunhong Wang. Terminal-world: Scaling terminal-agent environments via agent skills, 2026. URL https://arxiv.org/abs/2605.20876.

Penny Chong, Harshavardhan Abichandani, Jiyuan SHEN, Atin Ghosh, Min Pyae Moe, Yifan Mai, and Daniel Dahlmeier. Talk, evaluate, diagnose: User-aware agent evaluation with automated error analysis. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=fHsVNklKOc.

Jinyi Han, Yuanjian Xu, Ying Liao, Xinyi Wang, Zishang Jiang, Zixiang Di, Fanyang Lu, Zhichao Hu, and Yanghua Xiao. Skill-use: Can LLMs actually use skills in agentic harnesses? *arXiv preprint arXiv:2608.04828*, 2026a.

Tingxu Han, Yi Zhang, Wei Song, Chunrong Fang, Zhenyu Chen, Youcheng Sun, and Lijie Hu. Swe-skills-bench: Do agent skills actually help in real-world software engineering? *arXiv preprint arXiv:2603.15401*, 2026b.

Tenghao Huang, Kung-Hsiang Huang, Prafulla Kumar Choubey, Yilun Zhou, Muhao Chen, Jonathan May, and Chien-Sheng Wu. GTA: Generating long-horizon tasks for web agents at scale. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 18805–18820, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-390-6. doi: 10.18653/v1/2026.acl-long.856. URL https://aclanthology.org/2026.acl-long.856/.

Xiangyi Li, Yimin Liu, Wenbo Chen, Bingran You, Zonglin Di, Yifeng He, Shenghan Zheng, Kyoung Whan Choe, Jiankai Sun, Shuyi Wang, et al. Skillsbench: Benchmarking how well agent skills work across diverse tasks. *arXiv preprint arXiv:2602.12670*, 2026.

Francy Lisboa. Agent skills platform, 2026. URL https://github.com/FrancyJGLisboa/agent-skills-platform.

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*, pp. 1160–1183, Albuquerque, New Mexico, April 2025. Association for Computational Linguistics. ISBN 979-8-89176-195-7. doi: 10.18653/v1/2025.findings-naacl.65. URL https://aclanthology.org/2025.findings-naacl.65/.

Chang Ma, Junlei Zhang, Zhihao Zhu, Cheng Yang, Yujiu Yang, Yaohui Jin, Zhenzhong Lan, Lingpeng Kong, and Junxian He. Agentboard: An analytical evaluation board of multi-turn LLM agents. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2024. URL https://openreview.net/forum?id=4S8agvKjle.

Sami Marreed, Alon Oved, Avi Yaeli, Segev Shlomov, Ido Levy, Offer Akrabi, Aviad Sela, Asaf Adi, and Nir Mashkif. Towards enterprise-ready computer using generalist agent. *arXiv preprint arXiv:2503.01861*, 2025.

Microsoft. STATE-Bench: Benchmark for enterprise workflows. GitHub repository, 2026. URL https://github.com/microsoft/STATE-Bench.

Nous Research. Hermes agent, 2025. URL https://github.com/NousResearch/hermes-agent.

Maksim Shaposhnikov, Nicolas Fortuin, Simon Stipcich, Maria I. Gorinova, Amy Heineike, and Rob Willoughby. A framework for evaluating agentic skills at scale. In *KDD workshop on Evaluation and Trustworthiness of Agentic AI*, 2026. URL https://openreview.net/forum?id=VqDJtvGnHH.

Yongliang Shen, Kaitao Song, Xu Tan, Wenqi Zhang, Kan Ren, Siyu Yuan, Weiming Lu, Dongsheng Li, and Yueting Zhuang. Taskbench: Benchmarking large language models for task automation. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 4540–4574. Curran Associates, Inc., 2024. doi: 10.52202/079017-0148. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/085185ea97db31ae6dcac7497616fd3e-Paper-Datasets_and_Benchmarks_Track.pdf.

Dingfeng Shi, Jingyi Cao, Qianben Chen, Weichen Sun, Weizhen Li, Hongxuan Lu, Fangchen Dong, Tianrui Qin, King Zhu, Minghao Liu, Yuchen Eleanor Jiang, Jian Yang, Ge Zhang, Jiaheng Liu,

Changwang Zhang, Jun Wang, and Wangchunshu Zhou. Taskcraft: Automated generation of agentic tasks. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=UJFCyrYM1V.

Segev Shlomov, Alon Oved, Sami Marreed, Ido Levy, Offer Akrabi, Avi Yaeli, Łukasz Strak, Elizabeth Koumpan, Yinon Goldshtein, Eilam Shapira, et al. From benchmarks to business impact: Deploying ibm generalist agent in enterprise production. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 40423–40431, 2026.

Zelin Tan, Yiqun Zhang, Hao Li, Zhiyao Cui, Hejia Geng, Shao Zhang, Hangfan Zhang, Yang Chen, Xiaosong Wang, Lilong Wang, Zhenfei Yin, Shuyue Hu, Chen Zhang, and Lei Bai. SKT: Skill-use training at scale via verified synthetic data generation, 2026. URL https://arxiv.org/abs/2608.02287.

Darshan Tank and Baran Nama. The regression tax: Decomposing why skills help and hurt LLM agents. *arXiv preprint arXiv:2607.22520*, 2026.

Leye Wang, Zixing Wang, and Anjie Xu. Skilltester: Benchmarking utility and security of agent skills, 2026a. URL https://arxiv.org/abs/2603.28815.

WenHao Wang, Peizhi Niu, Gongyi Zou, Xiyuan Yang, Jingxing Wang, Haoting Shi, Yaxin Du, Jingyi Chai, Xianghe Pang, Shuo Tang, Yanfeng Wang, and Siheng Chen. MCP-persona: Benchmarking LLM agents on real-world personal applications via environment simulation. In *Forty-third International Conference on Machine Learning*, 2026b. URL https://openreview.net/forum?id=vmT3p2w83l.

Yixuan Wang, Yiyang Zhou, Yiming Liang, Congyu Zhang, Fuxiao Liu, Jiawei Zhou, and Huaxiu Yao. Not all skills help: Measuring and repairing agent knowledge. *arXiv preprint arXiv:2606.15390*, 2026c.

Zhenting Wang, Qi Chang, Hemani Patel, Shashank Biju, Cheng-En Wu, Quan Liu, Aolin Ding, Alireza Rezazadeh, Ankit Parag Shah, Yujia Bao, and Eugene Siow. MCP-bench: Benchmarking tool-using LLM agents with complex real-world tasks via MCP servers. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations*, volume 2026, pp. 97377–97425, 2026d. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/9e4b14eb6f16fe7b5818a8d633a0606a-Paper-Conference.pdf.

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

Yifan Wei, Li Du, Xiaoyan Yu, Yang Feng, and Angsheng Li. Towards compositional generalization of LLMs via skill taxonomy guided data synthesis, 2026. URL https://arxiv.org/abs/2601.03676.

Jingxu Xie, Dylan Xu, Xuandong Zhao, and Dawn Song. Agentsynth: Scalable task generation for generalist computer-use agents. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=CoBxmXThM6.

Dexu Yu, Youhua Li, Zhaoyang Guan, Xianhao Lin, Jining Luan, Zihao Rao, Xuanqi Lan, Yang Ran, Bo Lan, Nai-Xin Zhai, et al. Skillaudit: From fixed-suite benchmarking to skill-centered assessment. *arXiv preprint arXiv:2606.22613*, 2026.

Mario Zechner. Pi: There are many agent harnesses but this one is yours, 2025. URL https://pi.dev/.

Barry Zhang, Keith Lazuka, and Mahesh Murag. Equipping agents for the real world with Agent Skills. Anthropic Engineering, October 2025. URL https://www.anthropic.com/engineering/equipping-agents-for-the-real-world-with-agent-skills.

Shanshan Zhong, Yi Lu, Jingjie Ning, Yibing Wan, Lihan Feng, Yuyi Ao, Leonardo F. R. Ribeiro, Markus Dreyer, Sean Ammirati, and Chenyan Xiong. Skilllearnbench: Benchmarking continual learning methods for agent skill generation on real-world tasks. In *Third Conference on Language Modeling*, 2026. URL https://openreview.net/forum?id=ulrZEJrNXA.

Jiayin Zhu, Kelong Mao, Yudong Guo, Dengbo He, Sulong Xu, Simiu Gu, and Yutao Yue. Skill-coach: Self-evolving rubrics for evaluating and enhancing agentic skill-use. *arXiv preprint arXiv:2607.01874*, 2026.

APPENDIX CONTENTS

A.1 Capability Correctness LLM-as-a-judge Prompt Template 15
A.2 Limitations 16
A.3 Detailed pruning results 16
A.4 STATE-travel revision details 19
A.5 STATE-customer-support revision details 20
A.6 STATE-shopping revision details 21
A.7 Prompt Template for Task Generation 22
A.8 Capability Correctness Is Invariant to Skill Grouping 24
A.9 Other Evaluation Metrics 25
A.9.1 Skill Precision and Recall 25
A.10 Agent and Dataset 25
A.11 Details of the three different experimental settings 27
A.12 Additional Experimental Results 28
A.13 Test cases Example 32
A.13.1 General Domain 32
A.13.2 Single Domain 32
A.13.3 Cross Domain 35
A.14 Human validation of capability judgments 37
A.15 Experiment Result Plots 37

A Appendix

A.1 Capability Correctness LLM-as-a-judge Prompt Template

Capability Validation Prompt

As a judge, your task is to assess whether the agent has invoked the capability specified in the [Ground Truth Capability] by examining the tool calls recorded in the [Agent Tool Call Steps].

A capability may be fulfilled by one tool call or by a sequence of multiple tool calls that together achieve the same intent. The agent does NOT need to use the exact capability or tool named in the [Ground Truth Capability]; using a functionally equivalent capability, tool, or combination of tools that achieve the same result is equally valid. For example, using `create_file` is equivalent to using `bash` with a `cat > file` redirect, and `query_database` is equivalent to using `bash` with a raw SQL client command—what matters is that the intended effect is achieved, not the specific tool name.

IMPORTANT — wrapper and path equivalence: When the expected capability refers to a specific command or script (e.g. `ocr.recognize_file` or `word/write_to_file.py`), and the agent invokes that same command or script through a shell wrapper (e.g. `run_command`, `bash`, `exec`), a subprocess call, or by specifying a different but valid path (absolute path, relative path, nested directory path) to the SAME script or function, this MUST be treated as the SAME tool or capability—not a different one. The invocation mechanism (direct call vs. shell wrapper) and the directory path used to reach the script are irrelevant to the match classification; only the identity of the underlying command/script and its parameters matter.

Classify the match into one of the following three categories:

- EXACT_MATCH: The agent invoked the same underlying command, script, or function as the expected capability, the output match the expected values, AND the invocation completed successfully. This includes cases where the capability is called through a shell wrapper (e.g. `run_command`, `bash`) or via a different directory path (absolute or relative) to the same script—the invocation mechanism and path style do not affect the match classification. IMPORTANT: The script or command name must correspond to the ground truth capability name. A custom or ad-hoc script written by the agent that achieves the same result does NOT qualify as EXACT_MATCH. A failed invocation of the exact tool also does NOT qualify—if the exact tool was called but failed, and the result was achieved by other means, this is not EXACT_MATCH.
- FUZZY_MATCH: The agent achieved the intended capability but used a genuinely different tool, script, or approach (not merely a different invocation mechanism or path), or produced the result under a different file name or path that serves the same function. The file paths, output values, and any specific paths mentioned in the capability description do NOT need to match the expected values—what matters is that the overall capability was fulfilled and the end result is functionally equivalent, regardless of the exact paths or file locations used.
- NO_MATCH: The agent did not invoke the capability at all, or the tool calls do not achieve the intended effect.

If the capability name is not specified, EXACT_MATCH is not possible—classify as either FUZZY_MATCH or NO_MATCH only.

`[BEGIN DATA]`
[Ground Truth Capability]:
  Capability name: {`capability_name`}
  Description: {`capability_description`}
  Expected output: {`expected_output`}
[Agent Tool Call Steps]: {`tool_call_steps`}
`[END DATA]`

Begin your response with a concise argument to confirm the validity of your conclusion. End with your answer formatted as **`GRADE: $LETTER`** (without quotes) where LETTER is one of **`E`**, **`F`**, or **`N`**. Reply with **`GRADE: E`** (without quotes) to indicate EXACT_MATCH. Reply with **`GRADE: F`** (without quotes) to indicate FUZZY_MATCH. Reply with **`GRADE: N`** (without quotes) to indicate NO_MATCH.

A.2 LIMITATIONS

While our experiments span diverse domains, further evaluation is needed to assess how the observed effects scale to larger skill libraries and longer sequences of additions. Test generation may overlook interactions, and LLM-based judging may introduce assessment errors (see Appendix A.14). Our pruning experiments illustrate the usefulness of capability-level insights, while systematic skill refinement remains future work. Several reported estimates have wide, overlapping 95% bootstrap confidence intervals, limiting confidence in individual differences. Although our analysis reveals patterns obscured by aggregate scores, differences without clear statistical support should be interpreted as directional evidence. Larger test suites and additional trials could improve the precision of these estimates.

A.3 DETAILED PRUNING RESULTS

Table 6 and 7 reports the detailed pruning results under setting 1 and 3. A clear bar chat that visualizes the pruning results for setting 1 is in Figure 3.

Table 6: Results by task type for Setting 1. Each cell shows the mean and its 95% bootstrap CI on the line below. Old system uses OfficeBench + τ^2-airline only (no STATE-travel skill). Unpruned system adds the new STATE-travel skill (14 tools). CC-guided pruned system removes 6 capability-metric-identified redundant tools from the new STATE-travel skill. The random prune system refers to randomly pruning 6 capabilities from STATE-travel skill.

(a) General + single-domain ($n_{old} = 49$, $n_{new} = 12$)

	CC		Mean progress		Tokens (M)	
System	T_{old}	T_{new}	T_{old}	T_{new}	T_{old}	T_{new}
Old system	**0.789** [0.724, 0.852]	0.735 [0.623, 0.841]	0.603 [0.524, 0.679]	0.577 [0.486, 0.659]	3.73 [2.63, 5.05]	5.53 [3.55, 7.93]
Unpruned	0.786 [0.718, 0.852]	0.825 [0.703, 0.925]	0.604 [0.528, 0.678]	0.691 [0.560, 0.817]	2.70 [2.14, 3.32]	4.50 [2.67, 6.58]
Random prune	0.771 [0.694, 0.844]	0.736 [0.631, 0.835]	0.617 [0.544, 0.689]	0.623 [0.499, 0.752]	8.70 [6.72, 10.94]	19.95 [14.37, 25.40]
CC-guided	0.780 [0.709, 0.846]	**0.843** [0.734, 0.936]	**0.641** [0.565, 0.716]	**0.748** [0.627, 0.860]	3.83 [2.92, 4.83]	6.87 [4.84, 9.25]

(b) Cross-domain ($n_{old} = 17$, $n_{new} = 13$)

	CC		Mean progress		Tokens (M)	
System	T_{old}	T_{new}	T_{old}	T_{new}	T_{old}	T_{new}
Old system	0.845 [0.769, 0.912]	0.805 [0.725, 0.883]	**0.585** [0.432, 0.729]	0.687 [0.524, 0.825]	6.94 [4.95, 8.95]	6.84 [4.40, 9.44]
Unpruned	0.837 [0.769, 0.902]	0.790 [0.685, 0.885]	0.541 [0.394, 0.679]	0.709 [0.566, 0.837]	4.10 [2.69, 5.75]	8.27 [5.20, 11.50]
Random prune	0.838 [0.747, 0.921]	**0.860** [0.777, 0.929]	0.535 [0.391, 0.679]	**0.725** [0.589, 0.845]	12.74 [8.96, 16.85]	16.23 [11.69, 21.37]
CC-guided	**0.873** [0.794, 0.941]	0.838 [0.766, 0.905]	0.524 [0.379, 0.671]	0.693 [0.555, 0.817]	7.62 [5.63, 9.51]	9.55 [6.40, 12.98]

(c) General + single-domain + cross-domain ($n_{old} = 66$, $n_{new} = 25$)

	CC		Mean progress		Tokens (M)	
System	T_{old}	T_{new}	T_{old}	T_{new}	T_{old}	T_{new}
Old system	**0.804** [0.752, 0.854]	0.772 [0.700, 0.838]	0.599 [0.530, 0.667]	0.634 [0.537, 0.722]	4.56 [3.50, 5.68]	6.21 [4.61, 7.98]
Unpruned	0.799 [0.743, 0.852]	0.807 [0.727, 0.877]	0.588 [0.517, 0.654]	0.700 [0.606, 0.790]	3.06 [2.49, 3.72]	6.46 [4.53, 8.57]
Random prune	0.788 [0.726, 0.847]	0.801 [0.731, 0.865]	0.596 [0.528, 0.662]	0.676 [0.584, 0.767]	9.74 [7.91, 11.73]	18.02 [14.36, 21.84]
CC-guided	**0.804** [0.746, 0.857]	**0.840** [0.779, 0.899]	**0.611** [0.542, 0.678]	**0.720** [0.631, 0.805]	4.81 [3.88, 5.79]	8.26 [6.21, 10.41]

