SkillForge: Learning from Success–Failure Boundaries for Consistent Long-Horizon Agents

Anonymous ACL submission

Abstract

Long-horizon LLM agents often fail from procedural errors such as forgetting constraints, misordering tool calls, or stopping prematurely. Because these slips are unpredictable, the same agent can succeed on one run and fail on an identical task later. The resulting execution trajectories contain rich signals about which procedural choices lead to success or failure, yet existing approaches that extract reusable guidance from agent trajectories either introduce noise through imprecise distillation or depend on external retrievers for deployment. We propose SkillForge, a training-free framework that rethinks both the construction and deployment of procedural guidance: it pairs successful and failed attempts on the same task to identify critical procedural choices, consolidates and filters candidate guidance to remove low-quality entries, and loads relevant guidance on demand at inference time only when the agent's current context signals its relevance. Across agent benchmarks including τ^2-Bench and Terminal-Bench, SkillForge improves repeated-run consistency, reducing the consistency gap by up to 37% and lowering total token usage by up to 23.6% compared to baselines without degrading single-trial performance. These results suggest a practical path toward distilling high-quality procedural guidance from past trajectories for more consistent and efficient long-horizon agents.[1]

1 Introduction

Large language model (LLM) agents have demonstrated impressive capabilities on tasks requiring planning, tool usage, and extended interaction with external environments (Shinn et al., 2023; Wang et al., 2023; Yao et al., 2023; Wei et al., 2022). Many practical tasks are *long-horizon*: resolving a customer-service request may require verifying account status, checking policy eligibility, and coordinating across multiple tools over multiple steps (Ye et al., 2026b). On such tasks, model behavior remains unreliable. Repeated runs on the same or a nearly identical instance regularly produce different outcomes (Gan et al., 2025), as small procedural deviations such as forgetting a constraint, misordering tool calls, or terminating prematurely compound over the extended action sequence (Wan et al., 2025a), making *consistency* the dominant practical concern (Wang et al., 2024; Zhou et al., 2025).

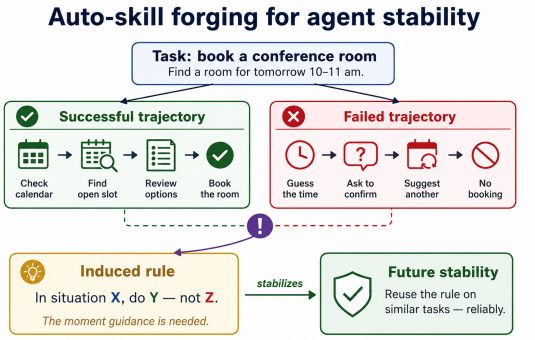

Figure 1: **Motivation.** Comparing successful and failed trajectories on the same task reveals a decisive divergence point; the rule induced there stabilizes the agent on future similar tasks.

A natural response is to improve the context supplied to the agent at inference time (Snell et al., 2025) and learn from past mistakes (Hu et al., 2026). Prior work has explored episodic memory and verbal self-feedback (Shinn et al., 2023; Park et al., 2023; Packer et al., 2023), persistent skill libraries (Zheng et al., 2025; Yang et al., 2026), and automatically induced guidelines (Fu et al., 2024; Chen et al., 2024; Li et al., 2026). But key limitations remain: retrieved guidance is often noisy or overly detailed, diluting the agent's reasoning with irrelevant context

[1]Code is available at https://anonymous.4open.science/r/adk_agentskill-DD64.

and wasting tokens (Zhao et al., 2024; Xiong et al., 2025); high-quality guidance typically requires costly human curation; and automatically generated skills tend to be too generic to transfer reliably across tasks (Li et al., 2026). This raises a key question: **how can we automatically distill high-quality guidance from past trajectories and deploy it efficiently at inference time, without human curation or external retrievers?**

A key insight to answer this question comes from recent work on difficulty-aware learning (Gao et al., 2026; Wang et al., 2025): **the richest signal for procedural guidance lies at the decision boundary between success and failure.** Tasks that are too easy (all runs pass) or too hard (all runs fail) provide little signal about which procedural choices matter. The most informative cases are tasks of intermediate difficulty where the agent sometimes succeeds and sometimes fails, as these mixed-outcome cases expose where procedural choices diverge. By pairing successful and failed attempts on such tasks, we obtain *contrastive trajectory evidence*: rather than separately summarizing what went right or wrong, this evidence isolates decisive procedural differences of the form "in situation X, do Y rather than Z.", as illustrated in Fig. 1.

We thus propose **SkillForge**, a training-free framework that distills reusable procedural guidance from past trajectories. SkillForge operationalizes this idea in four stages: **(1) Collect** *contrastive* trajectories—executions of the same task where the agent sometimes succeeds and sometimes fails—providing direct evidence about which procedural choices are decisive; **(2) Extract** candidate guidance by prompting a generator LLM to contrast procedural differences between successful and failed runs; **(3) Validate** each candidate on held-out tasks, discarding those that fail to improve pass rate or cause net-negative effects; and **(4) Retrieve** on demand at test time, exposing only lightweight metadata by default and loading full skill content through an explicit tool call. This design keeps the default input small, preserves agent autonomy, and surfaces detailed procedural support only when the trajectory makes its relevance clear.

