TOWARDS RELIABLE SKILL INTERNALIZATION VIA TRANSFERABILITY AND NON-INTERFERENCE

Anonymous authors
Paper under double-blind review

ABSTRACT

LLM agents can extract reusable skills from successful task experiences and internalize them into model parameters, enabling skill-guided behavior without explicit instructions at inference time. However, a single successful trajectory does not uniquely identify a reusable skill: multiple candidates may explain the same success but behave differently on new tasks. Internalizing an insufficiently validated skill may therefore reinforce behavior that fails to transfer to new tasks or interferes when the skill is unnecessary. We introduce TANI, a framework for validating and refining extracted skills before internalization. It evaluates candidate skills for *transferability* across tasks requiring the same knowledge and *non-interference* on tasks where that knowledge is unnecessary. The framework generates diagnostic tasks to test both properties and uses execution outcomes to select and refine candidate skills. We evaluate the effectiveness of our framework by internalizing retained skills using on-policy self-distillation (OPSD). On τ^2-bench Telecom and Retail, TANI achieves 50.0% and 70.0% Pass@3, respectively, outperforming existing baselines. Our analyses show that source-task success alone is insufficient for identifying skills suitable for internalization. Evaluating both transferability and non-interference enables more effective skill selection and refinement.

1 INTRODUCTION

LLM agents can acquire reusable knowledge by extracting *skills* from successful trajectories (Zhao et al., 2024; Wang et al., 2025b; Ouyang et al., 2026). These skills encode procedures or heuristics that guide future task execution, but typically require explicit retrieval and additional context at inference time (Wang et al., 2026). Recent work therefore explores *skill internalization*, which incorporates extracted skills into model parameters so that agents exhibit skill-guided behavior without explicit skill access (Huang et al., 2026a; Lin et al., 2026b). By incorporating extracted knowledge into model parameters, the agent learns persistent behavioral bias that affects across future interactions.

The challenge is that a single successful experience does not uniquely determine a reusable skill. A *source experience* records a task and the agent's successful execution trajectory, providing one observation of what worked in a particular setting. Yet multiple rules may explain that success while making different assumptions about future tasks. For example, successfully refunding an eligible order could suggest a sequence of actions tied to that particular order, a procedure that checks refund eligibility, or an instruction to refund whenever requested. All may succeed on the original task, but their behavior can diverge when the order changes or the requested refund is ineligible.

Internalizing such a skill without testing can reinforce two kinds of inappropriate behavior (Lin et al., 2026a). A task-specific procedure may mislead the agent on similar tasks with different detail, while an overly broad instruction may interfere when its guidance is unnecessary(Dong et al., 2026; Xiong et al., 2026). Prior work has reported negative transfer from low-quality extracted skills that omit essential knowledge or induce irrelevant actions (Huang et al., 2026a). These risks motivate two requirements for skills suitable for internalization: *transferability*, benefiting other tasks that require the same knowledge, and *non-interference*, avoiding interference when that knowledge is unnecessary. However, existing approaches that construct internalization data primarily from source experiences (Ye et al., 2026; Wang et al., 2026) leave these properties largely untested.

We introduce TANI (Transferability And Non-Interference before Skill Internalization), a framework for validating and refining extracted skills before internalization. Treating each candidate skill as a

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.

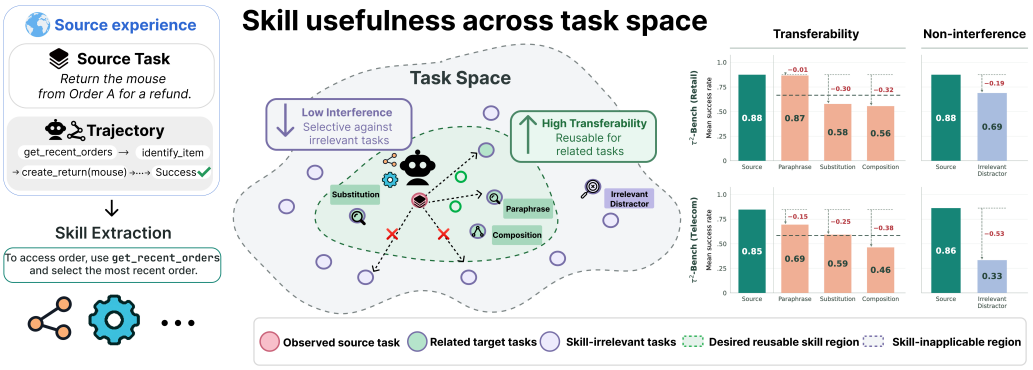

Figure 1: **Source-task success does not guarantee transferability or non-interference. Left:** A reusable skill should remain useful across related tasks requiring the same knowledge (transferability) without interfering with tasks where that knowledge is unnecessary (non-interference). **Right:** Despite high source-task success, extracted skills exhibit limited transferability and high interference.

behavioral hypothesis, TANI generates diagnostic tasks to evaluate two properties: transferability, whether the skill remains useful under varied task conditions where it is needed, and non-interference, whether it avoids impairing performance when its guidance is unnecessary. To evaluate transferability, we generate Paraphrase, Substitution, and Composition tasks that vary task wording, instance details, and additional requirements, respectively. To evaluate non-interference, we construct Irrelevant Distractor tasks that share context with the source task but do not require the candidate skill.

By comparing executions on the diagnostic tasks with the candidate skill, TANI measures the skill's contribution across these conditions. The resulting evidence guides skill preparation (selection and refinement): transferability failures inform revisions to procedural instructions, while non-interference failures can reveal applicability conditions that need to be narrowed. During the process, TANI naturally obtain source and synthesized tasks on which the verified skill demonstrates positive utility. These task–skill pairs are used for skill internalization. Thus, synthesized tasks serve two complementary purposes: preparing what skills should be internalized, and providing varied contexts the retained knowledge can be learned.

We evaluate TANI on the Telecom and Retail domains of τ^2-bench (Barres et al., 2025). TANI outperforms both source-only internalization and utility-filtered task augmentation. Using its full training pool, TANI achieves 50.0% and 70.0% Pass@3 on Telecom and Retail, respectively, exceeding existing baselines on both domains. Further analyses show that diagnostic evaluation improves skill selection even when synthesized tasks are excluded from student training. Removing Non-interference checks also reduces downstream performance in both domains, highlighting the value of evaluating where a skill should not be applied.

Our contributions are threefold:

- We identify a fundamental ambiguity in experience-based skill extraction: a successful experience does not uniquely determine transferable skill for internalization.
- We introduce TANI, which generates candidate-specific behavioral tests to curate skills before internalization, then reuses selected diagnostic tasks as additional training experience.
- We demonstrate the effects of skill validation and additional training tasks on downstream internalization.

2 How Source-Task Success Fails to Predict Skill Utility

When an agent successfully completes a task, its execution trajectory records the actions and decisions that led to success. This experience can then be used to extract a skill intended to guide future tasks. We define the *source experience* as $e_{\text{src}} = (t_{\text{src}}, \tau_{\text{src}})$, where t_{src} is the original task and τ_{src} is its successful execution trajectory.

A skill-induction process extracts a candidate skill from the source experience:

$$s = \mathcal{I}(e_{\mathrm{src}}; p), \tag{1}$$

where p is the skill-induction prompt. A single successful trajectory, however, may support multiple plausible skills that make different assumptions about which parts of the experience are essential and when they apply. Two common approaches to get reusable skills are (1) controlling the level of abstraction via prompt (Zhao et al., 2024; Wang et al., 2025b; Zhang et al., 2026); and (2) validating their performance on the source task (Wang et al., 2023; 2025a). However, neither prompting for generality nor source-task validation establishes whether a candidate will remain useful under different task conditions.