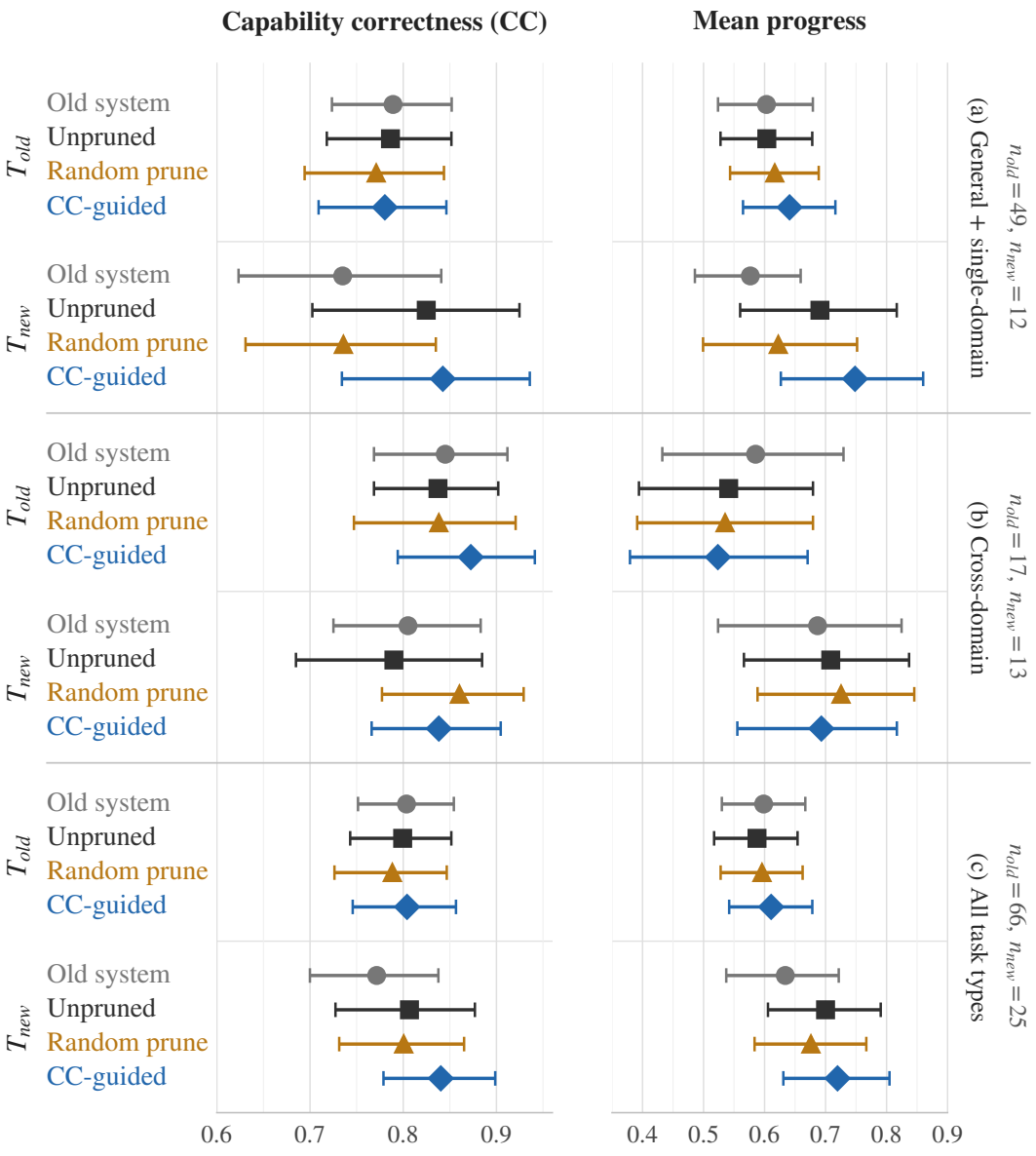


Figure 3: CC and mean progress for the old system and three various versions of the new system under setting 1. Read each column vertically: T_{old} is above T_{new} for each metric. Points are means; error bars are 95% bootstrap CIs from 10,000 resamples of test cases.

Table 7: Results by task type for Setting 3. Each cell shows the mean and its 95% bootstrap CI on the line below. Old system uses τ^2-retail + toolsandbox only (no customer-support or shopping skills). Unpruned adds STATE-customer (11 tools) and STATE-shopping (18 tools). CC-guided pruned removes 8 capability-metric-identified redundant tools (state-customer-support: 7 tools; state-shopping: 14 tools).

(a) General + single-domain ($n_{old} = 60$, $n_{new} = 74$)

	CC		Mean progress		Tokens (M)	
System	T_{old}	T_{new}	T_{old}	T_{new}	T_{old}	T_{new}
Old system	**0.844** [0.800, 0.887]	0.270 [0.212, 0.329]	0.633 [0.568, 0.694]	0.183 [0.139, 0.229]	3.24 [2.66, 3.91]	6.72 [5.96, 7.53]
Unpruned	0.832 [0.787, 0.874]	0.583 [0.520, 0.645]	0.634 [0.570, 0.696]	0.411 [0.350, 0.471]	3.68 [2.95, 4.49]	10.24 [9.07, 11.46]
CC-guided	0.828 [0.784, 0.870]	**0.603** [0.547, 0.657]	0.610 [0.542, 0.676]	**0.449** [0.387, 0.510]	7.62 [6.35, 8.94]	18.31 [16.28, 20.37]

(b) Cross-domain ($n_{old} = 13$, $n_{new} = 52$)

	CC		Mean progress		Tokens (M)	
System	T_{old}	T_{new}	T_{old}	T_{new}	T_{old}	T_{new}
Old system	0.628 [0.477, 0.768]	0.278 [0.212, 0.349]	**0.518** [0.408, 0.612]	0.206 [0.158, 0.258]	6.33 [4.54, 8.50]	8.54 [7.45, 9.69]
Unpruned	0.652 [0.504, 0.783]	0.481 [0.406, 0.559]	0.498 [0.388, 0.601]	0.334 [0.278, 0.392]	7.32 [4.82, 10.77]	12.45 [10.73, 14.32]
CC-guided	**0.658** [0.513, 0.797]	**0.481** [0.415, 0.548]	0.515 [0.396, 0.628]	**0.341** [0.290, 0.393]	15.24 [11.13, 21.52]	21.05 [17.88, 24.58]

(c) General + single-domain + cross-domain ($n_{old} = 73$, $n_{new} = 126$)

	CC		Mean progress		Tokens (M)	
System	T_{old}	T_{new}	T_{old}	T_{new}	T_{old}	T_{new}
Old system	**0.805** [0.757, 0.852]	0.273 [0.230, 0.318]	**0.612** [0.556, 0.668]	0.193 [0.160, 0.227]	3.79 [3.15, 4.51]	7.47 [6.83, 8.14]
Unpruned	0.800 [0.751, 0.845]	0.541 [0.491, 0.590]	0.610 [0.553, 0.666]	0.379 [0.335, 0.423]	4.32 [3.51, 5.28]	11.15 [10.13, 12.22]
CC-guided	0.798 [0.750, 0.843]	**0.553** [0.507, 0.597]	0.593 [0.534, 0.652]	**0.404** [0.361, 0.448]	8.98 [7.49, 10.70]	19.44 [17.69, 21.29]

A.4 STATE-TRAVEL REVISION DETAILS

For the CC-guided new system, only the newly added STATE-travel skill is changed. The existing τ^2-airline and OfficeBench skills and all capability requirements remain fixed. Table 8 gives entire match statistics for the removed tools and retained booking, cancellation, and lookup operations.

Table 8: The capability match statistics before pruning. Six redundant capabilities are removed and the rest are retained. E: exact match; F: functionally equivalent invocation; N: no match. Percentages are calculated from the reported counts, with *n* denoting pooled capability judgements on T_{new}.

Capability	n	E (%)	F (%)	N (%)	Edit
`get_flight_status`	275	13.5	80.4	6.2	Remove
`get_user_details`	100	7.0	67.0	26.0	Remove
`search_hotels`	75	0.0	65.3	34.7	Remove
`search_flights`	150	3.3	64.7	32.0	Remove
`create_booking`	50	0.0	60.0	40.0	Remove
`update_booking`	150	0.0	57.3	42.7	Remove
`get_booking`	225	46.7	51.1	2.2	Retain
`search_car_rentals`	75	41.3	45.3	13.3	Retain
`book_hotel`	50	48.0	42.0	10.0	Retain
`cancel_car_rental`	125	75.2	8.0	16.8	Retain
`cancel_booking`	75	53.3	6.7	40.0	Retain
`cancel_hotel_reservation`	125	77.6	6.4	16.0	Retain
`book_car_rental`	50	90.0	0.0	10.0	Retain
`get_policies`	350	9.7	42.3	48.0	Retain

Clarifying the skill's scope. The original description overlaps with the airline skill's scope. The revised description identifies the reservation system it owns and explicitly excludes standalone airline search and ticketing. Below is the complete description change.

```
@@ SKILL.md: description (reflowed) @@
- Travel booking & service. Use when acting as a customer-service agent
- for a travel company handling flight/hotel/car bookings, changes,
- cancellations, delays, fees, loyalty, and multi-product trip coordination.
- Provides a small stateful sandbox database and the domain tools via a CLI.
+ This travel company's own reservation back-office over a local world.db:
+ manage EXISTING hotel and car-rental reservations and this company's
+ packaged bookings - look them up, price cancellation/change fees, and
+ confirm cancellations. NOT a general airline reservation service; do not
+ use for standalone airline search/ticketing tasks. Domain tools run via a CLI.
```

Removing redundant interfaces. The six removed tools have low exact-match rates and are often replaced by the agent's own database operations; flight-related interfaces also appear in misrouted airline traces. We remove their entries from `SKILL.md`, the exported schema list, and the handler registry. Removing the registry entries makes the tools unavailable through the CLI; the underlying function definitions remain in the source. The excerpts below show the schema and write-tool changes, with unchanged entries and comments omitted. This retains booking, cancellation, and supporting lookup operations rather than removing the entire skill.

```
@@ tools.py: exported schemas (unchanged entries omitted) @@
 TOOL_SCHEMAS: list[dict[str, Any]] = [
-    SEARCH_FLIGHTS,
-    GET_USER_DETAILS,
     GET_USER_RESERVATIONS,
     GET_BOOKING,
-    GET_FLIGHT_STATUS,
     ...
-    CREATE_BOOKING,
-    UPDATE_BOOKING,
     CANCEL_BOOKING,
-    SEARCH_HOTELS,
     BOOK_HOTEL,
     ...
 ]
@@ tools.py: write-tool list (unchanged entries omitted) @@
 WRITE_TOOL_NAMES: list[str] = [
-    "create_booking",
-    "update_booking",
     "cancel_booking",
     ...
 ]
```

A.5 STATE-CUSTOMER-SUPPORT REVISION DETAILS

For the CC-guided new system, only the newly added STATE-customer-support skill is changed. The existing τ^2-Retail and toolsandbox-assistant skills and all capability requirements remain fixed. Table 9 gives the capability match statistics for the removed tools and the retained customer, order, warranty, and write operations.

Table 9: Capability match statistics before pruning for STATE-customer-support. Four redundant capabilities are removed and the rest are retained. E: exact match; F: functionally equivalent invocation; N: no match. Percentages are calculated from the reported counts, with *n* denoting pooled capability judgements on T_{new}.

Capability	n	E (%)	F (%)	N (%)	Edit
`process_refund`	25	0.0	4.0	96.0	Remove
`get_policies`	1150	8.8	44.8	46.4	Remove
`get_product_details`	1225	21.1	42.9	35.9	Remove
`search_products`	400	10.7	19.8	69.5	Remove
`get_order`	1000	3.1	40.8	56.1	Retain
`get_warranty_status`	350	1.4	34.6	64.0	Retain
`get_customer`	550	21.3	27.8	50.9	Retain
`cancel_order`	375	1.9	26.1	72.0	Retain
`process_exchange`	375	1.9	17.9	80.3	Retain
`process_warranty_claim`	200	0.0	17.0	83.0	Retain
`process_return`	575	0.9	12.0	87.1	Retain

Removing redundant interfaces. The four removed tools are either dead (`process_refund`: 0% completion, 8 invocations across all trials) or duplicate capabilities already provided by co-loaded skills: `get_product_details` and `search_products` appear under the same names in both STATE-shopping and τ^2-Retail, causing the agent to bypass all three versions in favour of direct SQLite queries; `get_policies` is identically named in STATE-shopping. We remove their entries from `SKILL.md` only; the backing code is unchanged.

```
@@ SKILL.md: How this skill works @@
```

```
-3. Read policy from [references/policies.md](references/policies.md) or the `get_policies`
     tool.
+3. Read policy from [references/policies.md](references/policies.md).

@@ SKILL.md: Tools  Read tools @@
 - `get_order`  Retrieve full details of an order by its order ID. ...
 - `get_customer`  Retrieve customer profile including membership tier, ...
-- `search_products`  Search the task-local product catalog by free-text query. ...
-- `get_product_details`  Retrieve full product details by exact product ID, ...
-- `get_policies`  Look up the company's policies for a given topic. ...
 - `get_warranty_status`  Check the warranty status for a specific item. ...

@@ SKILL.md: Tools  Write tools @@
 - `process_return`  Process a return for an order item. ...
-- `process_refund`  Process a refund for an order item. ...
 - `cancel_order`  Cancel an order or specific items within an order. ...

@@ SKILL.md: Operating procedure @@
-2. **Check policy** with `get_policies` (some write tools require this first) and consult
-   [references/policies.md](references/policies.md).
+2. **Check policy** by consulting [references/policies.md](references/policies.md).
```

A.6 STATE-SHOPPING REVISION DETAILS

For the CC-guided new system, only the newly added STATE-shopping skill is changed. The existing τ^2-Retail and toolsandbox-assistant skills and all capability requirements remain fixed. Table 10 gives the capability match statistics for the removed tools and the retained cart management, product discovery, and checkout operations.

Table 10: Capability match statistics before pruning for STATE-shopping. Four redundant capabilities are removed and the rest are retained. E: exact match; F: functionally equivalent invocation; N: no match. Percentages are calculated from the reported counts, with n denoting pooled capability judgements on T_{new}. Tools marked S1 were never invoked in any trace and have no match statistics.

Capability	n	E (%)	F (%)	N (%)	Edit
remove_promo	—	—	—	—	Remove (S1)
cancel_loyalty_redemption	—	—	—	—	Remove (S1)
validate_promo	100	0.0	57.0	43.0	Remove
redeem_loyalty_points	125	0.0	52.8	47.2	Remove
get_shipping_options	325	13.2	72.6	14.2	Retain
set_shipping_option	275	22.9	51.6	25.5	Retain
update_cart_item	325	15.4	51.1	33.5	Retain
get_policies	1150	8.8	44.8	46.4	Retain
get_product_details	1225	21.1	42.9	35.9	Retain
get_customer_account	425	27.1	43.3	29.6	Retain
check_compatibility	425	6.1	41.9	52.0	Retain
remove_from_cart	350	39.4	41.4	19.1	Retain
get_cart	625	39.0	39.2	21.8	Retain
apply_promo	50	18.0	32.0	50.0	Retain
get_promotions	150	46.0	28.0	26.0	Retain
get_variants	300	33.3	21.3	45.3	Retain
search_products	400	10.7	19.8	69.5	Retain
add_to_cart	175	37.7	16.0	46.3	Retain

Removing redundant interfaces. Two tools were never invoked in any of the 995 evaluation trials (`remove_promo`, `cancel_loyalty_redemption`; S1 signal) and two were bypassed with 0% exact-match rate across all trials (`validate_promo` and `redeem_loyalty_points`: agent used direct SQLite writes instead of the documented tools). We remove their entries from `SKILL.md` only; the backing code is unchanged.

```
@@ SKILL.md: Tools  Read tools @@
 - `get_policies`  Look up the store's policy for a given topic. ...
-- `validate_promo`  Check whether a promo code would validate against the customer's current
     cart. ...
 - `get_shipping_options`  List shipping options available for the customer's cart. ...
```

```
@@ SKILL.md: Tools  Write tools @@
 - `apply_promo`  Apply a promo code to the cart. ...
-- `remove_promo`  Remove a previously applied promo code from the cart. ...
-- `redeem_loyalty_points`  Redeem the customer's loyalty points for a dollar discount on the
     current cart. ...
-- `cancel_loyalty_redemption`  Cancel a prior redemption on the current cart, crediting
     points back. ...
 - `set_shipping_option`  Write a shipping option to the cart. ...
```

A.7 Prompt Template for Task Generation

Skill Extraction Prompt

```
Read every file in this directory (SKILL.md, policies.md, any other docs)
and extract a structured skill definition.

Return ONLY valid JSON -- no prose before or after. No markdown fences.

The JSON must have this exact structure:
{
  "name": "<skill name from SKILL.md frontmatter>",
  "description": "<one-sentence skill description>",
  "capabilities": [
    {
      "name": "<tool/capability name -- snake_case matching the tool name in SKILL.md>",
      "description": "<what this capability does>",
      "inputs": {
        "<param_name>": {
          "type": "<string|integer|boolean|array|object>",
          "description": "<what it is>",
          "required": true|false
        }
      },
      "output_schema": {
        "<field>": {
          "type": "<type>",
          "description": "<what it returns>"
        }
      },
      "invocation_pattern": "<how to call it, e.g. tools.get_order('<order_id>')>"
    }
  ],
  "flows": [
    {
      "name": "<flow name -- a realistic multi-step user task>",
      "steps": [
        "<capability_name_1>",
        "<capability_name_2>",
        ...
      ]
    }
  ]
}

CRITICAL -- name and ID fidelity:

- capability names must EXACTLY match the tool/function names as written in
  SKILL.md -- do not rename, re-case, or paraphrase them

- Parameter names in "inputs" must EXACTLY match the parameter names shown in
  the tool signature or parameter table in SKILL.md -- copy them verbatim,
  do not substitute synonyms (e.g. if SKILL.md says "reservation_id" do NOT
  write "booking_id"; if it says "flight_number" do NOT write "flight_id")

- Output field names must also match the documented return fields exactly

- If a tool uses "cabin" do not write "cabin_class"; if it uses "option"
  do not write "shipping_option"

Other rules:

- Extract ONE capability per distinct tool/command listed in SKILL.md

- Do NOT include capabilities that are not listed as tools in SKILL.md

- For inputs: include EVERY parameter the tool accepts, with correct types
```

```
  and required flags -- do not omit optional parameters

- For output_schema: describe the shape of what the tool returns

- Make 2-4 flows representing realistic multi-step user tasks for this skill

- Respond with ONLY the JSON object, nothing else
```

Test-Case Generation Prompt

```
You are an expert at creating test cases for AI agent evaluation.

CRITICAL RULES:
- Use ONLY entity values (IDs, names, amounts, dates) from the
  DATABASE SNAPSHOT
- Do NOT invent any IDs, names, or values
- task_summary must start with "You are ..." (user-proxy perspective)
- arguments keys must EXACTLY match the parameter names in the
  input_schema -- copy key names verbatim, never substitute synonyms
  (e.g. "reservation_id" not "booking_id", "flight_number" not
  "flight_id", "option" not "shipping_option", "content" not
  "message", "contents" not "text")
- arguments values must be exact values from the DB snapshot
- expected_output must be a realistic representation of what the
  tool returns, using DB values
- EXCEPTION: for tools that CREATE new entities
  (e.g. book_reservation, book_hotel, book_car_rental,
  create_booking), the newly generated ID does NOT exist in the
  snapshot yet. For those output fields (reservation_id,
  booking_id, rental_id, etc.), use a short natural-language phrase
  like "newly generated reservation ID" or
  "newly generated booking ID" -- do NOT reuse an input entity ID
  (flight_id, hotel_id, car_id) as the output
  booking/reservation/rental ID
- subgoals are assertions about the AGENT's final response --
  what it said or did
- All subgoals must reference specific values from the DB snapshot

Generate a test case for evaluating an AI agent executing this
capability chain.

## Capability Chain
{chain_summary}

## Skill Definitions (for schema context)
{skill_definitions}

## Database Snapshot (ONLY source of real values)
{db_snapshot}

## Feasibility Note
{feasibility}

---

Generate a JSON object with this exact structure:

{
  "task_summary":
    "You are [real name from DB, include their ID/account if relevant].
     You want to [natural user request that requires ALL capabilities
     in this chain]. [2-3 sentences including relevant context with
     real values from DB].",

  "expected_capabilities": [
    {
      "name": "<capability name>",
      "skill": "<skill name>",
      "description": "<capability description>",

      "input_schema": {
        "<param>": {
          "type": "<type>",
          "description": "<what it is>",
          "required": true|false
        }
      },
```

```
      "output_schema": {
        "<field>": {
          "type": "<type>",
          "description": "<what it returns>"
        }
      },

      "arguments": {
        "<param>": <real value from DB snapshot>
      },

      "expected_output": {
        <realistic output using real DB values -- keys MUST be taken
         verbatim from output_schema above, nothing else.

         EXCEPTION: for creation tools (book_reservation, book_hotel,
         book_car_rental, create_booking), use a natural-language
         phrase like "newly generated reservation ID" for the new
         entity ID -- never reuse an input entity ID as the output ID.>
      }
    }
  ],

  "subgoals": [
    {
      "description":
        "<one sentence assertion about what the AGENT must say or do,
         referencing real values>"
    },
    ...
  ]
}

Rules:

1. expected_capabilities must include ALL {n_caps} capabilities in
   the chain, in order

2. arguments must use ONLY values that appear in the DB snapshot

3. CRITICAL -- argument keys must EXACTLY match the parameter names
   in the input_schema above -- copy the key names verbatim
   (e.g. if input_schema has "reservation_id" the argument key MUST
   be "reservation_id", NOT "booking_id"; if input_schema has
   "option" the argument key MUST be "option", NOT
   "shipping_option")

4. CRITICAL -- expected_output keys must EXACTLY match the field names
   defined in output_schema above -- do NOT invent new field names,
   do NOT use synonyms (e.g. if output_schema has "certificate_id"
   use "certificate_id", NOT "user_id" or "amount"; if output_schema
   has "status" use "status", NOT "success")

5. expected_output values must be consistent with the DB snapshot
   values -- EXCEPT for creation tools (book_reservation, book_hotel,
   book_car_rental, create_booking) where the new entity ID should
   be a natural-language phrase like "newly generated booking ID"
   since it does not exist yet

6. 3-5 subgoals, each a verifiable assertion about AGENT behavior
   (not DB facts)

7. Subgoals must reference specific IDs, names, or amounts from the
   DB snapshot

JSON Output:
```

A.8 Capability Correctness Is Invariant to Skill Grouping

Proposition. ***(Capability Correctness Is Invariant to Skill Grouping).*** *We define a task x, an agent execution trajectory τ, and the list of ground truth capabilities $C^{\mathrm{GT}}(x)$, with $M(x) > 0$. Let S and S' be two skill libraries that group these ground truth capabilities differently. Let $C_x(s)$ denote the ground truth capabilities assigned to skill s. For each skill library $L \in \{S, S'\}$, these assignments satisfy:*

$$\bigcup_{s \in L} C_x(s) = C^{\mathrm{GT}}(x), \quad \textit{where} \quad C_x(s_1) \cap C_x(s_2) = \emptyset \quad \textit{for distinct } s_1, s_2 \in L. \tag{11}$$

Suppose LLM_judge(c, τ) evaluates each ground truth capability independent of its skill grouping, then we have:

$$\mathrm{CC}_S(x, \tau) = \mathrm{CC}_{S'}(x, \tau) = \mathrm{CC}(x, \tau), \tag{12}$$

where the subscripts indicate the grouping used to compute the capability correctness metric.

Proof. Each grouping counts every ground-truth capability exactly once. Therefore, for either skill library $L \in \{S, S'\}$, we have:

$$\begin{aligned} \mathrm{CC}_L(x, \tau) &= \frac{1}{M(x)} \sum_{s \in L} \sum_{c \in C_x(s)} \text{LLM_judge}(c, \tau) \\ &= \frac{1}{M(x)} \sum_{m=1}^{M(x)} \text{LLM_judge}(c_m^{\mathrm{GT}}(x), \tau) \\ &= \mathrm{CC}(x, \tau). \end{aligned} \tag{13}$$

Since neither the individual judgments nor $M(x)$ changes, both groupings yield the same score. □

A.9 Other Evaluation Metrics

A.9.1 Skill Precision and Recall

These measure the quality and coverage of skill invocation: whether the agent invokes the right skills (precision) and whether it invokes all necessary skills (recall).

$$\text{Skill Precision} = \frac{1}{N} \sum_{i=1}^{N} \frac{|S_{\text{invoked},i} \cap S_{\text{ground truth},i}|}{|S_{\text{invoked},i}|}$$

$$\text{Skill Recall} = \frac{1}{N} \sum_{i=1}^{N} \frac{|S_{\text{invoked},i} \cap S_{\text{ground truth},i}|}{|S_{\text{ground truth},i}|}$$

where N = number of tasks, $S_{\text{invoked},i}$ = set of skills the agent invoked on task i, $S_{\text{ground truth},i}$ = ground-truth skill set for task i. Each element of S is a tuple $(\text{skill_name}, \text{args})$. The intersection $\cap$ currently requires exact match on both the skill name and its input arguments; Precision penalizes spurious invocations, while recall penalizes missed skills. Note that the two metrics only assess whether the agent invoke the skill and does not measure whether the outcome from invoking the skill is successful or not. The skill recall metric may also be known by other names such as skill invocation rate (Li et al., 2026).

A.10 Agent and Dataset

We bring together seven open-source agent domains and convert each into a reusable skill: τ^2-retail and τ^2-airline from τ^2-Bench (Barres et al., 2026); OfficeBench (Wang et al., 2024); ToolSandbox (Lu et al., 2025); and STATE-travel, STATE-customer, and STATE-shopping from STATE-Bench (Microsoft, 2026). We select some domains that intentionally overlap in their skill capabilities, to study skill redundancy and interference. We release 320 test cases using the procedure detailed in Section 3.4, comprising 42 general-domain, 103 single-domain, and 175 cross-domain test cases. We augment CUGA (Shlomov et al., 2026) and Pi (Zechner, 2025) agents with the seven domain skills.

To extend the generalist agent's capability with these domains, we use the `agent-skill-creator` skill (Lisboa, 2026) within Claude Code (Anthropic). For each domain, we instruct Claude Code to inspect the complete source implementation, including its task logic, domain policies, tool definitions and implementations, and use this information to construct the skill. Each skill packages the domain instructions and policies with the executable tools and supporting resources needed to perform its tasks.

We construct a unified environment ϵ by merging common entities from the source domains. For example, users are stored in a shared table, and the product tables from τ^2-retail and STATE-shopping

are combined, τ^2-airline and STATE-travel have flights combined. Most domains share the same users table. This represents a setting where newly added skills operate on data already used by existing skills, rather than introducing an isolated database for each domain. Operations performed through one skill can therefore affect records accessed by another, allowing us to study interactions through shared state that separate domain environments would exclude.

We populate the records in the database using the approach mentioned in Abichandani et al. (2026) approach. An LLM agent inspects the available capabilities to identify entity dependencies, plans workflows covering different operations to create and update related records in the shared database. Each workflow builds on the data already present, allowing records produced by earlier workflows to be reused across domains and ensure high diversity of the unified table. For Officebench and general domain, we additionally make use of Claude Code to generate files such as python script, text files, pdf files etc.

We connect each skill's capabilities to the unified database. All capabilities operate on this environment, with state persisting across different operations within a session. To isolate experimental sessions across test cases, we mount a fresh copy of the same initial environment for every run and execute the experiment inside that instance. This provides consistent starting conditions across runs and prevents state changes from one session from carrying over to another.

A.11 DETAILS OF THE THREE DIFFERENT EXPERIMENTAL SETTINGS

Table 11: Evaluation Settings 1 to 3. Sample old contains test cases for the existing skill library; Sample new contains test cases curated for the newly added skills. × denotes a cross-domain combination.

Test case type	**Sample old**	**Sample new**
Setting 1	**Existing Skill:** τ^2-airline, OfficeBench	**Added Skill:** STATE-travel
General	general	—
Single-domain	τ^2-airline, OfficeBench	STATE-travel
Cross-domain	τ^2-airline × OfficeBench; OfficeBench × general; τ^2-airline × general; τ^2-airline × OfficeBench × general	STATE-travel × OfficeBench; STATE-travel × general
Setting 2	**Existing Skill:** general tools only	**Added Skill:** OfficeBench
General	general	—
Single-domain	—	OfficeBench
Cross-domain	—	OfficeBench × general
Setting 3	**Existing Skill:** τ^2-retail, ToolSandbox	**Added Skill:** STATE-customer, STATE-shopping
General	general	—
Single-domain	τ^2-retail, ToolSandbox	STATE-customer, STATE-shopping
Cross-domain	τ^2-retail × ToolSandbox; τ^2-retail × general; ToolSandbox × general; τ^2-retail × ToolSandbox × general	τ^2-retail × STATE-customer; ToolSandbox × STATE-customer; ToolSandbox × STATE-shopping; STATE-customer × general; STATE-customer × STATE-shopping; STATE-shopping × general; τ^2-retail × ToolSandbox × STATE-customer; τ^2-retail × STATE-customer × general; ToolSandbox × STATE-customer × general; ToolSandbox × STATE-customer × STATE-shopping; ToolSandbox × STATE-shopping × general; STATE-customer × STATE-shopping × general; τ^2-retail × ToolSandbox × STATE-customer × general; ToolSandbox × STATE-customer × STATE-shopping × general

A.12 Additional Experimental Results

Table 12: Union of all old and new test cases spanning general, single-domain and cross-domain types. The CUGA agent uses the model Claude Opus 4.6. Δ columns report the change in metric value of the new agent system from old system. Values that are annotated with blue indicates improvement, while red indicates regression; for latency/tokens lower is better.

		Cap. corr.		Mean Prog. rate		Latency (s)		Tokens (M)	
Setting	Config	Value	Δ_{union}	Value	Δ_{union}	Value	Δ_{union}	Value	Δ_{union}
CUGA S1	Old system	0.795		0.608		632.0		5.01	
	New system	0.801	↑0.007	0.619	↑0.010	536.3	↓95.7	3.99	↓1.02
CUGA S2	Old system	0.657		0.661		405.8		1.84	
	New system	0.832	↑0.174	0.681	↑0.020	265.8	↓140.0	2.31	↑0.48
CUGA S3	Old system	0.468		0.347		551.9		6.12	
	New system	0.636	↑0.168	0.464	↑0.117	693.8	↑141.9	8.65	↑2.53
Pi S1	Old system	0.545		0.447		100.4		0.035	
	New system	0.545	0.000	0.463	↑0.015	94.9	↓5.5	0.035	↓0.001
Pi S2	Old system	0.619		0.584		72.1		0.040	
	New system	0.716	↑0.098	0.616	↑0.032	82.7	↑10.5	0.037	↓0.003
Pi S3	Old system	0.486		0.373		88.2		0.031	
	New system	0.566	↑0.080	0.428	↑0.055	98.8	↑10.6	0.033	↑0.002

Table 13: **General+single-domain** test cases for Settings 1-3. Δ columns report the change in metric value of the new agent system relative to old system. T_{old} and T_{new} denote old and new test cases, respectively. Values that are annotated with blue indicates improvement, while red indicates regression; for latency/tokens lower is better.

(a) Capability correctness, skill precision & skill recall

		Capability correctness				Skill precision			Skill recall		
Setting	Config	T_{old}	Δ_{old}	T_{new}	Δ_{new}	T_{old}	Δ_{old}	T_{new}	T_{old}	Δ_{old}	T_{new}
GPT-5.6-sol											
CUGA S1	Old system	0.016		0.000		0.367		–	0.367		–
	New system	0.020	↑0.004	0.000	0.000	0.363	↓0.004	0.017	0.363	↓0.004	0.017
CUGA S2	Old system	0.100		0.000		1.000		–	1.000		–
	New system	0.040	↓0.060	0.000	0.000	1.000	0.000	0.000	1.000	0.000	0.000
Claude Opus 4.6											
CUGA S1	Old system	0.789		0.735		0.961		–	0.971		–
	New system	0.786	↓0.003	0.825	↑0.090	0.916	↓0.045	0.733	0.926	↓0.045	0.800
CUGA S2	Old system	0.910		0.402		1.000		–	1.000		–
	New system	0.880	↓0.030	0.801	↑0.399	0.800	↓0.200	1.000	0.800	↓0.200	1.000
CUGA S3	Old system	0.844		0.270		0.913		–	0.917		–
	New system	0.832	↓0.012	0.583	↑0.313	0.888	↓0.025	0.568	0.890	↓0.027	0.608
Pi S1	Old system	0.605		0.117		0.526		–	0.526		–
	New system	0.583	↓0.022	0.222	↑0.105	0.498	↓0.028	0.117	0.502	↓0.024	0.167
Pi S2	Old system	0.860		0.320		1.000		–	1.000		–
	New system	0.900	↑0.040	0.574	↑0.254	1.000	0.000	0.080	1.000	0.000	0.080
Pi S3	Old system	0.639		0.400		0.597		–	0.600		–
	New system	0.630	↓0.008	0.520	↑0.120	0.486	↓0.111	0.253	0.503	↓0.097	0.279

(b) Mean progress rate, total latency (s) & total tokens (M)

		Mean progress rate				Total latency (s)				Total tokens (M)			
Setting	Config	T_{old}	Δ_{old}	T_{new}	Δ_{new}	T_{old}	Δ_{old}	T_{new}	Δ_{new}	T_{old}	Δ_{old}	T_{new}	Δ_{new}
GPT-5.6-sol													
CUGA S1	Old system	0.150		0.036		55.3		46.3		0.29		0.23	
	New system	0.159	↑0.009	0.052	↑0.016	58.5	↑3.2	38.7	↓7.6	0.31	↑0.02	0.21	↓0.02
CUGA S2	Old system	0.165		0.150		79.0		115.2		0.36		0.46	
	New system	0.160	↓0.005	0.135	↓0.015	63.3	↓15.7	56.3	↓58.9	0.30	↓0.07	0.31	↓0.16
Claude Opus 4.6													
CUGA S1	Old system	0.603		0.577		453.3		637.6		3.73		5.53	
	New system	0.604	↑0.001	0.691	↑0.114	411.5	↓41.8	575.6	↓62.0	2.70	↓1.03	4.50	↓1.03
CUGA S2	Old system	0.800		0.456		204.3		476.6		1.18		1.98	
	New system	0.765	↓0.035	0.520	↑0.064	155.5	↓48.8	300.9	↓175.6	1.41	↑0.23	2.38	↑0.40
CUGA S3	Old system	0.633		0.183		353.9		587.5		3.24		6.72	
	New system	0.634	↑0.001	0.411	↑0.228	349.6	↓4.2	828.8	↑241.3	3.68	↑0.43	10.24	↑3.51
Pi S1	Old system	0.520		0.114		98.1		62.6		0.035		0.025	
	New system	0.516	↓0.004	0.205	↑0.091	81.9	↓16.2	79.3	↑16.7	0.035	0.000	0.026	↑0.001
Pi S2	Old system	0.820		0.339		29.3		86.8		0.028		0.041	
	New system	0.815	↓0.005	0.414	↑0.075	33.7	↑4.4	117.5	↑30.7	0.028	0.000	0.039	↓0.003
Pi S3	Old system	0.465		0.387		54.0		98.3		0.032		0.027	
	New system	0.426	↓0.039	0.486	↑0.099	59.7	↑5.7	110.8	↑12.5	0.032	↓0.001	0.030	↑0.003

Table 14: **Cross-domain** test cases for Settings 1-3. Δ columns report the change in metric value of the new agent system relative to old system. T_{old} and T_{new} denote old and new test cases, respectively. Values that are annotated with blue indicates improvement, while red indicates regression.