We empirically investigate whether a candidate skill extracted from a source experience with a general-skill prompt remains reliable beyond the source task. In particular, we test whether source-task success provides reliable evidence of transferability and non-interference.

Experimental setup. For each successful source experience, we extract a candidate skill using a general skill-induction prompt. We evaluate each candidate on its source task over eight runs and retain those achieving at least 75% success. We then examine the retained skills under two sets of task conditions.

The first set evaluates **transferability** using Substitution and Composition tasks. Substitution changes incidental task details while preserving the original goal and required knowledge, whereas Composition introduces additional requirements while retaining the original goal. The second set evaluates **non-interference** using Irrelevant Distractor tasks, which share relevant context with the source task but do not require the extracted skill. For each task, we compare agent performance with and without explicit skill access. This allows us to examine whether the skill helps when it is needed and whether it harms performance when it is unnecessary. We conduct the study on the Retail and Telecom domains of τ^2-bench (Barres et al., 2025) using Qwen3-4B-Instruct (Yang et al., 2025). No parameter updates are performed, allowing us to study the behavior of extracted skills before internalization.

Source-task success does not ensure transferability. Figure 1 shows that despite high source-task success, extracted skills exhibit substantial performance drops under task variation. Mean success across Paraphrase, Substitution, and Composition drops from 88% to 67% on Retail and from 85% to 58% on Telecom, representing declines of 21 and 27 percentage points, respectively. The declines are particularly pronounced on Substitution and Composition, indicating that source-task success does not ensure transferability when task condition changes.

Source-task success does not ensure non-interference. On Irrelevant Distractor tasks, where the extracted skill is unnecessary, success falls to 69% on Retail and 33% on Telecom, compared with source-task success of 88% and 86%, respectively. Across both domains, average success drops from 87% to 51%. These results reveal substantial performance gaps on tasks outside the skill's intended application, although skill-free comparisons are needed to establish whether the skill itself causes interference.

Together, these results reveal limitations of extracting skills solely based on source-task success. Motivated by these results, we argue that evaluating candidates on task variations that test both transferability and non-interference are essential before it is internalized into agent policy, using execution outcomes to guide skill selection and refinement before internalization.

3 Validating and Refining Skills Before Internalization

The core of our framework is a pre-internalization skill preparation process guided by transferability and non-interference. We treat each extracted skill as a hypothesis and evaluate it through diagnostic tasks designed to test these two properties. Execution outcomes then guide skill validation and refinement, helping identify which skills are suitable for internalization.

As illustrated in Figure 2, TANI follows four stages. SYNTHESIZE generates diagnostic tasks to test skill transferability and non-interference. PROBE measures the candidate's utility across these

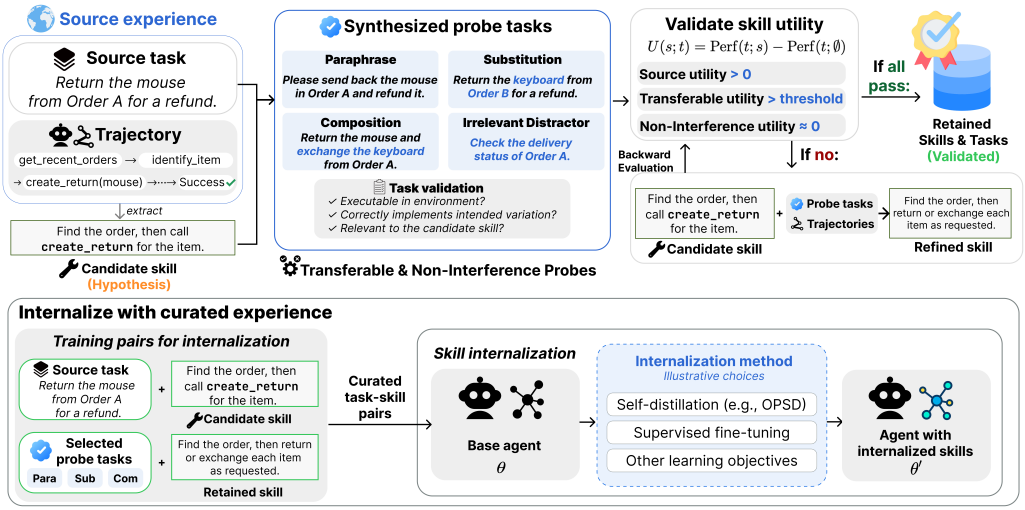

Figure 2: **Overview of TANI.** Starting from a source experience and candidate skill, TANI follows four stages. After obtaining validated pairs of skills and relevant tasks, synthesized tasks serve both as behavioral tests for skill preparation and as additional training tasks for the skill.

tasks. REFINE uses observed failures to revise the candidate's procedural instructions or applicability conditions, followed by re-evaluation. Finally, INTERNALIZE uses retained skills and selected positive-utility tasks for on-policy self-distillation (Zhao et al., 2026). Thus, synthesized experience serves both as behavioral evidence for determining what to learn and as additional experience for learning it. We demonstrate the downstream utility of the resulting skills using OPSD, without modifying its learning procedure.

3.1 SYNTHESIZE: GENERATING TRANSFERABILITY AND NON-INTERFERENCE VARIATIONS

Given a source task and a candidate skill, we generate two types of probe tasks to evaluate the skill's transferability and non-interference.

Transferability probes. We construct three types of variations that challenge different assumptions about a candidate's transferability. *Paraphrase* rephrases the source task while preserving its goal and instance details, testing robustness to changes in task wording. *Substitution* changes incidental details, such as the target account or product, while preserving the underlying strategy. This tests whether the skill captures a reusable procedure rather than depending on the original task instance. *Composition* retains the original goal but introduces additional requirements, testing whether the skill remains useful as part of a more complex task.

Non-interference probes. We generate *Irrelevant Distractor* tasks to test whether a candidate skill interferes with tasks that do not require it. Each probe is a simple, ordinary request from the same domain as the source task, sharing enough contextual similarity to make the skill appear potentially relevant while requiring a different procedure. We design these tasks so that the candidate skill is unnecessary and following its instructions would lead to an incorrect action. This allows us to assess whether the skill's applicability conditions are sufficiently precise to prevent it from interfering with related tasks outside its intended scope.

Task validation. To ensure that the generated tasks provide meaningful diagnostic evidence, we check executability programmatically and use an LLM judge to assess fidelity to the requested variation and relevance to the candidate skill. Invalid tasks are discarded or regenerated. Further details are provided in Appendix A.1.

To prevent probe categories with more valid tasks from dominating evaluation, we retain an equal number K of validated tasks from each category, determined by the minimum available count. Let $\mathcal{T}_c$

denote the resulting task set for category c. We define

$$\mathcal{T}_{\text{Trans}} = \mathcal{T}_{\text{Paraphrase}} \cup \mathcal{T}_{\text{Substitution}} \cup \mathcal{T}_{\text{Composition}}, \tag{2}$$

and denote the non-interference probe set by $\mathcal{T}_{\text{Non-interference}}$.

3.2 PROBE AND REFINE: TESTING AND REVISING SKILL HYPOTHESES

Evaluating skill utility. We compare agent performance with and without explicit skill access on the same task. Following prior work on skill evaluation (Huang et al., 2026b), we define

$$U(s;t) = \text{Perf}(t;s) - \text{Perf}(t;\emptyset), \tag{3}$$

where the two terms denote expected task performance with and without skill s, respectively. Positive utility indicates that the skill improves performance, whereas negative utility indicates interference.

We evaluate candidates using three criteria: source utility, reuse utility, and non-interference.

Skill evaluation. We evaluate each candidate using three criteria sequentially: source utility, reuse utility, and non-interference.

Source utility measures the skill's contribution on its original task, $U_{\text{src}}(s) = U(s; t_{\text{src}})$. For the diagnostic probes, we compute the average utility over each task set:

$$\bar{U}_c(s) = \frac{1}{|\mathcal{T}_c|} \sum_{t \in \mathcal{T}_c} U(s;t), \quad c \in \{\text{Trans}, \text{Non} - \text{interference}\}. \tag{4}$$

A candidate is retained only if it satisfies all three criteria: (i) its source utility is verified, $U_{\text{src}}(s) = 1$; (ii) its average reuse utility exceeds θ_{Trans}; and (iii) its average non-interference utility remains within a tolerance of zero, $0 \leq \bar{U}_{\text{Non-interference}}(s) \leq \epsilon$.

Candidate skills that fail any criterion during the sequential evaluation are examined using their diagnostic execution trajectories to guide skill revision.

Failure-guided refinement. We revise failed candidates using their source experience and informative successful and failed probe trajectories.

For example, consider a candidate extracted from a successful subscription cancellation on Site A:

> On Site A, open Account → Settings and cancel the subscription.

A failure on a Substitution probe involving another account may reveal the need to identify the target account before proceeding. A non-interference failure on a payment-method update may instead indicate that the skill is being applied outside its intended scope. These diagnostic cases can guide a revision such as:

> When subscription cancellation is requested, first identify the target account, then navigate to Account → Settings to cancel the requested subscription.

Inspired by Cao et al. (2026) which use contrastive trajectories for memory refinement, the revision model uses the trajectories of probe tasks to update the knowledge in candidate skill (*e.g.*, procedure or applicability conditions). We then re-evaluate the revised candidate on the source task and across all probe categories using the same acceptance criteria, since addressing one failure may introduce another.

Candidates that pass are retained, while those that continue to fail are rejected. Candidates that pass the initial evaluation are retained without revision. We denote each retained skill, whether unchanged or revised, by s^*. The execution outcomes therefore guide decisions about which skills to retain and how failed candidates should be revised. Revision prompts and details are described in Appendix A.2.

3.3 INTERNALIZE: LEARNING FROM VALIDATED SKILLS AND TASKS

For each retained skill s^* (*i.e.*, either the candidate skill or refined skill), we consider its original source task $t_{\text{src}}(s^*)$ and validated reuse probes $\mathcal{T}_{\text{Trans}}(s^*)$. From these, we select only tasks on which the skill demonstrates positive utility:

$$\mathcal{T}_{\text{int}}(s^*) = \{t \in \{t_{\text{src}}(s^*)\} \cup \mathcal{T}_{\text{reuse}}(s^*) \mid U(s^*; t) > 0\}. \tag{5}$$

non-interference probes are used only for diagnosis and are excluded from training.

Parameter updates. Our framework focuses on validating and refining skills before internalization, rather than introducing a new parameter-learning algorithm. We instantiate the internalization stage using on-policy self-distillation (OPSD) (Ye et al., 2026), which allows a student to learn from a skill-conditioned teacher along the student's own interaction trajectories. We adopt on-policy self-distillation (OPSD), a learning method widely used in prior work on skill internalization, to evaluate the effectiveness of our framework. Other learning methods, such as supervised fine-tuning on skill-conditioned trajectories, could also use the curated skills and tasks. We adopt OPSD as a common internalization procedure to evaluate the effectiveness of the proposed skill preparation process.

For each selected pair (t, s^*), the student interacts with task t without explicit skill access. A privileged teacher receives the same student-generated interaction history together with s^*.

For a student response $y \sim \pi_\theta(\cdot \mid h)$, we train the student to match the skill-conditioned teacher along its own on-policy trajectories by minimizing

$$\mathcal{L}_{\text{int}} = \mathbb{E}_{\substack{(t,s^*)\sim\mathcal{D}_{\text{int}},\\ h\sim p_{\pi_\theta}(\cdot|t),\\ y\sim\pi_\theta(\cdot|h)}} \left[\frac{1}{|y|} \sum_{j=1}^{|y|} D_{\text{KL}} \left(\pi_\theta(\cdot \mid h, y_{<j}) \,\|\, \pi_{\text{teacher}}(\cdot \mid h, s^*, y_{<j}) \right) \right]. \tag{6}$$

4 EXPERIMENTAL SETUP

Benchmarks. We evaluate on the Telecom and Retail domains of τ^2-bench (Barres et al., 2025), which require multi-step tool use under domain-specific customer-service policies. Each domain contains 74 training tasks and 40 test tasks. We extract candidate skills from successful trajectories collected on the training split and reserve the test split exclusively for final evaluation.

Evaluation. We report Pass@3, the fraction of tasks solved at least once in three independent executions, and Avg@3[1], the average success rate across executions. These metrics measure task-solving breadth and consistency, respectively.

Models and internalization. Inspired by recent approaches using different-sized models for agentic systems ((Xu et al., 2023; Wu et al., 2026), we use Qwen3-9B-Instruct-2507 for skill extraction, task synthesis, validation, and refinement and Qwen3-4B-Instruct-2507 as the solver agent for trajectory collection and as the student model for internalization.

All controlled experiments use the on-policy self-distillation procedure described in Section 3.3, with the same student initialization and optimization configuration within each comparison. Further implementation details are provided in Appendix B.

Baselines. We compare TANI against experience-reuse and parameter-learning methods. *Vanilla* is the base agent without additional learning or explicit skill access. *ReasoningBank* retrieves reusable experience at inference time without parameter updates (Ouyang et al., 2026). Among parameter-learning methods, *GRPO* optimizes the agent using task rewards (Yang et al., 2025), while *Skill-GRPO* additionally incorporates extracted skills (Wang et al., 2026). *Skill-SD* (Wang et al., 2026) and *Online Experiential Learning (OEL)* (Ye et al., 2026)use experience- or skill-derived supervision for parameter updates. All methods have access to the full training-task set and follow their respective training procedures.

[1] $Pass^3$ in the original τ^2-bench paper.

Table 1: Held-out performance on τ^2-bench. 74 task–skill pairs are used for controlled internalization methods. TANI (Full) uses all eligible pairs.

	Telecom		Retail	
Method	Pass@3	Avg@3	Pass@3	Avg@3
No parameter updates				
Vanilla	.300	.150	.575	.400
ReasoningBank	.325	.175	.650	**.492**
Parameter-based learning				
GRPO	.375	.242	.625	.400
Skill-GRPO	.300	.183	.650	.483
Skill-SD	.350	.258	.400	.225
OEL	.425	.250	.525	.400
Controlled internalization				
Source-only	.375	.183	.525	.400
Task aug.	.250	.133	.575	.392
TANI (Matched)	.400	.167	.625	.425
TANI (Full)	**.500**	**.325**	**.700**	.467

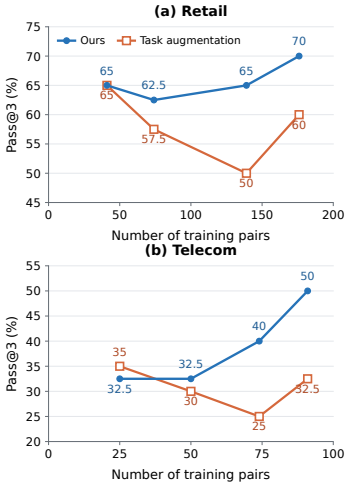

Figure 3: Downstream performance with increasing numbers of training (task–skill pairs).