(a) Capability correctness, skill precision & skill recall

		Capability correctness				Skill precision			Skill recall		
Setting	Config	T_{old}	Δ_{old}	T_{new}	Δ_{new}	T_{old}	Δ_{old}	T_{new}	T_{old}	Δ_{old}	T_{new}
GPT-5.6-sol											
CUGA S1	Old system	0.041		0.005		0.000		–	0.000		–
	New system	0.029	↓0.012	0.015	↑0.010	0.000	0.000	0.008	0.000	0.000	0.015
CUGA S2	Old system	–		0.090		–		–	–		–
	New system	–		0.018	↓0.072	–		0.000	–		0.000
Claude Opus 4.6											
CUGA S1	Old system	0.845		0.805		0.908		–	0.947		–
	New system	0.837	↓0.008	0.790	↓0.015	0.588	↓0.320	0.408	0.571	↓0.377	0.408
CUGA S2	Old system	–		0.660		–		–	–		–
	New system	–		0.814	↑0.154	–		1.000	–		1.000
CUGA S3	Old system	0.629		0.278		1.000		–	0.800		–
	New system	0.652	↑0.023	0.481	↑0.203	0.985	↓0.015	0.728	0.731	↓0.069	0.553
Pi S1	Old system	0.671		0.548		0.406		–	0.394		–
	New system	0.654	↓0.017	0.556	↑0.008	0.265	↓0.141	0.062	0.377	↓0.017	0.054
Pi S2	Old system	–		0.676		–		–	–		–
	New system	–		0.675	↓0.001	–		0.000	–		0.000
Pi S3	Old system	0.631		0.401		0.250		–	0.236		–
	New system	0.633	↑0.002	0.531	↑0.131	0.176	↓0.074	0.272	0.162	↓0.074	0.237

(b) Mean progress rate, total latency (s) & total tokens (M)

		Mean progress rate				Total latency (s)				Total tokens (M)			
Setting	Config	T_{old}	Δ_{old}	T_{new}	Δ_{new}	T_{old}	Δ_{old}	T_{new}	Δ_{new}	T_{old}	Δ_{old}	T_{new}	Δ_{new}
GPT-5.6-sol													
CUGA S1	Old system	0.094		0.059		65.4		78.3		0.33		0.38	
	New system	0.094	0.000	0.064	↑0.005	68.3	↑2.9	85.8	↑7.5	0.34	↑0.01	0.43	↑0.05
CUGA S2	Old system	–		0.273		–		124.0		–		0.54	
	New system	–		0.211	↓0.062	–		86.9	↓37.1	–		0.45	↓0.09
Claude Opus 4.6													
CUGA S1	Old system	0.585		0.687		991.8		830.1		6.94		6.84	
	New system	0.541	↓0.044	0.709	↑0.022	557.0	↓434.8	943.5	↑113.4	4.10	↓2.84	8.27	↑1.43
CUGA S2	Old system	–		0.727		–		536.6		–		2.35	
	New system	–		0.758	↑0.031	–		341.1	↓195.5	–		3.15	↑0.80
CUGA S3	Old system	0.518		0.206		574.8		724.0		6.33		8.54	
	New system	0.499	↓0.019	0.334	↑0.128	636.6	↑61.8	913.2	↑189.2	7.32	↑0.99	12.45	↑3.91
Pi S1	Old system	0.503		0.408		141.1		90.6		0.039		0.042	
	New system	0.494	↓0.009	0.455	↑0.048	148.9	↑7.7	87.8	↓2.8	0.038	↓0.001	0.039	↓0.003
Pi S2	Old system	–		0.593		–		100.2		–		0.051	
	New system	–		0.620	↑0.027	–		96.8	↓3.5	–		0.044	↓0.006
Pi S3	Old system	0.474		0.292		60.9		108.4		0.029		0.032	
	New system	0.488	↑0.014	0.381	↑0.089	71.2	↑10.3	120.0	↑11.6	0.032	↑0.003	0.034	↑0.002

Table 15: **General+single-domain+cross-domain** test cases for Settings 1-3. Δ columns report the change in metric value of the new agent system relative to old system. T_{old} and T_{new} denote old and new test cases, respectively. Values that are annotated with blue indicates improvement, while red indicates regression; for latency/tokens lower is better.

(a) Capability correctness, skill precision & skill recall

		Capability correctness				Skill precision			Skill recall		
Setting	Config	T_{old}	Δ	T_{new}	Δ	T_{old}	Δ	T_{new}	T_{old}	Δ	T_{new}
GPT-5.6-sol											
CUGA S1	Old system	0.022		0.003		0.273		–	0.273		–
	New system	0.023	↑0.001	0.008	↑0.005	0.270	↓0.003	0.012	0.270	↓0.003	0.016
CUGA S2	Old system	0.100		0.045		1.000		–	1.000		–
	New system	0.040	↓0.060	0.009	↓0.036	1.000	0.000	0.000	1.000	0.000	0.000
Claude Opus 4.6											
CUGA S1	Old system	0.804		0.771		0.948		–	0.965		–
	New system	0.799	↓0.005	0.807	↑0.035	0.832	↓0.116	0.564	0.835	↓0.130	0.596
CUGA S2	Old system	0.910		0.531		1.000		–	1.000		–
	New system	0.880	↓0.030	0.807	↑0.276	0.800	↓0.200	1.000	0.800	↓0.200	1.000
CUGA S3	Old system	0.805		0.273		0.929		–	0.896		–
	New system	0.800	↓0.006	0.541	↑0.268	0.906	↓0.023	0.634	0.862	↓0.034	0.585
Pi S1	Old system	0.622		0.341		0.495		–	0.492		–
	New system	0.601	↓0.021	0.396	↑0.054	0.438	↓0.058	0.088	0.470	↓0.023	0.108
Pi S2	Old system	0.860		0.498		1.000		–	1.000		–
	New system	0.900	↑0.040	0.625	↑0.127	1.000	0.000	0.040	1.000	0.000	0.040
Pi S3	Old system	0.634		0.400		0.397		–	0.390		–
	New system	0.632	↓0.002	0.528	↑0.128	0.308	↓0.090	0.267	0.307	↓0.084	0.248

(b) Mean progress rate, total latency (s) & total tokens (M)

		Mean progress rate				Total latency (s)				Total tokens (M)			
Setting	Config	T_{old}	Δ	T_{new}	Δ	T_{old}	Δ	T_{new}	Δ	T_{old}	Δ	T_{new}	Δ
GPT-5.6-sol													
CUGA S1	Old system	0.136		0.048		57.9		62.9		0.30		0.31	
	New system	0.142	↑0.006	0.058	↑0.010	61.0	↑3.1	63.2	↑0.3	0.31	↑0.02	0.32	↑0.01
CUGA S2	Old system	0.165		0.212		79.0		119.6		0.36		0.50	
	New system	0.160	↓0.005	0.173	↓0.039	63.3	↓15.7	71.6	↓48.0	0.30	↓0.07	0.38	↓0.13
Claude Opus 4.6													
CUGA S1	Old system	0.599		0.634		592.0		737.7		4.56		6.21	
	New system	0.588	↓0.011	0.700	↑0.066	448.9	↓143.0	766.9	↑29.2	3.06	↓1.50	6.46	↑0.25
CUGA S2	Old system	0.800		0.592		204.3		506.6		1.18		2.17	
	New system	0.765	↓0.035	0.639	↑0.048	155.5	↓48.8	321.0	↓185.6	1.41	↑0.23	2.77	↑0.60
CUGA S3	Old system	0.612		0.193		393.2		643.8		3.79		7.47	
	New system	0.610	↓0.003	0.379	↑0.187	400.7	↑7.5	863.6	↑219.8	4.32	↑0.53	11.15	↑3.68
Pi S1	Old system	0.516		0.267		109.2		77.2		0.036		0.034	
	New system	0.511	↓0.005	0.335	↑0.068	99.1	↓10.0	83.7	↑6.5	0.035	0.000	0.033	↓0.001
Pi S2	Old system	0.820		0.466		29.3		93.5		0.028		0.046	
	New system	0.815	↓0.005	0.517	↑0.051	33.7	↑4.4	107.1	↑13.6	0.028	0.000	0.041	↓0.004
Pi S3	Old system	0.470		0.317		58.0		105.8		0.031		0.031	
	New system	0.462	↓0.008	0.408	↑0.092	66.3	↑8.4	117.6	↑11.8	0.032	↑0.001	0.033	↑0.002

A.13 TEST CASES EXAMPLE

A.13.1 GENERAL DOMAIN

General-domain test case `tc_00d92b1d`

Task. James Okafor (E002) wants to create a short workspace report from the Q3 sales file listing his won deals and save it as a new file. In `data/sales_q3.csv`, his won deals include D001 for $87,500, D003 for $43,000, D006 for $120,000, and D009 for $95,000.

Expected skill: `None`

Subgoal checks:
1. The agent reads `data/sales_q3.csv` before creating the report.
2. The agent identifies James Okafor's won deals as D001, D003, D006, and D009.
3. The agent includes the real amounts $87,500, $43,000, $120,000, and $95,000 in the written summary.
4. The agent writes a file containing *James Okafor (E002)* and references `data/sales_q3.csv` as the source.

Expected capabilities:

Capability	Description	Expected output
`read_file`	`Read file contents from the workspace, optionally filtering by line range or regular-expression pattern.`	`{"content": "deal_id, rep_name,region,product, amount,close_date, status\n D001,James Okafor,North,Meridian Insights Pro,87500, 2026-07-05,won\n D003, James Okafor,East, Meridian DataBridge, 43000,2026-07-12,won\n D006,James Okafor,South, Meridian Insights Pro, 120000,2026-07-22,won\n D009,James Okafor,North, Meridian Insights Pro, 95000,2026-08-05,won"}`
`write_file`	`Write content to a file, creating it if it does not exist or overwriting it if it does.`	`{"success": true}`

A.13.2 SINGLE DOMAIN

Single-domain test case `tc_4a78f21a`

Task. Elena Marquez (`user_59b22691`) wants to review her confirmed booking BK-B70889 for flight FL658038 from JFK to LHR, check that the flight is still on time, review the change policy, search for available JFK to LHR options, and then update the booking to `premium_economy` on the same flight. She originally paid $1,180 for BK-B70889, and also has a return booking BK-CE5BA9 on FLF49AF9; the agent must modify only BK-B70889.

Expected skill: `state-travel`

Subgoal checks:
1. The agent confirms that BK-B70889 belongs to Elena Marquez (`user_59b22691`) and is a confirmed booking on flight FL658038 with business cabin and aisle seat, priced at $1,180.

2. The agent reports that flight FL658038 is `scheduled` with 0 delay minutes before making any recommendation about changing BK-B70889.
3. The agent cites the change policy for topic `change` and explains that same-flight edits can be made by passing only the changed fields for BK-B70889.
4. The agent identifies FL658038 on 2026-10-15 from JFK to LHR in search results and mentions the cabin prices $520 for economy, $780 for premium_economy, and $1,180 for business.
5. The agent updates only BK-B70889 to `premium_economy` and states the resulting values: `booking_id` BK-B70889, status confirmed, `fare_difference` -400, and `new_total` 780, without altering BK-CE5BA9.

Expected capabilities:

Capability	Description	Expected output
update_booking	Update an existing booking. Only pass fields to change for same-flight edits. For flight changes, pass confirm=false first to preview fees, then confirm=true to execute.	{"booking_id": "BK-B70889", "status": "confirmed", "change_fee": 0, "fare_difference": -400, "new_total": 780}
get_booking	Retrieve the details of an existing booking by its booking ID.	{"booking_id": "BK-B70889", "user_id": "user_59b22691", "flight_id": "FL658038", "cabin_class": "business", "seat_type": "aisle", "status": "confirmed", "price_paid": 1180}
get_flight_status	Check the current status of a flight (scheduled, delayed, or cancelled).	{"flight_id": "FL658038", "status": "scheduled", "delay_minutes": 0}
get_policies	Look up travel policies for a specific topic. Topics include cancel, change, baggage, delay_compensation, loyalty, upgrade, hotel_cancel, and car_rental_cancel.	{"topic": "change", "rules": "Changes are allowed for confirmed bookings; same-flight edits only require passing changed fields, while flight changes require confirm=false first to preview fees, then confirm=true to execute."}
search_flights	Search for available flights. Returns a list of matching flights with pricing for all cabin classes. Each flight includes a cabin_prices map.	{"flights": [{"flight_id": "FL658038", "origin": "JFK", "destination": "LHR", "departure_time": "2026-10-15T19:00:00", "arrival_time": "2026-10-16T07:00:00", "status": "scheduled", "delay_minutes": 0, "cabin_prices": {"economy": 520, "premium_economy": 780, "business": 1180}}]}

A.13.3 CROSS DOMAIN

Cross-domain test case `tc_3312976f`

Task. Elena Marquez (`user_59b22691`) wants to review her existing travel spend from `data/travel_expenses.xlsx`, find a new flight option, check the baggage policy, book that flight with her saved card `cc_b3676891`, and then look for a hotel in San Francisco. Her spreadsheet already shows confirmed flight bookings BK-B70889 for $1,180 and BK-CE5BA9 for $540, and the new planning should stay consistent with her current travel records.

Expected skills: `OfficeBench`, `state-travel`

Subgoal checks:

1. The agent reads `data/travel_expenses.xlsx` and identifies that `user_59b22691` has confirmed bookings BK-B70889 for $1,180 and BK-CE5BA9 for $540.
2. The agent searches JFK to LHR for 2026-10-15 and surfaces flight FL658038 with cabin prices economy $520, premium_economy $780, and business $1,180.
3. The agent checks the baggage policy by calling `get_policies` with topic `baggage` before making the booking.
4. The agent books FL658038 for Elena Marquez using `user_id` `user_59b22691` and payment method `cc_b3676891`, and reports a confirmed booking with `price_paid` = 520.
5. The agent searches San Francisco hotels for 2026-09-03 to 2026-09-05 and presents HTL-099A91 Bayview Business Hotel at `total_price` = 378.

Expected capabilities:

Capability	Description	Expected output
read	Read all cell values from an Excel file, returning (row, col): value pairs.	{"cells": {"Travel:R1C1": "Booking ID", "Travel:R1C2": "User", "Travel:R1C3": "Type", "Travel:R1C4": "Route/Location", "Travel:R1C5": "Dates", "Travel:R1C6": "Status", "Travel:R1C7": "Amount ... "}}
search_flights	Search for available flights. Returns a list of matching flights with pricing for all cabin classes. Each flight includes a cabin_prices map.	{"flights": [{"flight_id": "FL658038", "airline_code": "", "origin": "JFK", "destination": "LHR", "departure_time": "2026-10-15T19:00:00" ... }}]}
get_policies	Look up travel policies for a specific topic. Topics include cancel, change, baggage, delay_compensation, loyalty, upgrade, hotel_cancel, and car_rental_cancel.	{"topic": "baggage", "rules": ["Baggage policy information for baggage."]}
create_booking	Book a flight for a customer. Requires flight_id, user_id, ocabin_class, seat_type, ... , add_insurance, and payment_method.	{"booking_id": "*newly generated booking ID*", "status": "confirmed", "price_paid": 520}
search_hotels	Search for available hotels in a city. Filter by dates, room type, and price.	{"hotels": [{"hotel_id": "HTL-099A91", "hotel_name": "Bayview Business Hotel", "city": "San Francisco", "check_in": "", "check_out": "", "room_type": "Standard Queen", "nightly_rate": 189, "total_price": 378}]}

A.14 HUMAN VALIDATION OF CAPABILITY JUDGMENTS

We randomly sample 40 trial traces covering 118 capabilities and compare human annotations with the LLM-as-a-judge labels. As shown in Table 16, human annotators and the LLM-as-a-judge assign the same label to 111 of the 118 capabilities (94.1%). They assign different labels in six cases (5.1%), while one case (0.8%) is considered ambiguous by human annotators. Overall, the high agreement rate indicates that the LLM-as-a-judge labels are well aligned with human judgments.