Controlled internalization baselines. To examine the effects of skill validation and refinement separately from those of additional training tasks, we construct two baselines using the same OPSD procedure as TANI. *Source-only internalization* trains directly on the original source task–skill pairs, without utility filtering, diagnostic probing, or refinement.

Task augmentation (Task aug.) generates diverse task variations using GPT-5.6 Luna, following Liu et al. (2025), without conditioning generation on candidate skills. Generated tasks undergo the same validity and executability checks as TANI, and only positive-utility task–skill pairs are retained. Unlike TANI, these tasks are not used to diagnose or refine candidate skills.

Comparison protocol. Because different skill-preparation methods may produce different numbers of eligible training pairs, we match the number of task–skill pairs and the optimization budget within each controlled comparison. The main comparison uses 74 training pairs for Source-only, Task aug., and TANI. We additionally report TANI (Full), which uses all eligible pairs (non-budget-matched). For the scaling experiments, TANI and Task aug. are compared under matched pair budgets at each evaluation point.

5 EXPERIMENTAL RESULTS

We investigate three research questions: (i) **RQ1:** Does validating and refining skills before internalization improve downstream performance? (ii) **RQ2:** How do skill preparation and additional training tasks contribute to downstream performance? (iii) **RQ3:** How effectively do diagnostic probes identify skills suitable for internalization?

5.1 DOWNSTREAM PERFORMANCE AND SCALING

TANI improves downstream internalization performance. Table 1 compares TANI with existing methods and controlled internalization baselines. Under the same 74-pair training budget, TANI achieves 40.0% and 62.5% Pass@3 on Telecom and Retail, respectively, exceeding both Source-only and Task aug. In contrast, Task aug. does not consistently outperform Source-only, achieving 25.0% versus 37.5% on Telecom and 57.5% versus 52.5% on Retail. These results suggest that additional utility-verified training tasks alone do not account for the improvements from the proposed framework

With its full eligible training pool, TANI achieves 50.0% and 70.0% Pass@3 on Telecom and Retail, respectively, exceeding the existing baselines in both domains on this metric. Because this

configuration uses more training pairs, we further compare TANI with Task aug. across multiple matched training budgets.

Scaling with synthesized experience. Figure 3 shows that increasing the training budget does not consistently improve Task aug. Its Pass@3 drops from 65.0% to 50.0% on Retail and from 35.0% to 25.0% on Telecom before partially recovering. In contrast, TANI generally benefits from larger training budgets, reaching 70.0% and 50.0% Pass@3 on Retail and Telecom, respectively. These results extend the fixed-budget comparison, showing that the effectiveness of additional synthesized experience depends on how it is prepared and used.

5.2 DISENTANGLING THE SOURCES OF IMPROVEMENT

TANI uses synthesized tasks both to prepare skills and to provide additional training contexts. We examine these two contributions separately on Retail, matching the number of training pairs budget within each comparison.

To assess skill preparation, we compare original extracted skills with those retained by TANI using source and selected reuse tasks, with 139 training pairs per condition. To assess task diversification, we fix the retained skills and compare source-only training with training on source and selected reuse tasks, using 41 pairs per condition.

The first comparison evaluates original and retained skills using the source and selected reuse task pool, with 139 training pairs sampled separately. The second comparison fixes the retained skills and compares training contexts using 41 pairs per condition. Budgets are matched within, but not across, the two comparisons.

Table 2: Effects of skill preparation and additional training contexts on Retail.

(a) Skill preparation
139 training pairs

Teacher skill	Pass@3
Original s	.625
Retained s^*	**.650**

(b) Training contexts
41 training pairs

Training tasks	Pass@3
Source only	.575
Source + Reuse	**.650**

Skill preparation and task diversification provide distinct benefits. Table 2(a) shows that using retained skills increases Pass@3 from 62.5% to 65.0% with 139 training pairs. Because the two conditions may differ in their eligible task–skill pairs, this comparison reflects the skill-preparation process rather than skill revision alone. Separately, Table 2(b) shows that adding selected reuse tasks increases Pass@3 from 57.5% to 65.0% when the retained skills and 41-pair training budget are fixed. These results suggest that both skill preparation and additional learning contexts contribute to downstream internalization.

5.3 EFFECTS OF DIAGNOSTIC EVALUATION ON SKILL PREPARATION

Diagnostic skill selection. We compare five skill-preparation strategies on Retail, starting from the same pool of extracted candidates: selection by source-task utility, utility on skill-agnostic task variations, textual generality, and TANI with or without probe-guided refinement. Random-task evaluation uses the same reuse-utility threshold as TANI. To isolate the diagnostic role of synthesized tasks, all strategies retain the same number of skills and train only on their corresponding source tasks, with matched training budgets.

Table 3: Comparison of skill evaluation and refinement strategies on Retail.

Method	Pass@3	Avg@3
Source utility	.525	.400
Random-task evaluation	.600	.442
Textual generality	.575	.425
TANI w/o refinement	.625	**.500**
TANI	**.650**	.475

Diagnostic evaluation improves skill selection. Table 3 shows that source-utility selection achieves 52.5% Pass@3, while textual generality and random-task evaluation achieve 57.5% and 60.0%, respectively. Validation with diagnostic tasks achieves 62.5% without refinement and 65.0% with refinement. Refinement provides an additional 2.5 percentage-point gain in Pass@3, although Avg@3 decreases from 50.0% to 47.5%.

These results indicate that candidate-specific diagnostic tasks provide useful evidence for skill preparation even when the synthesized tasks are excluded from training.

Non-interference ablation. We assess the contribution of Non-interference by comparing the full TANI pipeline with a variant that removes Non-interference probes and their acceptance criterion. Both variants start from the same candidate pool and retain source and reuse evaluation, probe-guided refinement, and on-policy self-distillation. Applying the Non-interference criterion $S(s)$ reduces eligible training pairs by 21.6% on Telecom (116 to 91) and 4.3% on Retail (184 to 176).

Reuse checks alone are insufficient for skill preparation. Figure 4 shows that removing Non-interference reduces Pass@3 from 50.0% to 37.5% on Telecom and from 70.0% to 62.5% on Retail. Avg@3 also decreases in both domains, despite the variant without Non-interference probes using more training pairs. This suggests that without Non-interference checks, candidates with overly broad applicability conditions may be retained despite demonstrating positive utility on compatible tasks, which highlights the need for testing skill applicability boundaries before internalization.

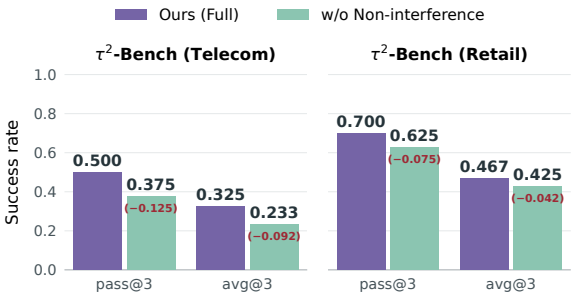

Figure 4: Effect of Non-interference on downstream performance.

6 Related Work

6.1 Skill Extraction and Validation

Recent work has examined how reusable skills can be extracted, evaluated, and improved from agent experience. Huang et al. (Huang et al., 2026a) systematically study model-generated skills and identify negative transfer across different task contexts. SkillOpt (Yang et al., 2026) optimizes external skill documents through execution-guided revisions and held-out validation. Related failure analyses further show that seemingly relevant skills can induce incorrect or unnecessary actions (Dong et al., 2026). Building on these findings, TANI generates candidate-specific diagnostic tasks to evaluate both skill reuse and applicability before internalization.

6.2 Skill Internalization

Recent approaches transfer external skills into model parameters to enable skill-guided behavior without inference-time retrieval. Online Experiential Learning (OEL) (Ye et al., 2026) extracts transferable knowledge from interaction trajectories and consolidates it through on-policy context distillation. SKILL0 (Lu et al., 2026) progressively withdraws skill context according to its on-policy helpfulness, while SkillC (Lin et al., 2026b) uses paired skill-conditioned and skill-free rollouts for contrastive credit assignment. While these methods develop mechanisms for learning from external knowledge, TANI focuses on preparing and validating the skills and training experiences supplied for internalization.

7 Conclusion

We introduced TANI, a framework that validates and refines extracted skills based on transferability and non-interference before internalization. Experiments on Telecom and Retail demonstrate improved downstream performance. Our analyses show that evaluating skills on generated tasks improves skill selection, while using selected tasks for internalization provides additional training contexts. These findings highlight the importance of validating which skills should be learned before incorporating them into model parameters.

AI USE STATEMENT

In this work, we used generative AI tools for manuscript writing and editing, including improving clarity, grammar, and presentation. We also used these tools for code generation and programming assistance, as well as for monitoring experimental outputs to identify anomalous behavior, inconsistencies, or potential issues. Generative AI tools were not used to autonomously produce research findings, experimental results, or scientific conclusions. All AI-assisted text was reviewed and revised by the authors, and all AI-generated or AI-assisted code was manually inspected, verified, and tested before being incorporated into the final codebase. AI-assisted monitoring was used only as a supplementary mechanism, and the authors independently examined the experimental outputs and made all final judgments regarding their validity and interpretation. The authors take full responsibility for the final content of this work, including its text, claims, code, experimental analyses, and other artifacts produced with the assistance of generative AI.

REFERENCES

Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. τ^2-bench: Evaluating conversational agents in a dual-control environment, 2025.

Zouying Cao, Jiaji Deng, Li Yu, Weikang Zhou, Zhaoyang Liu, Bolin Ding, and Hai Zhao. Remember me, refine me: A dynamic procedural memory framework for experience-driven agent evolution. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 16803–16822, 2026.

Gen Dong, Yanjie Gao, Liqun Li, Tianyin Xu, Yu Hua, and Fan Yang. Agent skills can be harmful: An empirical study of skill-induced failures in llm agents. *arXiv preprint arXiv:2608.11888*, 2026.

Zisu Huang, Jingwen Xu, Yifan Yang, Ziyang Gong, Qihao Yang, Muzhao Tian, Xiaohua Wang, Changze Lv, Xuemei Gao, Qi Dai, Bei Liu, Kai Qiu, Xue Yang, Dongdong Chen, Xiaoqing Zheng, and Chong Luo. From raw experience to skill consumption: A systematic study of model-generated agent skills, 2026a.

Zisu Huang, Jingwen Xu, Yifan Yang, Ziyang Gong, Qihao Yang, Muzhao Tian, Xiaohua Wang, Changze Lv, Xuemei Gao, Qi Dai, et al. From raw experience to skill consumption: A systematic study of model-generated agent skills. *arXiv preprint arXiv:2605.23899*, 2026b.

Hongqiang Lin, Chao Liu, Xiaofan Bai, Xuan Jin, Yuhong Li, Nenggan Zheng, and Xipeng Cao. Rethinking self-evolution: A constrained exploration-exploitation process for mitigating skill overfitting. *arXiv preprint arXiv:2607.26643*, 2026a.

Hongxiang Lin, Zhirui Kuai, Erpeng Xue, and Lei Wang. SKILLC: Learning autonomous skill internalization in LLM agents via contrastive credit assignment. *arXiv preprint arXiv:2605.27899*, 2026b.

Bo Liu, Chuanyang Jin, Seungone Kim, Weizhe Yuan, Wenting Zhao, Ilia Kulikov, Xian Li, Sainbayar Sukhbaatar, Jack Lanchantin, and Jason Weston. Spice: Self-play in corpus environments improves reasoning. *arXiv preprint arXiv:2510.24684*, 2025.

Zhengxi Lu, Zhiyuan Yao, Jinyang Wu, Chengcheng Han, Qi Gu, Xunliang Cai, Weiming Lu, Jun Xiao, Yueting Zhuang, and Yongliang Shen. SKILL0: In-context agentic reinforcement learning for skill internalization. *arXiv preprint arXiv:2604.02268*, 2026.

Siru Ouyang, Jun Yan, I Hsu, Yanfei Chen, Ke Jiang, Zifeng Wang, Rujun Han, Long Le, Samira Daruki, Xiangru Tang, et al. Reasoningbank: Scaling agent self-evolving with reasoning memory. In *International Conference on Learning Representations*, volume 2026, pp. 94327–94354, 2026.

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

Hao Wang, Guozhi Wang, Han Xiao, Yufeng Zhou, Yue Pan, Jichao Wang, Ke Xu, Yafei Wen, Xiaohu Ruan, Xiaoxin Chen, et al. Skill-sd: Skill-conditioned self-distillation for multi-turn llm agents. *arXiv preprint arXiv:2604.10674*, 2026.

Zora Zhiruo Wang, Apurva Gandhi, Graham Neubig, and Daniel Fried. Inducing programmatic skills for agentic tasks. *arXiv preprint arXiv:2504.06821*, 2025a.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 63897–63911. PMLR, 13–19 Jul 2025b.

Wenyi Wu, Sibo Zhu, Kun Zhou, and Biwei Huang. Planner matters! an efficient and unbalanced multi-agent collaboration framework for long-horizon planning. *arXiv preprint arXiv:2605.02168*, 2026.

Zidi Xiong, Yuping Lin, Wenya Xie, Pengfei He, Zirui Liu, Jiliang Tang, Himabindu Lakkaraju, and Zhen Xiang. How memory management impacts llm agents: An empirical study of experience-following behavior. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 623–645, 2026.

Binfeng Xu, Zhiyuan Peng, Bowen Lei, Subhabrata Mukherjee, Yuchen Liu, and Dongkuan Xu. Rewoo: Decoupling reasoning from observations for efficient augmented language models. *arXiv preprint arXiv:2305.18323*, 2023.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Yifan Yang, Ziyang Gong, Weiquan Huang, Qihao Yang, Ziwei Zhou, Zisu Huang, Yan Li, Xuemei Gao, Qi Dai, Bei Liu, et al. Skillopt: Executive strategy for self-evolving agent skills. *arXiv preprint arXiv:2605.23904*, 2026.

Tianzhu Ye, Li Dong, Qingxiu Dong, Xun Wu, Shaohan Huang, and Furu Wei. Online experiential learning for language models. *arXiv preprint arXiv:2603.16856*, 2026.

Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, et al. Agentic context engineering: Evolving contexts for self-improving language models. In *International Conference on Learning Representations*, volume 2026, pp. 86069–86100, 2026.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. Expel: Llm agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 19632–19642, 2024.

Siyan Zhao, Zhihui Xie, Mengchen Liu, Jing Huang, Guan Pang, Feiyu Chen, and Aditya Grover. Self-distilled reasoner: On-policy self-distillation for large language models. *arXiv preprint arXiv:2601.18734*, 2026.

A Details of Tani

A.1 Details of Task Validation Process in Tani

Every probe written by the generator (Appendix D.3) is checked by a validator before it is used to measure skill utility. The validator applies up to three criteria, and which of them a probe must pass depends on its category (Table 4). A probe is valid only if it passes every criterion required for its category.