Table 16: Human validation of capability judgments. Entries report counts (percentages) over all 118 annotated capabilities.

Total capabilities	Agree	Disagree	Ambiguous
118	111 (94.1%)	6 (5.1%)	1 (0.8%)

All six disagreements arise at the boundary between functionally equivalent invocation (F) and no match (N). In three cases, the LLM-as-a-judge credited failed or insufficient executions as matches, whereas human annotators did not. In the other three cases, the LLM-as-a-judge did not credit valid alternative implementations that human annotators considered functionally equivalent. The single ambiguous case concerns `get_policy`: the agent eventually retrieved the policy, but only after performing several other tool calls, whereas the policy should have been read first. Because the agent used its own code to retrieve the policy but violated the required execution order, either F or N can reasonably apply.

A.15 EXPERIMENT RESULT PLOTS

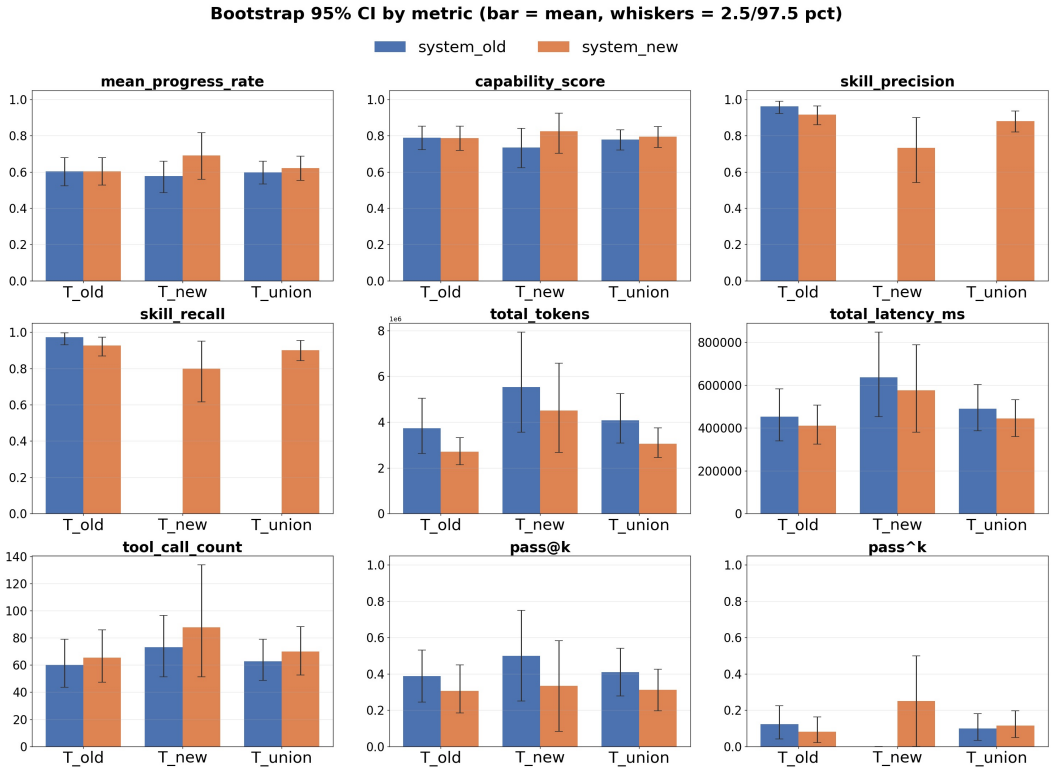


Figure 4: Bootstrap 95% confidence intervals (bar = mean, whiskers = 2.5/97.5 pct) for capability score, progress rate, skill precision/recall, and efficiency metrics on **general and single-domain** tasks for the **CUGA Opus 4.6** agent (Setting 1).

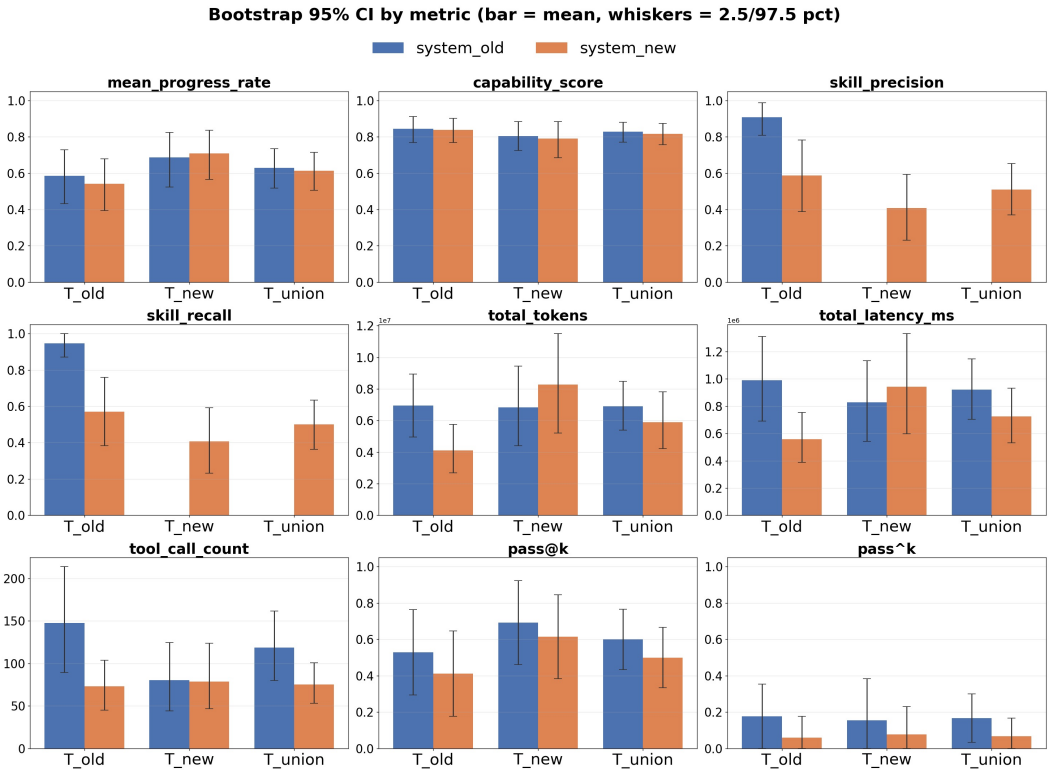


Figure 5: Bootstrap 95% confidence intervals for capability score, progress rate, skill precision/recall, and efficiency metrics on **cross-domain** tasks for the **CUGA Opus 4.6** agent (Setting 1).

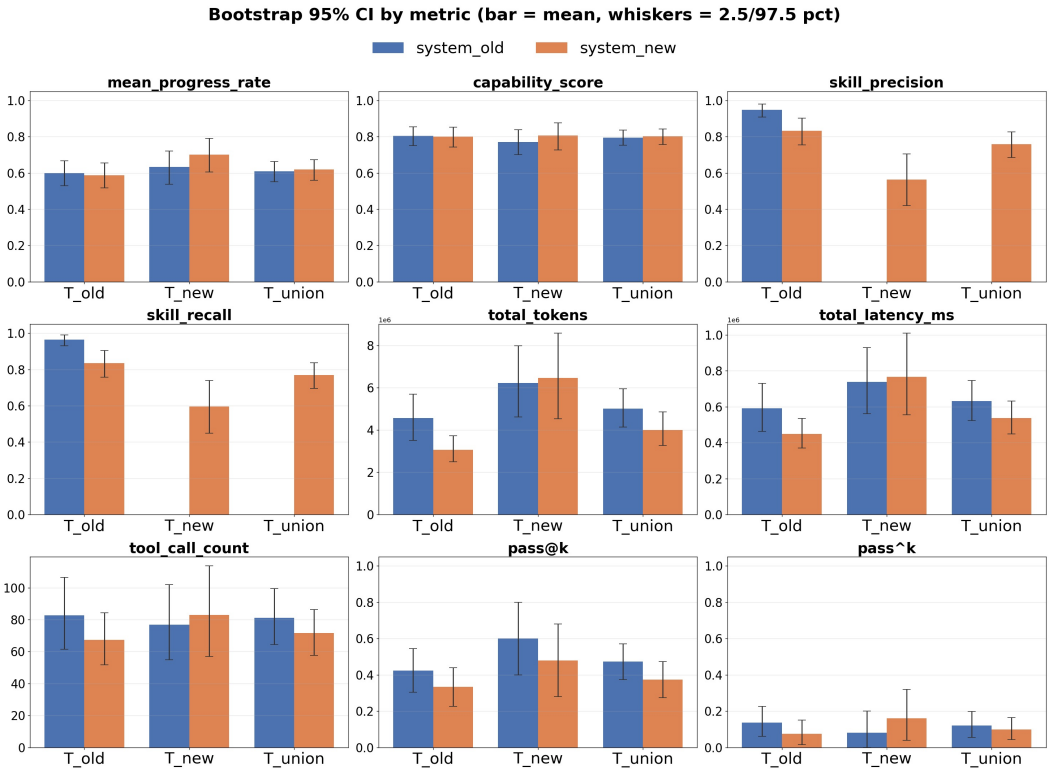


Figure 6: Bootstrap 95% confidence intervals for capability score, progress rate, skill precision/recall, and efficiency metrics on **general, single, and cross-domain** tasks for the **CUGA Opus 4.6** agent (Setting 1).

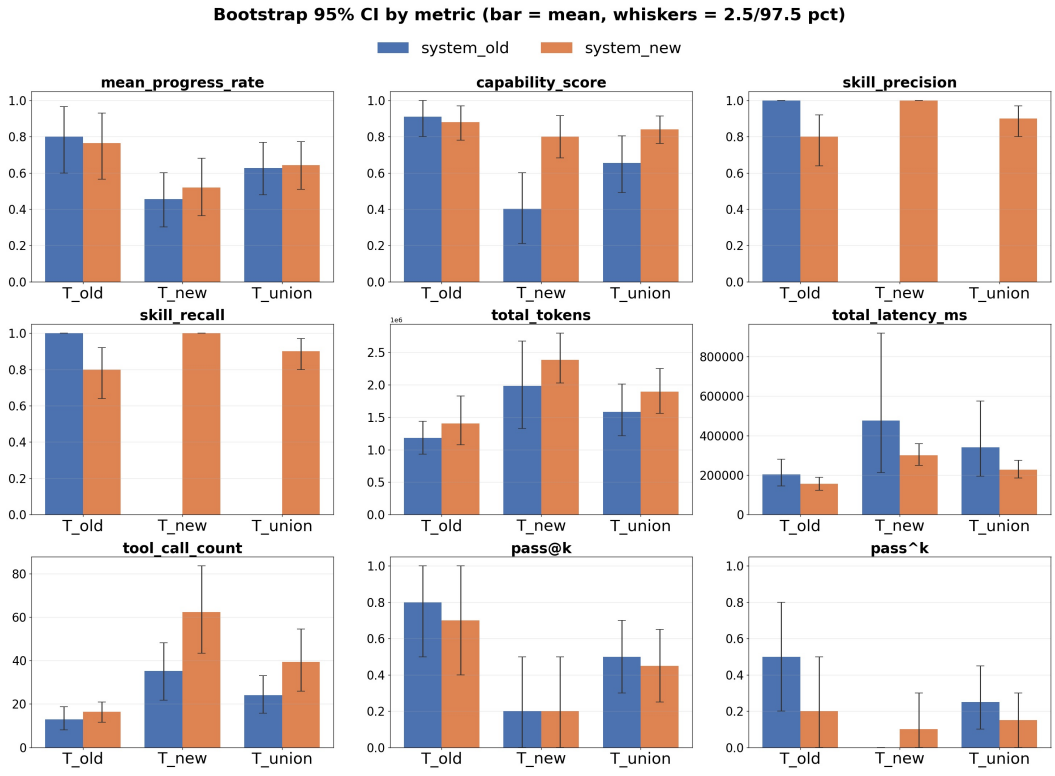


Figure 7: Bootstrap 95% confidence intervals for capability score, progress rate, skill precision/recall, and efficiency metrics on **general and single-domain** tasks for the **CUGA Opus 4.6** agent (Setting 2).

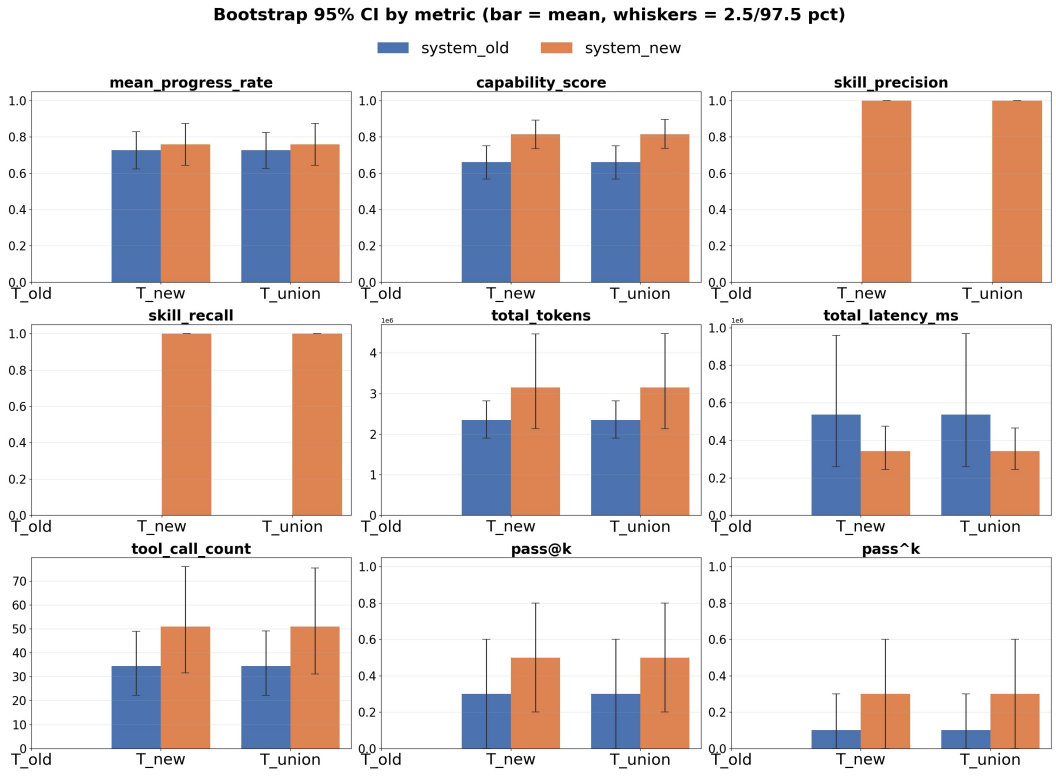


Figure 8: Bootstrap 95% confidence intervals for capability score, progress rate, skill precision/recall, and efficiency metrics on **cross-domain** tasks for the **CUGA Opus 4.6** agent (Setting 2).

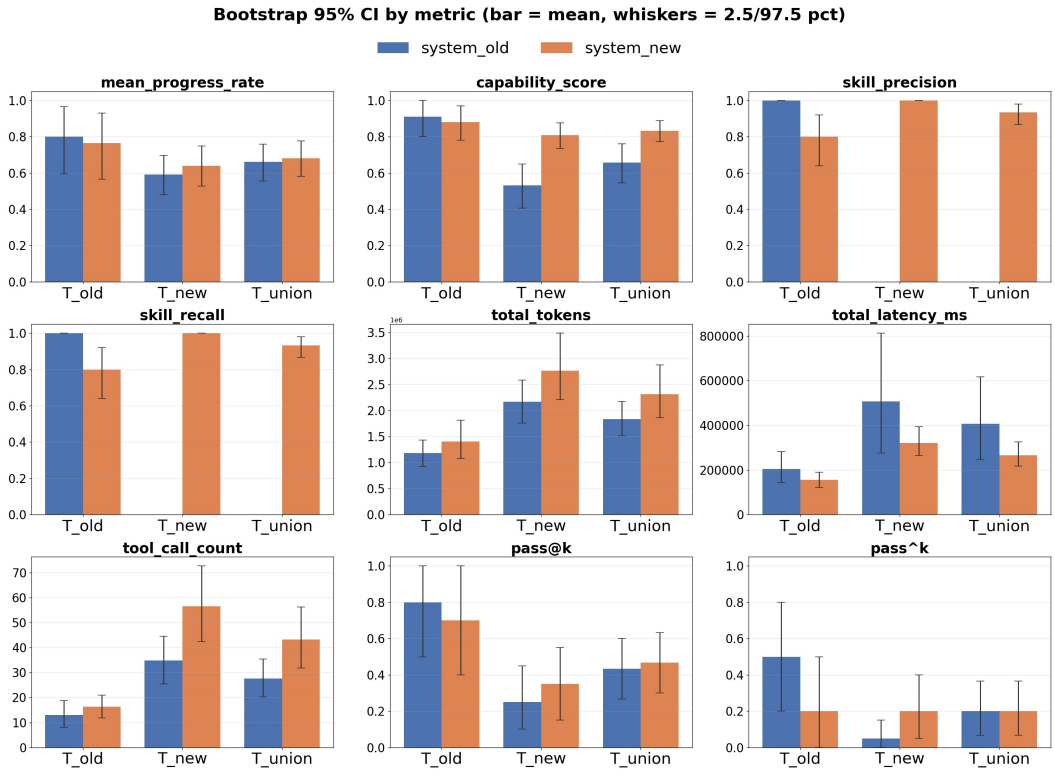


Figure 9: Bootstrap 95% confidence intervals for capability score, progress rate, skill precision/recall, and efficiency metrics on **general, single, and cross-domain** tasks for the **CUGA Opus 4.6** agent (Setting 2).

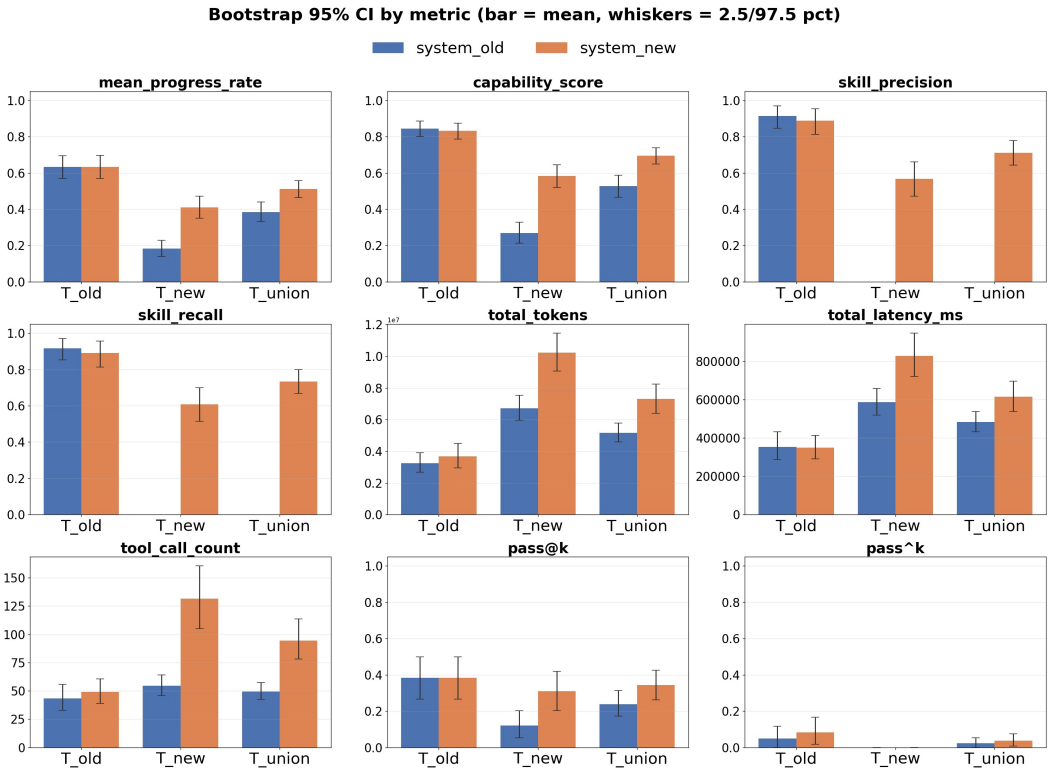


Figure 10: Bootstrap 95% confidence intervals for capability score, progress rate, skill precision/recall, and efficiency metrics on **general and single-domain** tasks for the **CUGA Opus 4.6** agent (Setting 3).

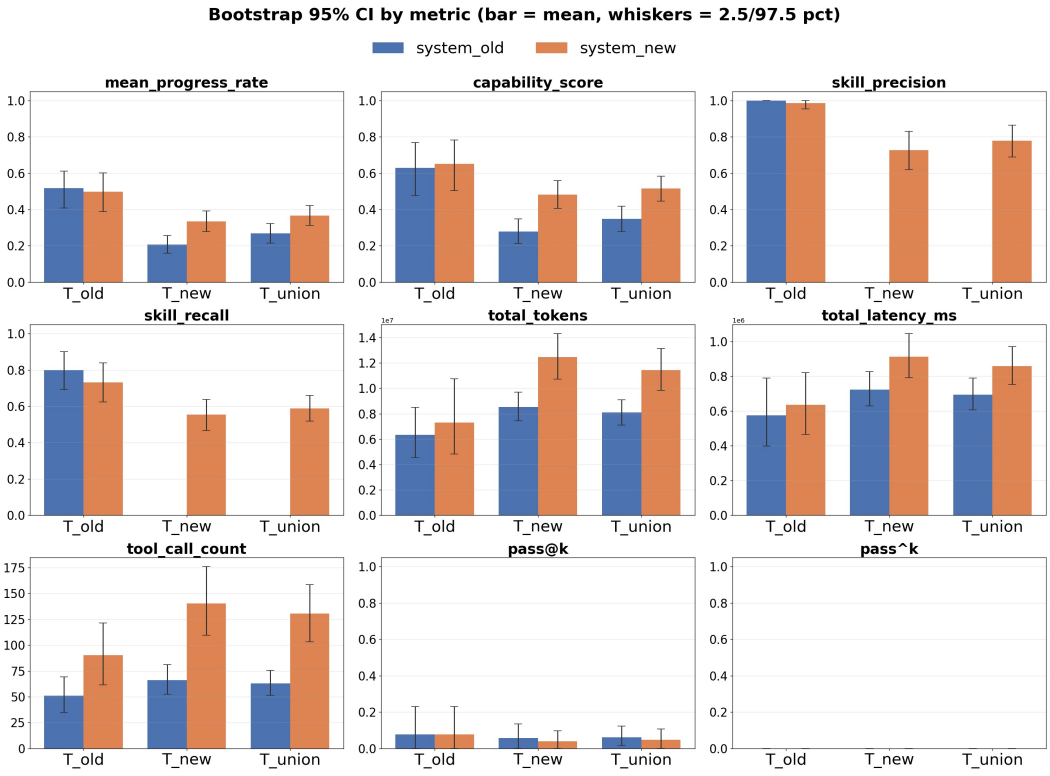


Figure 11: Bootstrap 95% confidence intervals for capability score, progress rate, skill precision/recall, and efficiency metrics on **cross-domain** tasks for the **CUGA Opus 4.6** agent (Setting 3).

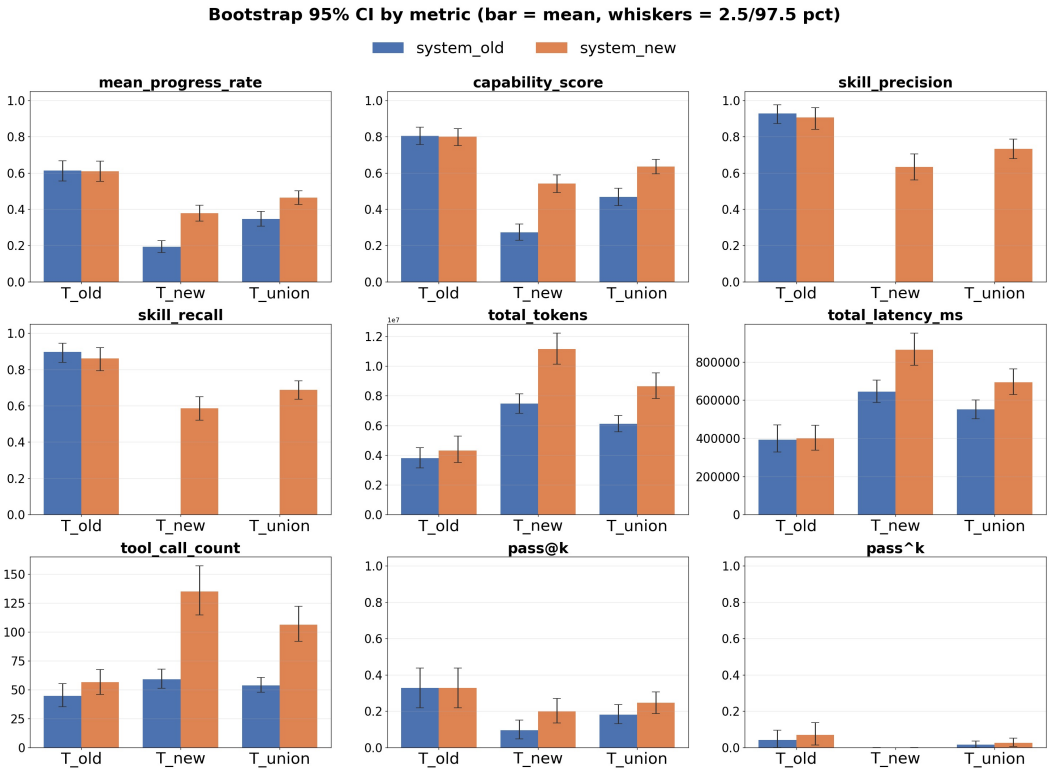


Figure 12: Bootstrap 95% confidence intervals for capability score, progress rate, skill precision/recall, and efficiency metrics on **general, single, and cross-domain** tasks for the **CUGA Opus 4.6** agent (Setting 3).

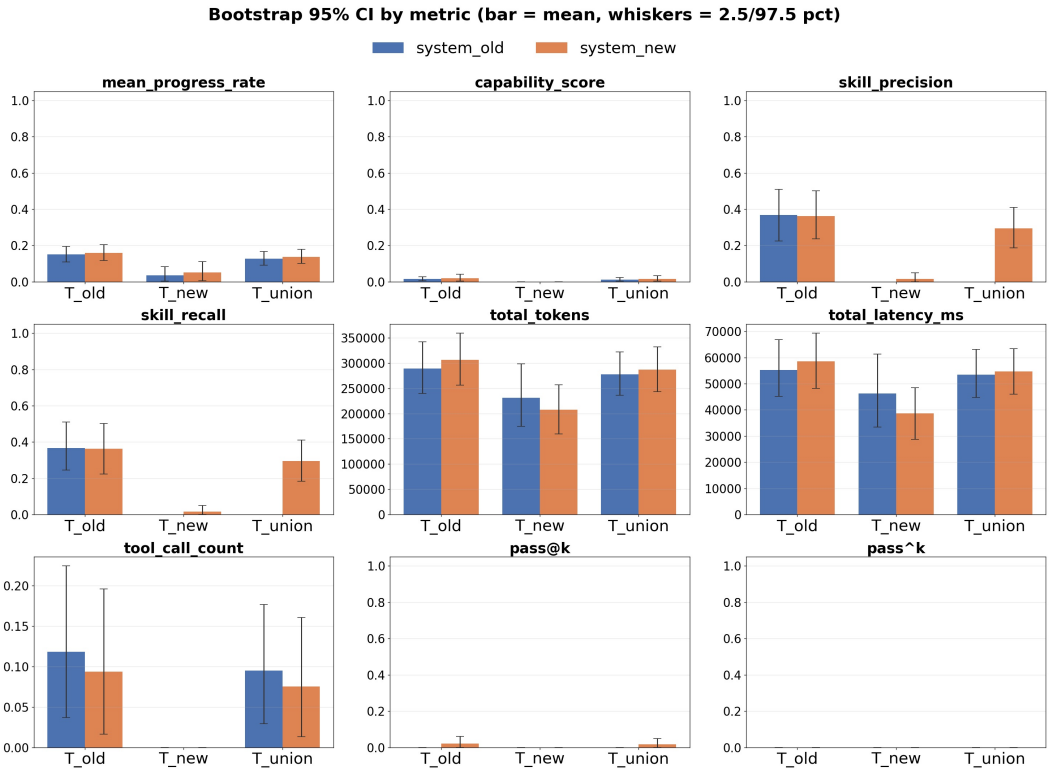


Figure 13: Bootstrap 95% confidence intervals for capability score, progress rate, skill precision/recall, and efficiency metrics on **general and single-domain** tasks for the **CUGA GPT-5.6-Sol** agent (Setting 1).

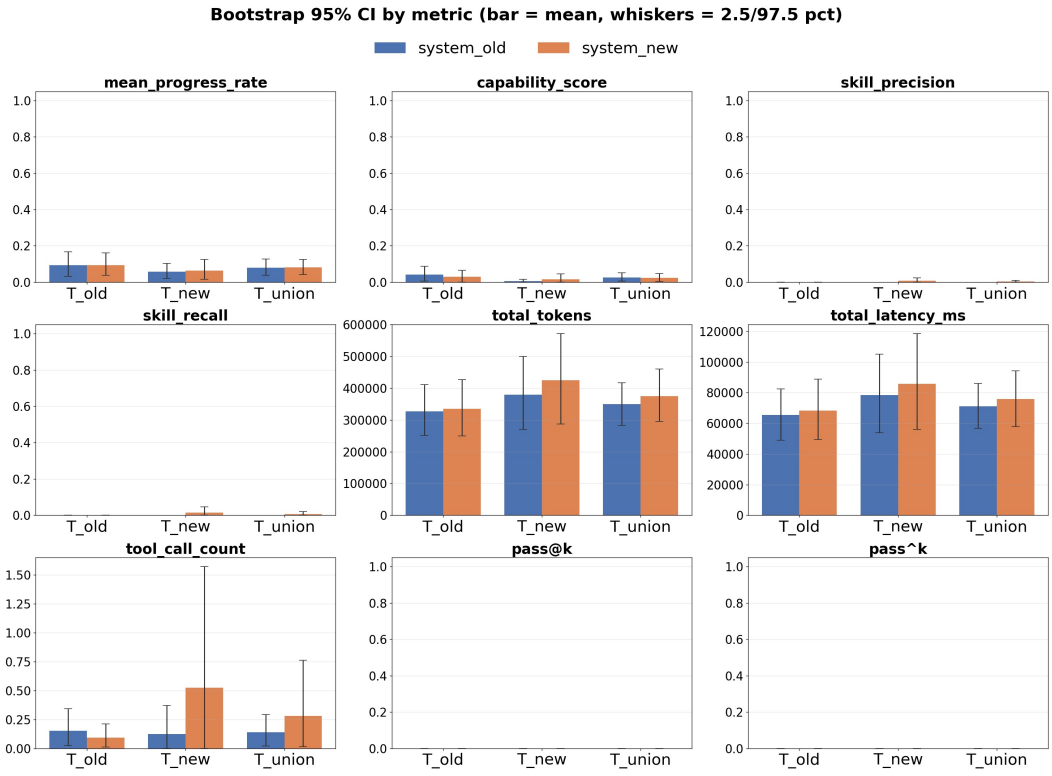


Figure 14: Bootstrap 95% confidence intervals for capability score, progress rate, skill precision/recall, and efficiency metrics on **cross-domain** tasks for the **CUGA GPT-5.6-Sol** agent (Setting 1).

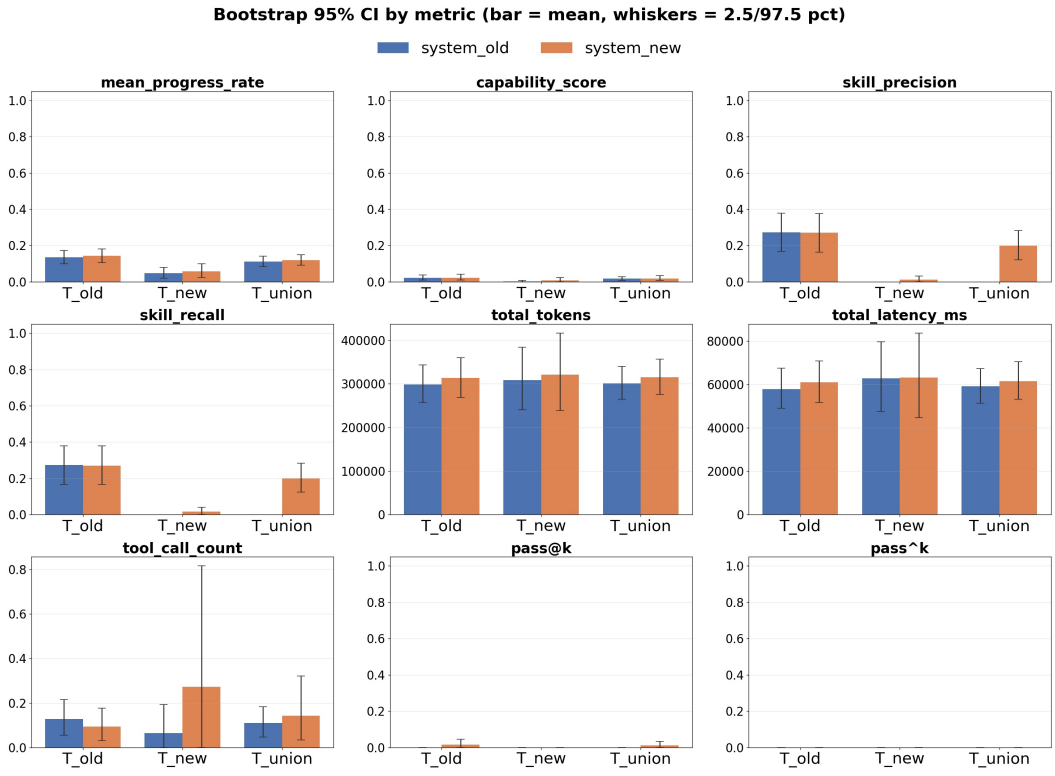


Figure 15: Bootstrap 95% confidence intervals for capability score, progress rate, skill precision/recall, and efficiency metrics on **general, single, and cross-domain** tasks for the **CUGA GPT-5.6-Sol** agent (Setting 1).