Executability. Executability is checked deterministically, without an LLM. First, every identifier the probe refers to must exist in the environment database and be consistent with it: in Retail, for example, an order must belong to the stated user, an item may be exchanged only for another option of the same product, and a return requires a delivered order; in Telecom, the customer, line, and injected faults must match the environment. Second, the probe's reference actions are replayed on a freshly initialized environment (in Telecom, after the initialization actions that inject the faults), and every call must succeed. Probes that fail the first step are discarded as soon as they are generated.

Table 4: Validation criteria required for each probe category.

Category	Executability	Fidelity	Relevance
Paraphrase	✓		
Substitution	✓	✓	✓
Composition	✓		✓
Irrelevant Distractor	✓	✓	✓

Fidelity. Fidelity asks whether the probe made the kind of change its category requests, relative to the source task. A Substitution probe must keep the request type and the applicable policy branch while changing the entities; a Non-interference probe must be a different kind of request from the same domain rather than the source request with its entities swapped or its outcome flipped.

Relevance. Relevance asks whether the probe is a valid item of its category with respect to the skill: in Substitution, the skill's trigger and procedure must still apply after the change; in Composition, both skills must be necessary, each for a different part of the request; in Selectivity, the correct procedure must differ from the skill's, so that applying the skill would be a mistake.

Fidelity and relevance are judged by Qwen3.5-9B, which returns a JSON verdict with a short reason; the prompts are given in Appendix D.4. Each criterion is judged with a single call. Paraphrase tasks are checked only for executability, since they restate the source task and therefore preserve its request and the skill's applicability by construction. Composition tasks are not checked for fidelity, since they are new requests that combine two skills rather than controlled edits of the source task.

Probe budget and selection. Failed probes are neither repaired nor regenerated: the generator writes a fixed number of probes per category (5 Paraphrase and 10 each of Substitution, Composition, and Irrelevant Distractor), and only those that pass validation are kept. From the valid probes of each category, up to four are drawn at random with a fixed seed. A candidate skill with no valid probe in any one category is discarded before any rollout. The selected probes are then solved once with the skill injected (Appendix D.2) and once without it. In RETAIL, 9,620 probes were generated, of which 4,024 passed validation; in TELECOM, 4,245 were generated and 2,120 passed.

A.2 DETAILS OF REFINEMENT PROCESS IN TANI

A candidate skill can pass its source task and still fail on the probes, typically because its trigger is stated too narrowly for the task it came from, or because it omits a step that the source task did not require. Refinement gives such a skill one chance to be rewritten.

Which skills are refined. Skills that already pass the final validation rule are kept unchanged and never refined. For a skill that fails it, each Substitution or Composition category in which at least one with-skill execution failed is refined separately, yielding at most one revision per category. Paraphrase and Irrelevant Distractor failures do not trigger refinement: a Paraphrase restates the skill's own source task, so its failure gives a rewrite little to learn from, and Irrelevant Distractor is a non-interference check in which not using the skill is the intended behavior.

Refinement input. The refinement model (Qwen3.5-9B) receives the skill, a one-line description of what the category tests, and the task description and trajectory of one randomly chosen failed with-skill execution from that category; long trajectories are truncated to their final turns, where failures usually appear. When the same category also contains a successful with-skill execution, its task description and trajectory are added as a contrast. The model is asked to keep what was already correct, fix what fell short (e.g., a trigger stated too narrowly, a missing step, or advice that should be an instruction), and return a single general skill without episode-specific details, together with a short note on what changed. The prompt is given in Appendix D.5.

Re-validation. A revised skill is evaluated only in the category it was revised for, on the same probes that were selected for the original skill; no new probes are synthesized or judged. Only the with-skill executions are rerun; the without-skill executions from the first pass are reused, since they

Table 5: Targeted task variations used to probe a candidate skill. For illustration, the source task is *cancel the Pro plan on site A*.

Probe	Task variation	What it tests	Example task
Paraphrase	Paraphrase the task while preserving its semantics and instance details	Robustness to surface-form changes	Terminate the Pro subscription on site A
Substitution	Change incidental details while preserving the required strategy	Dependence on source-specific details	Cancel the Team plan for another account on site A
Composition	Add another requirement while retaining the original one	Usability alongside additional requirements	Cancel the Pro plan on site A and then request a refund
Selectivity	Construct a related task where the strategy is unnecessary	Interference outside the skill's useful scope	Update the payment method on site A

do not depend on the skill. The revision is accepted when its utility in that category is above zero, and it contributes training pairs only from that category. We perform a single refinement round.

B Additional Experimental Details

B.1 Benchmark Environment and Evaluation

The RETAIL and TELECOM environments provide 16 domain-specific tools for customer-service operations and 13 for account and service management, respectively; TELECOM additionally gives the simulated user 30 device-side tools (e.g., toggling airplane mode or reseating the SIM card). We use the standard τ^2-bench user simulator, instantiated with GPT-4.1-mini at temperature 0.0. Task success is determined by the benchmark reward, which requires all task-specific final-state checks to pass.

For synthesized tasks, the generator produces a JSON task specification comprising the user's goal, the information the user knows and does not know, and behavioral instructions for the simulated user; a reference solution given as an ordered sequence of tool calls; and success checks in the form of natural-language assertions and values the agent must communicate. For TELECOM, it additionally specifies initialization actions that inject faults into the device and account, and environment assertions that the reported symptom is resolved. The specification is converted into a τ^2-bench task whose target database state is obtained by replaying the reference solution.

The solver and the user simulator are sampled at temperature 0.0, and the solver runs without thinking. Validation rollouts are capped at 100 simulation steps and held-out evaluations at 200 steps (the τ^2-bench default). All reported evaluations use three independent executions per held-out task.

B.2 Skill Extraction and Source Validation

For each source task, we run the solver (Qwen3-4B-Instruct-2507) once against the GPT-4.1-mini user simulator, with at most 100 simulation steps, and keep the resulting trajectory regardless of whether it succeeds. From each trajectory, GPT-4.1 extracts seven candidate skills, using a prompt that depends on the trajectory's outcome: skills that capture what made a successful trajectory work, or skills that would have prevented a failed one (Appendix D.1). We evaluate each candidate on its source task by injecting it into the solver's system prompt and executing the task three times under the same settings; its source success rate is the fraction of these executions that succeed. A candidate enters the controlled experiment pool if at least one of the three executions succeeds (pass@3), which retains 276 of 518 candidates in RETAIL and 123 of 518 in TELECOM.

The same eligible source cohort is used across controlled internalization comparisons. When a baseline does not require a particular validation stage, we nevertheless start from the same source experiences so that differences do not arise from using an easier subset of source tasks.

B.3 Controlled Baseline Construction

Unfiltered internalization. The unfiltered baseline pairs each source task with its extracted candidate skill and applies the same teacher–student self-distillation procedure as TANI. No utility-based filtering, synthetic task generation, or skill refinement is performed. Candidates are drawn uniformly at random from all 518 extracted skills per domain; since this pool is larger than every training budget we use, each pair appears at most once and no resampling is needed.

General task augmentation. For each source task, GPT-5.6 Luna receives the domain policy, the agent tool catalog, the source task as the sole semantic template, and grounded records from the environment database (RETAIL: candidate users, orders, and products; TELECOM: the fault catalogue, the available environment calls, and a target customer line and user persona), but is not shown the candidate skill. The prompt asks for an executable variant of the source task that keeps its intent and procedure while changing its entities (in TELECOM, the customer line and persona while keeping the reported issue and injected faults), rather than a candidate-specific diagnostic case, and lists earlier variants of the same source to avoid duplicates. We generate 25 candidates per source in RETAIL; in TELECOM, a source admits at most 20 distinct line–persona combinations, so it yields at most 20.

Generated tasks undergo the same executability checks as TANI. We then evaluate the original candidate skill on each valid task and compute

$$U(s;t) = \mathrm{Perf}(t;s) - \mathrm{Perf}(t;\emptyset), \tag{7}$$

with one execution of the solver per condition. Pairs with $U(s;t) = 1$, i.e., tasks solved with the skill and failed without it, form the eligible augmentation pool (807 pairs over 245 skills in RETAIL; 266 pairs over 76 skills in TELECOM).

We sample 74 task–skill pairs per domain uniformly at random from this pool, equal to the matched TANI training budget.

Importantly, we match the number of final student training pairs, not the number of generation attempts. The total number of generated tasks and the computation required to obtain the matched pool are therefore reported separately.

TANI matched sampling. The complete TANI pipeline may yield more eligible reuse pairs than are used in the matched experiment. We first construct the full eligible pool after diagnostic evaluation and any necessary skill refinement (176 pairs in RETAIL and 91 in TELECOM). We then sample 74 task–skill pairs per domain uniformly at random from this pool, matching the controlled training budget.

The unrestricted TANI-Full model instead trains on all eligible pairs (176 in RETAIL; 91 in TELECOM). Its result is reported separately because its student-training budget is larger.

B.4 Skill Evaluation and Refinement

For reuse probes (Paraphrase, Substitution, and Composition), we compute skill utility using one execution with and one without access to the candidate skill on each selected probe,

$$U_c(s) = \frac{1}{|T_c|} \sum_{t \in T_c} \big[\mathrm{Perf}(t;s) - \mathrm{Perf}(t;\emptyset)\big], \tag{8}$$

where T_c is the set of (at most four) selected probes of category c and Perf is binary task success. The reuse criterion passes if the mean utility over the three reuse categories is above zero. Irrelevant Distractor passes if $U_{\text{sel}}(s) \geq 0$, i.e., the skill does not reduce performance on tasks where it should not be applied; Irrelevant Distractor is a non-interference check, not a benefit requirement. A candidate without any paired Irrelevant Distractor execution fails this check.

Candidates satisfying both criteria are retained unchanged. For candidates that fail, each Substitution or Composition category containing at least one failed with-skill execution triggers refinement;

Paraphrase and Irrelevant Distractor failures do not. The refinement model (Qwen3.5-9B) receives the candidate skill, a description of what the category tests, the task description and trajectory of one failed with-skill execution, and, when available, those of one successful execution from the same category. The revised candidate is evaluated again on the same probes of the category it was revised for, with the skill injected; the without-skill executions from the first pass are reused. A revision is accepted when its utility in that category is above zero. We allow at most one refinement round per failed category.

The final skill $\hat{s}$ used in internalization is therefore either the unchanged original candidate or a revised candidate that satisfies the refinement criterion in the category it was revised for.

B.5 Self-Distillation Details

For each task–skill pair, the privileged teacher receives the skill appended to the end of the agent's system prompt, after the domain policy, as an `- EXPERIENCE ITEM:` line (truncated to at most 2,048 tokens), together with the same tool schemas and conversation history as the student. The student receives the same interaction context without the skill. We optimize the token-level reverse KL divergence $D_{\mathrm{KL}}(\pi_\theta(\cdot \mid x) \,\|\, \pi_{\mathrm{T}}(\cdot \mid x, s))$, computed over the student's top-256 vocabulary entries at each position (without renormalization) and averaged over all response tokens, over student rollouts: the student first plays every task once without the skill (one rollout per task, sampling temperature 0.7, with a GPT-4.1-mini user simulator at temperature 0), and every agent turn of these trajectories is then re-scored by the teacher. The teacher does not generate tokens; it is a frozen copy of the initial checkpoint that only provides log-probabilities.

All controlled experiments use AdamW (β_1=0.9, β_2=0.999, weight decay 0.01) with a constant learning rate of 3×10^{-6}, no warmup, and gradient clipping at 1.0; a batch of 8 task–skill pairs per update (all agent turns of the 8 episodes), with token-based micro-batches of at most 10,240 tokens per GPU accumulated into a single optimizer step; a maximum prompt length of 8,192 tokens and a maximum response length of 1,024 tokens per agent turn; one epoch, i.e. $\lceil N/8 \rceil$ updates for N pairs (with N trimmed to a multiple of 8); one student rollout per task; and no teacher sampling (the teacher only scores the student's tokens). Each run starts from the same `Qwen3-4B-Instruct-2507` checkpoint (non-thinking mode). We train one independent seed per configuration.

B.6 Matched-Pair Component Analysis

Skill-quality comparison. This comparison isolates the contribution of refinement. One condition draws task–skill pairs only from original candidates that pass the selection criteria; the other additionally includes the revised skills $\hat{s}$ accepted by refinement, whose original candidates were rejected. Both conditions use the same source-plus-synthetic task pool and the same utility filtering. Because the refined condition yields more eligible pairs, we set the comparison budget to the smaller pool and sample the larger condition down to this size.

Training-pool comparison. Both conditions use the same final skills (original candidates that pass selection and accepted revisions). One draws training pairs only from source tasks; the other draws from the union of source and eligible synthetic tasks. The number of student-training pairs is matched to the smaller source-only pool, again by global uniform sampling. Each skill contributes at most one source pair, kept only when the skill improves on the source task over the memory-free execution; repeated source executions are used only to decide this, not added as separate pairs. We therefore match the number of unique task–skill pairs, not the number of rollouts.

These two comparisons use independently determined budgets and are therefore not interpreted as a single four-cell factorial experiment.

C Additional Diagnostic Ablations

Table 6 varies two design parameters of the diagnostic pipeline one at a time, holding all other settings at their defaults (bold). Downstream performance is Pass@3 of the trained student on the held-out test tasks.

Table 6: Sensitivity to probe budget and acceptance thresholds. Each block varies one hyperparameter while keeping the remaining settings fixed.

Factor	Setting	Telecom Pass@3	Retail Pass@3
Probes/category K	1	.375	.650
	2	.250	.600
	4	**.500**	**.700**
Positive threshold τ_+	**0**	**.500**	**.700**
	0.125	.400	.650
	0.250	.375	.675

C.1 Probe Budget

We vary the number of validated probes per category over $K \in \{1, 2, 4\}$. The $K{=}1$ and $K{=}2$ settings are obtained by subsampling the valid probes of the default run (seed 42, so the $K{=}1$ set is contained in the $K{=}2$ set), and refinement in these settings uses only failures and successes inside the subsample; the synthesis budget is therefore identical across K, and only the number of validation rollouts changes. $K{=}4$ gives the best result in both domains (Telecom .500, Retail .700). With one or two probes per category, each category utility rests on one or two paired executions and can only take the values $\{-1, 0, 1\}$ or $\{-1, -0.5, 0, 0.5, 1\}$, so a single noisy execution decides whether a candidate is kept. The ordering between $K{=}1$ and $K{=}2$ is not consistent with a monotone trend, which we attribute to this estimation noise rather than to a systematic effect of the budget.