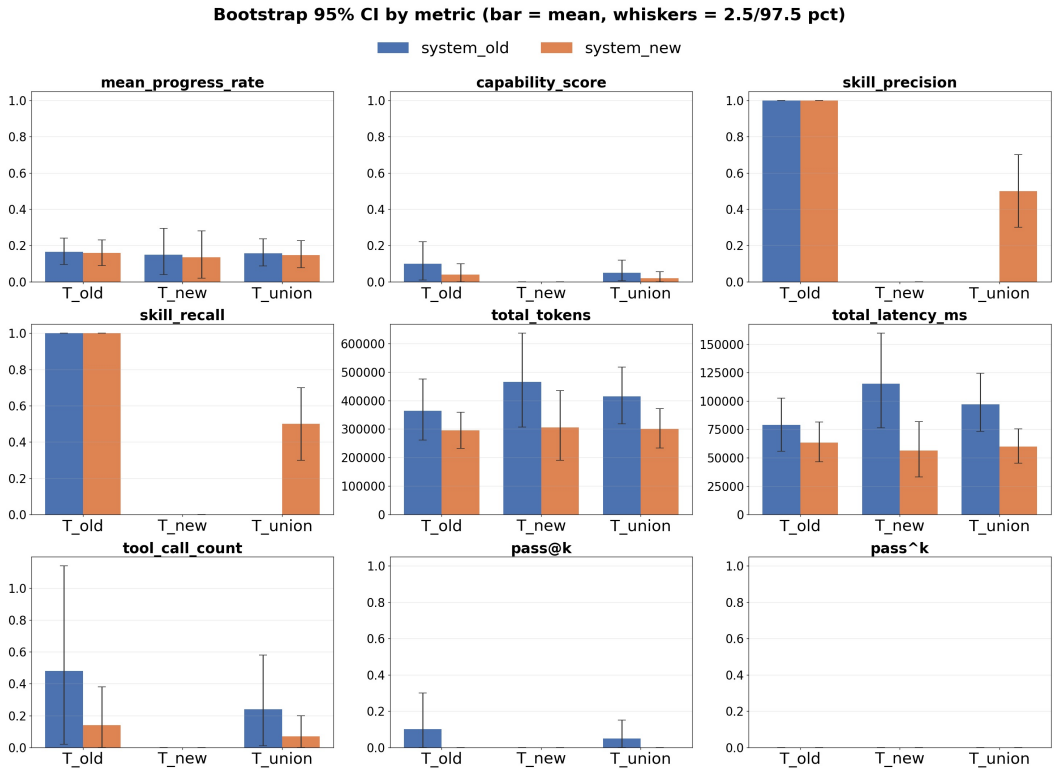


Figure 16: Bootstrap 95% confidence intervals for capability score, progress rate, skill precision/recall, and efficiency metrics on **general and single-domain** tasks for the **CUGA GPT-5.6-Sol** agent (Setting 2).

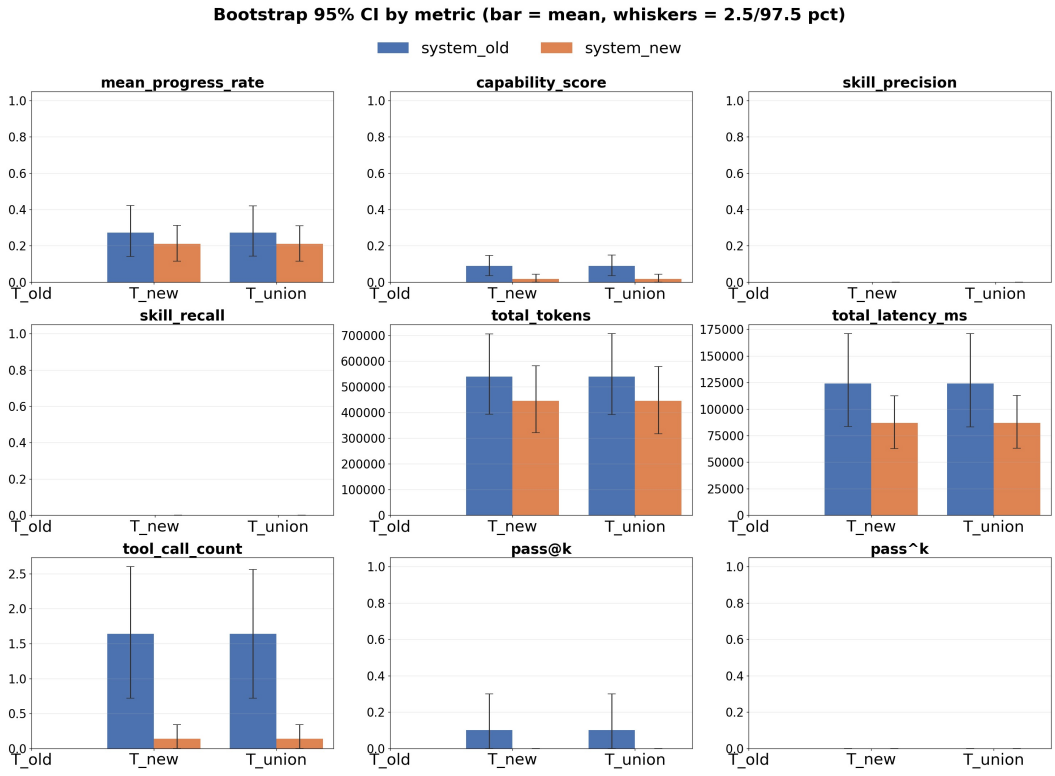


Figure 17: Bootstrap 95% confidence intervals for capability score, progress rate, skill precision/recall, and efficiency metrics on **cross-domain** tasks for the **CUGA GPT-5.6-Sol** agent (Setting 2).

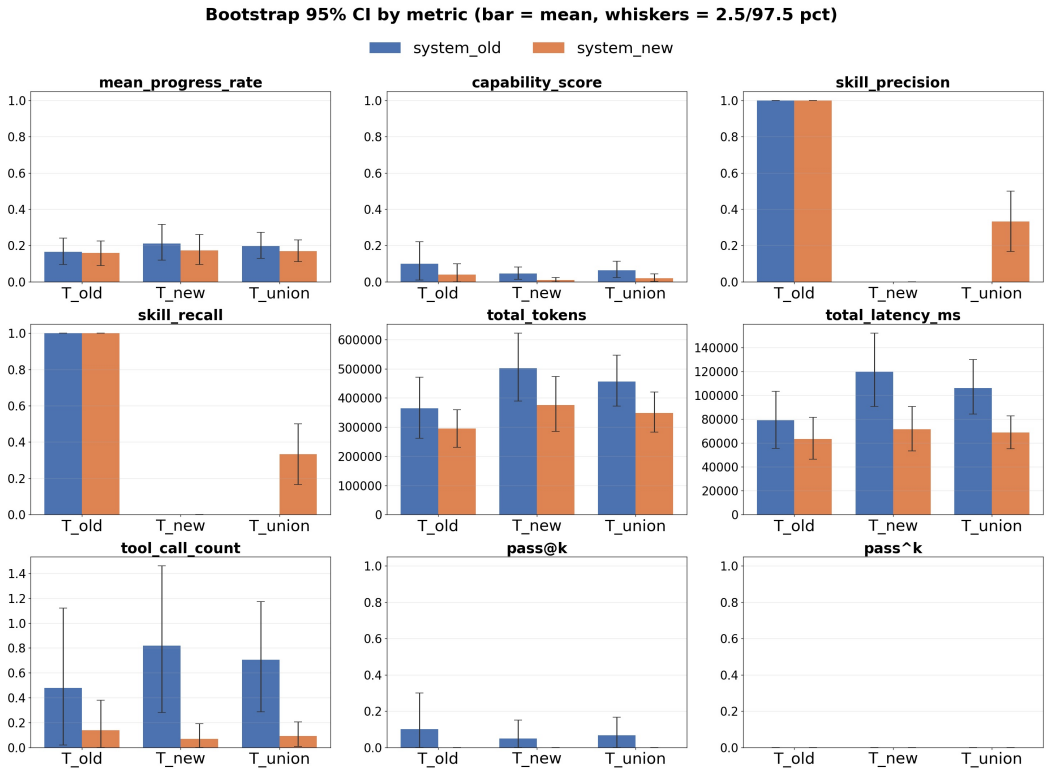


Figure 18: Bootstrap 95% confidence intervals for capability score, progress rate, skill precision/recall, and efficiency metrics on **general, single, and cross-domain** tasks for the **CUGA GPT-5.6-Sol** agent (Setting 2).

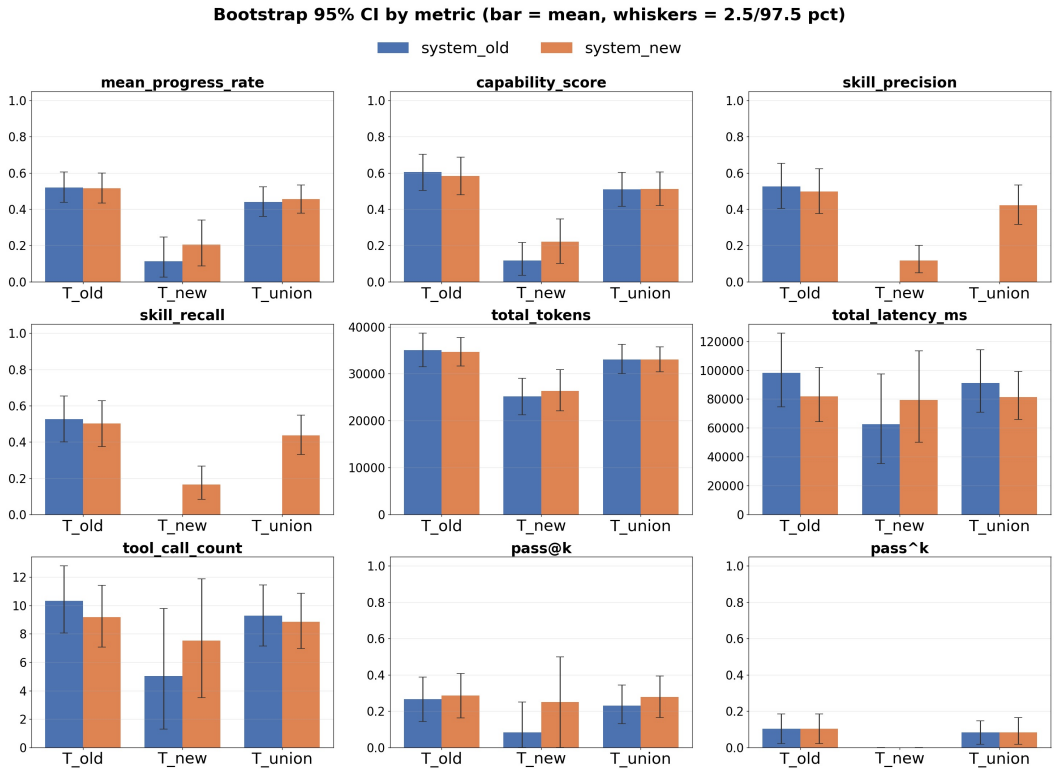


Figure 19: Bootstrap 95% confidence intervals for capability score, progress rate, skill precision/recall, and efficiency metrics on **general and single-domain** tasks for the **PI Opus 4.6** agent (Setting 1).

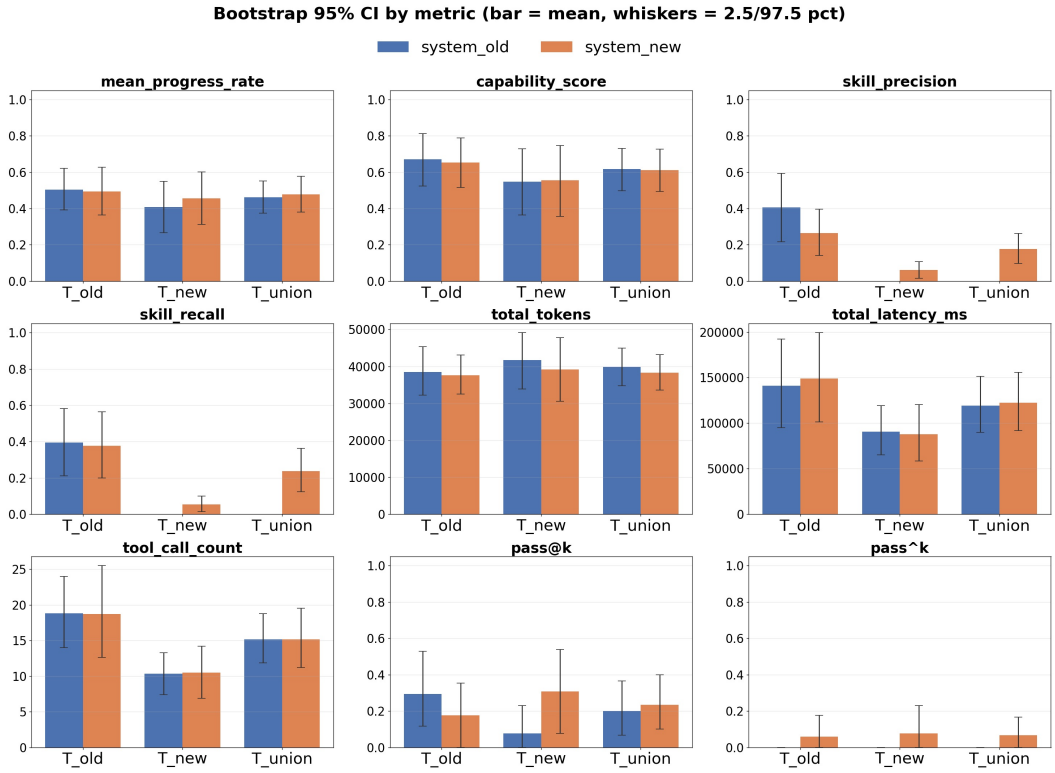


Figure 20: Bootstrap 95% confidence intervals for capability score, progress rate, skill precision/recall, and efficiency metrics on **cross-domain** tasks for the **PI Opus 4.6** agent (Setting 1).

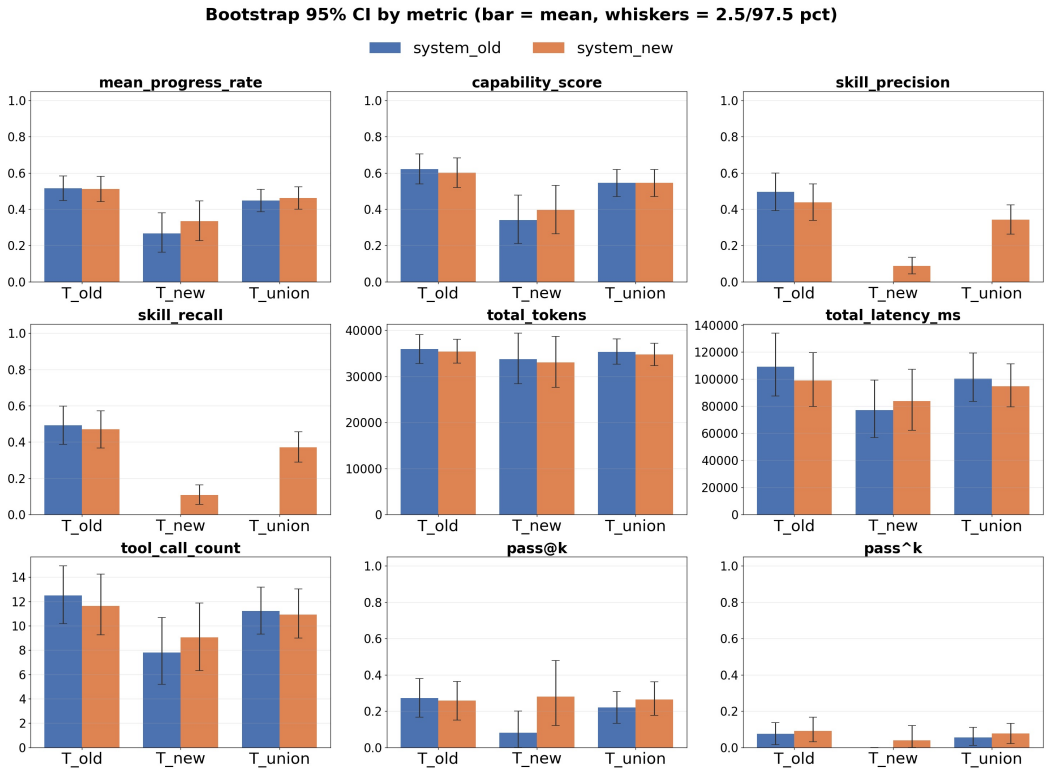


Figure 21: Bootstrap 95% confidence intervals for capability score, progress rate, skill precision/recall, and efficiency metrics on **general, single, and cross-domain** tasks for the **PI Opus 4.6** agent (Setting 1).