C.2 Utility Threshold

We vary the reuse-utility threshold τ_+ over $\{0, 0.125, 0.25\}$. Raising the threshold removes both skills and their training pairs, so it changes the skill set and the training pool at the same time. The default $\tau_+{=}0$ performs best in both domains. Retail is relatively insensitive to the threshold (.650–.700), whereas Telecom drops from .500 to .400 and .375. Since the positive-utility filter already restricts training pairs to tasks on which the skill changed the outcome, a stricter skill-level threshold mainly discards skills whose average benefit is small but whose retained pairs are still informative.

D Prompts

This section summarizes the prompts used at each stage of our pipeline. We show condensed versions that keep each prompt's structure and core instructions; text in braces (e.g., `{source_task}`) is filled in at run time. Domain-specific phrases, such as which identifiers exist and which policy branches apply, are instantiated separately for Retail and Telecom. Throughout, the user is played by the default τ^2-bench user simulator, which is given the task's scenario (`reason_for_call`, `known_info`, `unknown_info`, `task_instructions`); we do not modify its prompt.

D.1 Skill Extraction

For each source task, the extractor receives the task and the solver's trajectory, and induces candidate skills. The system prompt depends on the trajectory's outcome.

Skill extraction.

```
[System]
You are an expert customer-service tool agent. You will be given a task
description and {a successful | a failed} trajectory.
Extract exactly {k} concrete, actionable, and generalizable memory items that
{could help solve | would prevent the same failure in} future tasks in the
same domain. Focus on
  (success) policy reasoning, tool ordering, information verification,
            and communication strategy;
```

```
  (failure) policy violations, missing verification, incorrect tool
            ordering, and poor communication.
Do not include task-specific data (customer names, IDs, order numbers).
Format each item as:
  # Memory Item i
  ## Title <title>
  ## Description <one sentence on when to use the item>
  ## Content <1-3 sentences describing the reusable procedure>

[User]
Task: {source_task_description}
Trajectory: {trajectory}
```

D.2 SKILL INJECTION

A skill is given to the solver by appending it to the agent's system prompt. The instruction and policy blocks are the τ^2-bench defaults; in the without-skill condition the `<memory>` block is omitted.

Solver system prompt with an injected skill.

```
<instructions> {agent_instruction} </instructions>
<policy> {domain_policy} </policy>
<memory>
Below are some memory items that accumulated from past interaction with the
environment that may be helpful to solve the task. Use them when relevant.
In each step, first explicitly discuss whether to use each memory item, and
then take action.

{memory_items}
</memory>
```

D.3 PROBE SYNTHESIS

The generator writes one probe task per call as a JSON task specification, which is then converted into a τ^2-bench task. Every call shares the same system prompt and user-prompt layout; only the level instruction changes.

Probe synthesis: system prompt.

```
Generate a synthetic task for the {domain} environment: the user's goal,
what they know, how they behave, a reference solution, and evaluation checks.
The task is executed by an agent talking to a simulated user and scored
automatically, so it must be fully grounded and unambiguous.
Return a single JSON object and nothing else.
```

Probe synthesis: user prompt.

```
## Domain policy
{domain_policy}
## Available agent tools
{tool_catalog}
## Source task and outcome
{source_task}   (outcome: {source_status})
## Memory item under validation
{skill}
## Second memory for composition          <- composition level only
{partner_skill}, {partner_source_task}
## Grounded records
{grounded_records}   (Retail: candidate users, orders, and products;
                      Telecom: fault catalogue and environment calls)
## Your task
{level_instruction}
## Avoid duplicating these earlier variants
```

```
{earlier_variants}
## Required output
Return one JSON object with the user's goal, known and unknown information,
behavioral instructions, reference tool calls, success assertions, and
reward basis (Telecom adds fault-injection and environment-assertion calls).
Rules: every identifier must appear in the grounded records; user-facing
fields are written in the second person and never name tools; the user must
be identifiable; there is exactly one correct course of action; the
reference calls include every database write; the task is solvable under
the policy; a refusal is correct only where the policy forbids the request;
and the request is the smallest natural one for the level.
```

Probe synthesis: level instructions.

```
[Paraphrase]
Repeat the source task; keep the same user, record, required steps, and
success conditions, and change only surface wording.

[Substitution]
Replace only incidental entity values (e.g., {entity_kinds}) with a grounded,
isomorphic alternative. Form an entity mapping first, then apply it while
preserving the user's intent, dialogue policy, if-then replies, the
applicable policy branch, the reference-action structure, and the success
conditions.

[Composition]
Write a natural task in which the memory under validation and a second
memory both apply as written. Each memory must solve a different necessary
part of the request, so that removing either one breaks a different success
condition. Keep one coherent objective and a short solution path.

[Non-interference]
Write a simple, ordinary request from the same domain for which the memory's
procedure is not needed, with enough surface similarity to make memory
selection non-trivial. Applying the memory should cause a clear error; do
not reveal the test in the user-facing fields.
```

D.4 Probe Validation

Executability is checked deterministically (every referenced entity must exist, and the reference actions must run on a freshly initialized environment), so it has no prompt. Fidelity and relevance are judged by an LLM, which receives the level and the task(s) as JSON. Paraphrase probes are checked for executability only, substitution and irrelevant distractor tasks for all three criteria, and composition probes for executability and relevance.

Fidelity judge.

```
Judge whether the synthesized {domain} task made the intended kind of change
from the source task. Ignore wording, order, and extra clarifying detail.
  substitution: same request type and same policy branch, with different
                entities; fail if the request type changes, an errand is
                added, or the policy outcome flips.
  irrelevant distractor:  a different kind of request in the same domain; fail if it
                is the same request with entities swapped, with the outcome
                flipped, or merely paraphrased.
Input: {type, source_task, synthesized_task}
Return JSON: {"passed": true|false, "reason": "..."}
```

Relevance judge.

```
Judge whether the synthesized {domain} task is a valid item of the stated
type for the given memory. "Passed" means the item is valid for its type,
not that the memory applies.
```

```
  substitution: the memory's trigger and procedure still apply after the
                entity swap.
  composition:  both memories' procedures are necessary, and each covers a
                different part of the request.
  irrelevant distractor:  the correct procedure differs from the memory's, so using the
                memory would be the wrong move.
Input: {type, memories, synthesized_task}
Return JSON: {"passed": true|false, "reason": "..."}
```

D.5 Skill Refinement

A skill that fails the substitution or composition level is revised once, using a failed probe from that level and, when available, a successful one.

Skill refinement.

```
[System]
You revise an agent's memory item so that it keeps working beyond the task
it was written from. Return JSON only.

[User]
The memory below was extracted from one customer-service episode. It was then
tested on {level_description}, and the agent still failed.
## Memory under revision
{skill}
## The task where the agent failed
{failed_task_goal}
{failed_trajectory}
## A task where the agent succeeded with this memory      <- if available
{successful_task_goal}
{successful_trajectory}
## Your task
Rewrite the memory so an agent facing that kind of task would succeed. Keep
what was already right and fix what fell short (e.g., a trigger stated too
narrowly, a missing step, or advice that should be an instruction). Keep it
a single, general memory item without episode-specific details.
Return JSON: {"title", "description", "content", "what_changed"}
```

D.6 Random Task Augmentation

The task-augmentation baseline synthesizes variants from the source task alone, without any skill. It reuses the probe-synthesis system prompt and output format, and replaces the memory-specific sections with the following.

Source-only variant synthesis.

```
## Source task
Treat this task as the sole semantic template.
{source_task}
## Grounded records
{grounded_records}
## Your task
{substitution_instruction}
  (Telecom: keep the reported issue and all injected faults, and move the
   task to the assigned line and persona: {target_line}, {target_persona}.)
Generate one executable variation from the source task.
## Avoid duplicating these earlier variants
{earlier_variants}
```