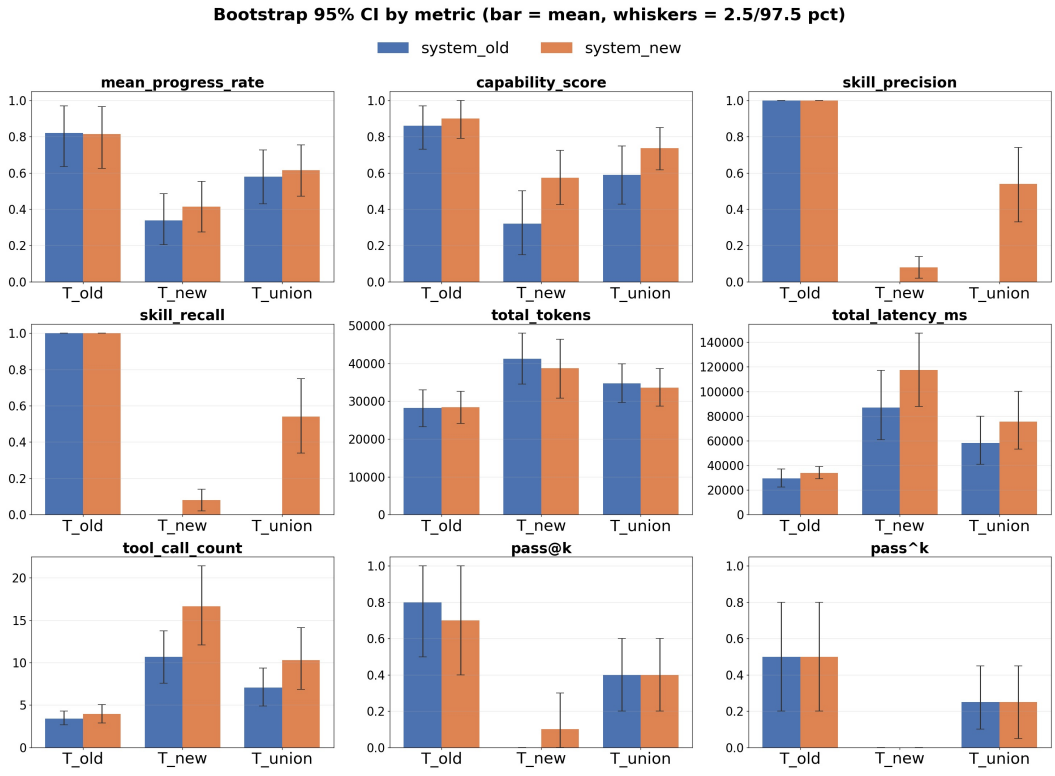


Figure 22: Bootstrap 95% confidence intervals for capability score, progress rate, skill precision/recall, and efficiency metrics on **general and single-domain** tasks for the **PI Opus 4.6** agent (Setting 2).

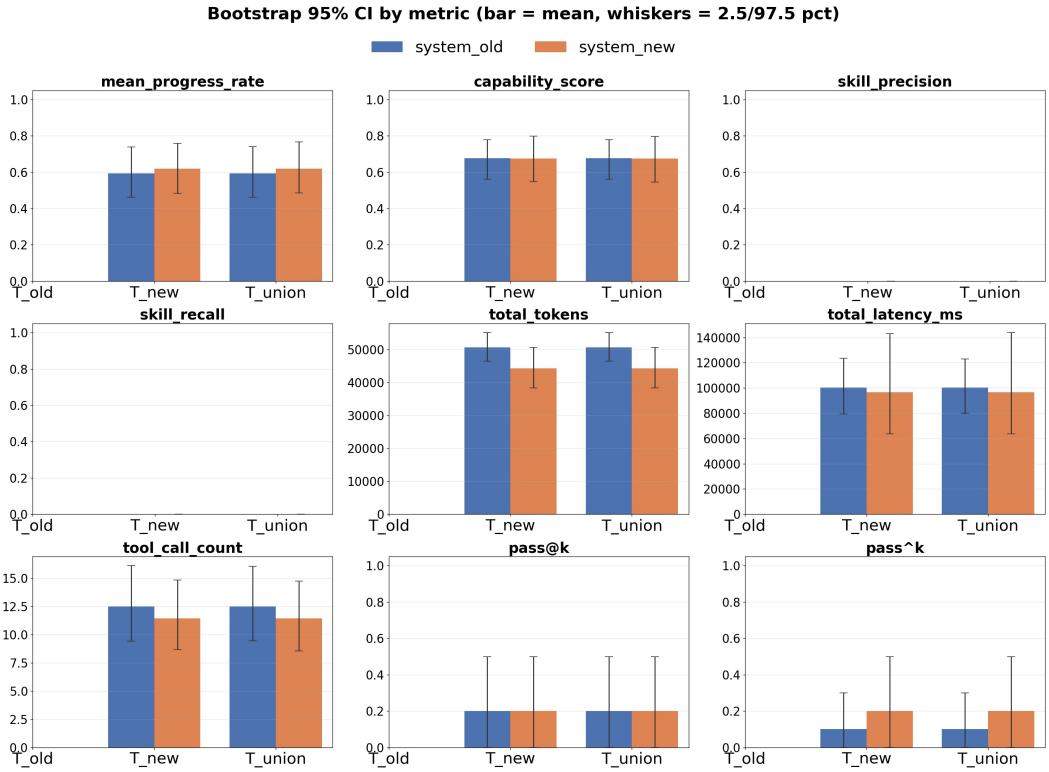


Figure 23: Bootstrap 95% confidence intervals for capability score, progress rate, skill precision/recall, and efficiency metrics on **cross-domain** tasks for the **PI Opus 4.6** agent (Setting 2).

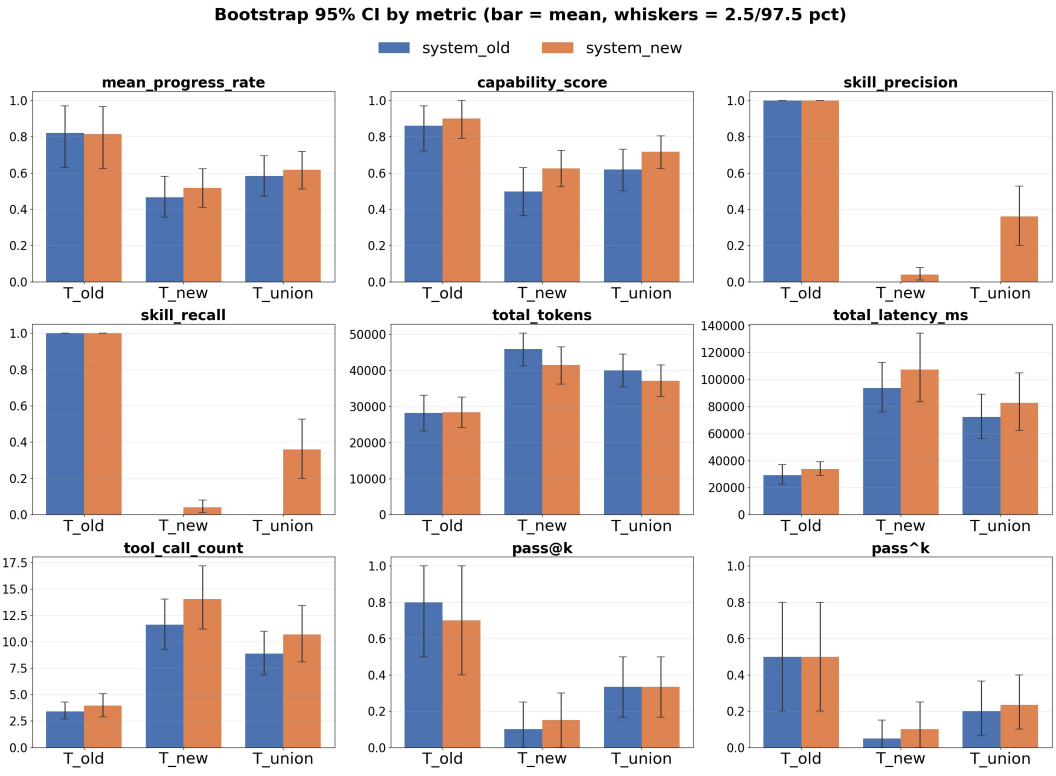


Figure 24: Bootstrap 95% confidence intervals for capability score, progress rate, skill precision/recall, and efficiency metrics on **general, single, and cross-domain** tasks for the **PI Opus 4.6** agent (Setting 2).

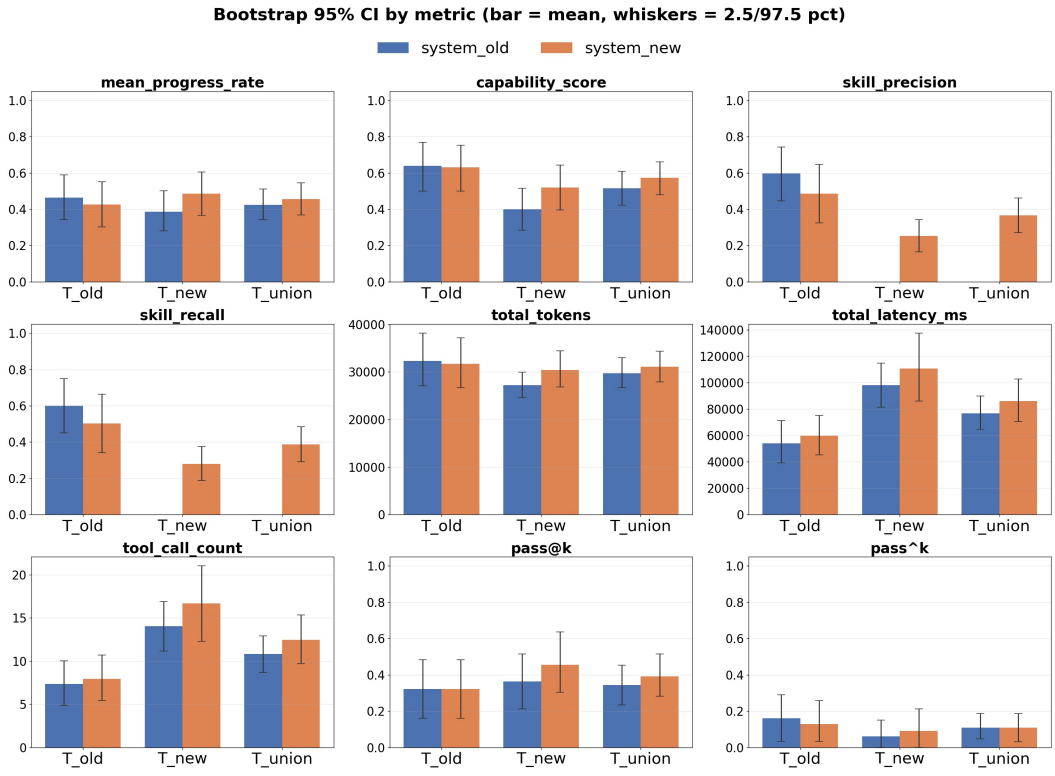


Figure 25: Bootstrap 95% confidence intervals for capability score, progress rate, skill precision/recall, and efficiency metrics on **general and single-domain** tasks for the **PI Opus 4.6** agent (Setting 3).

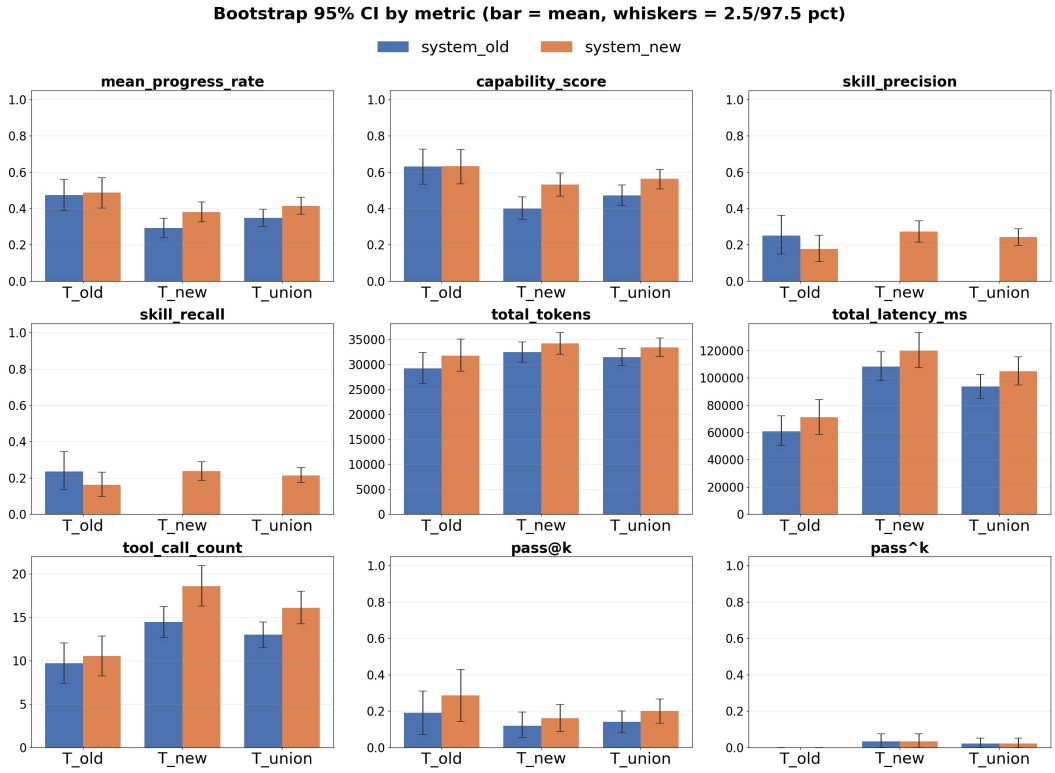


Figure 26: Bootstrap 95% confidence intervals for capability score, progress rate, skill precision/recall, and efficiency metrics on **cross-domain** tasks for the **PI Opus 4.6** agent (Setting 3).

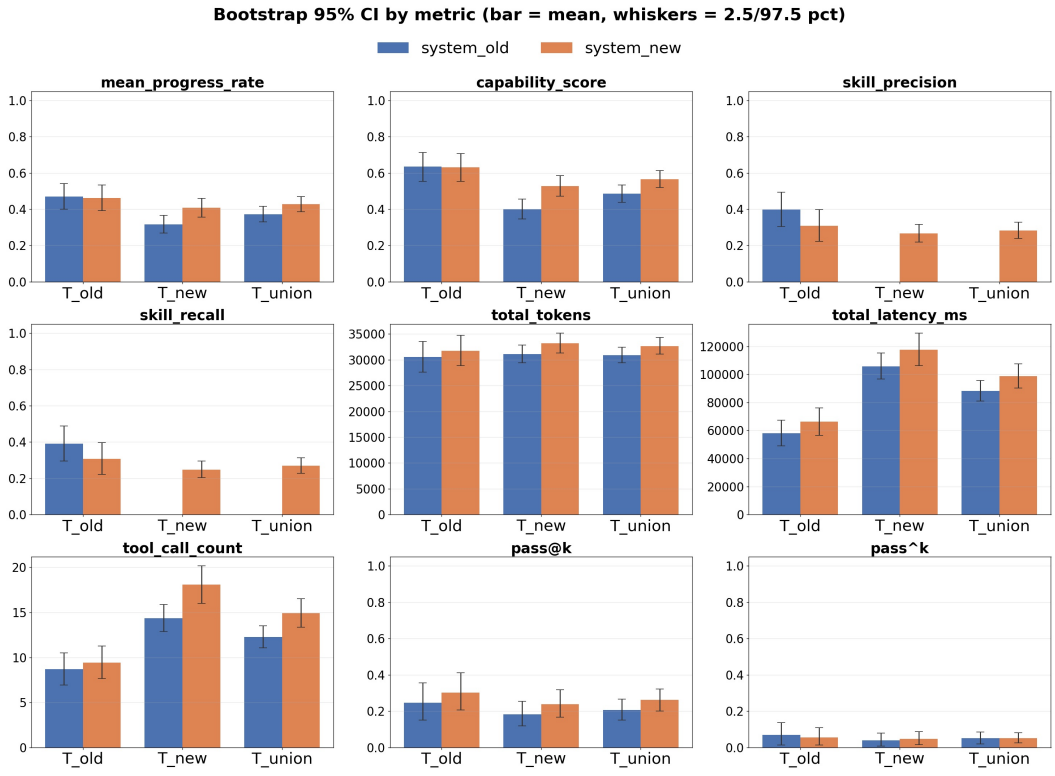


Figure 27: Bootstrap 95% confidence intervals for capability score, progress rate, skill precision/recall, and efficiency metrics on **general, single, and cross-domain** tasks for the **PI Opus 4.6** agent (Setting 3).