We evaluate SkillForge on τ^2-Bench (Barres et al., 2025) and Terminal-Bench (Merrill et al., 2026). SkillForge improves average task performance and remains more consistent as the number of attempts k grows. Our analyses further reveal that contrastive grounding and validation filtering are both essential for reliable skill quality, and that on-demand loading outperforms full-prompt injection, as prematurely exposing guidance distracts the agent rather than helping it. In sum, our contributions are: **(1)** a **contrastive extraction principle** that pairs successful and failed trajectories on mixed-outcome tasks to isolate decisive procedural choices, substantially outperforming guidance derived from a single outcome polarity or from task descriptions alone; **(2) SkillForge**, a training-free framework that operationalizes this principle through validated skill construction and hierarchical on-demand retrieval, reducing the consistency gap by up to 37% and token usage by up to 23.6% over existing procedure-building baselines; and **(3)** an **in-depth analysis** establishing the complementary roles of success and failure trajectories, the scaling effect of trajectory budget on skill quality, and the efficiency advantages, along with practical deployment guidelines and detailed case studies.

2 Method: SkillForge

This section presents the core design of **SkillForge**. As illustrated in Fig.2, we first collect mixed-outcome executions on the same tasks (§2.1), then extract and refine candidate skills by contrasting successful and failed trajectories (§2.2). Candidates are subsequently consolidated and filtered to retain only skills that reliably improve performance (§2.2). The retained skills are then exposed to the agent on demand at inference time (§2.3).

Terminology. *Consistency* is the degree to which an agent produces the same correct outcome across independent runs on the same task, measured via Pass^k (see §3.1). *Procedural guidance* refers to reusable, actionable knowledge extracted from past trajectories, such as decision rules, preferred tool-call orderings, and error-recovery patterns, which steers the agent toward correct behavior; we use *experience* exclusively to refer to raw trajectories, not to derived artifacts. A *skill* is the concrete format in which procedural guidance is packaged

for storage and retrieval (Anthropic, 2025; Li et al., 2026): each skill contains a name, a short description for lightweight metadata matching, and a body specifying trigger conditions, a key procedure, common failure variants, and reconciliation rules.

2.1 Trajectory Collection

We partition the task set $\mathcal{T}$ into equal-sized disjoint training, validation, and test splits ($\mathcal{T}_{\text{train}}$, $\mathcal{T}_{\text{val}}$, $\mathcal{T}_{\text{test}}$) to prevent data leakage: skills are induced from $\mathcal{T}_{\text{train}}$, filtered on $\mathcal{T}_{\text{val}}$, and evaluated on $\mathcal{T}_{\text{test}}$. For each task $t \in \mathcal{T}_{\text{train}}$, we execute an agent under K different configurations $\{c_1, c_2, \ldots, c_K\}$, producing a set of trajectories $\{\tau_t^{c_1}, \tau_t^{c_2}, \ldots, \tau_t^{c_K}\}$. Each trajectory records the sequence of observations, actions, and a binary outcome $r_t^c \in \{0, 1\}$ indicating whether configuration c failed or succeeded on task t. Here, a *configuration* refers to a specific choice of agent hyperparameters, such as sampling temperature and reasoning effort (thinking budget), while holding the backbone model and environment fixed.[2]

The key requirement is to produce *mixed outcomes* on a meaningful subset of tasks, providing both successful and failed trajectories for skill extraction. These paired sets need not be balanced, so a single success among many failures (or vice versa) is sufficient. We define the set of *contrastive tasks* as

$$\mathcal{T}_{\text{contrast}} = \{t \in \mathcal{T}_{\text{train}} \mid \exists\, c_i, c_j : r_t^{c_i} = 1 \wedge r_t^{c_j} = 0\}. \quad (1)$$

For each contrastive task t, we group trajectories into successful and failed sets:

$$\mathcal{S}_t = \{\tau_t^c \mid r_t^c = 1\}, \qquad \mathcal{F}_t = \{\tau_t^c \mid r_t^c = 0\}.$$

2.2 Contrastive Skill Extraction and Validation

Given the successful and failed trajectory sets for each task $t \in \mathcal{T}_{\text{contrast}}$, we prompt a generator LLM $\mathcal{G}$ to extract candidate skills by identifying the behavioural differences that distinguish successful attempts from failed ones. For each task t, we sample up to m successful and m failed trajectories from $\mathcal{S}_t$ and $\mathcal{F}_t$ respectively, and provide $\mathcal{G}$ with the task description d_t, the sampled trajectories with key decision points and observed failure modes, and structured extraction instructions (full prompt templates in Appendix C):

$$g_t = \mathcal{G}\big(d_t,\ \{\tau_1^+, \ldots, \tau_m^+\},\ \{\tau_1^-, \ldots, \tau_m^-\}\big). \quad (2)$$

Stage 1: LLM-based refinement. Poorly generated skills can be counterproductive by consuming context, creating conflicting instructions, or misleading the agent (Ye et al., 2026a; Zhang et al., 2026b). Based on prior work identifying criteria for high-quality skills (Li et al., 2026), we prompt $\mathcal{G}$ to evaluate and revise each candidate against four criteria: **actionability** (concrete, executable steps rather than vague advice), **scope** (targets a specific failure pattern rather than serving as a broad manual), **generality** (transfers beyond the source instance to similar tasks), and **conciseness** (remains within a target token budget). Each candidate receives a binary pass/fail judgment per criterion; those failing any criterion are revised, and those that remain non-general after revision are discarded.

Stage 2: Skill consolidation and update. After candidate skills $\{c_i\}_{i=1}^N$ are induced from contrastive pairs, we consolidate them through an incremental library update process. Each candidate is first **normalized** into a standard skill format with a name, description, use condition, optional do-not-use condition, procedural guidance, and failure signature. We then retrieve **similar existing skills** from the current library $\mathcal{L}$ based on overlap in task triggers, tool-use patterns, constraints, and failure modes. If an existing skill already covers the candidate's trigger, failure signature, and corrective procedure, the candidate is treated as **redundant** and only useful examples or sharper wording are merged. If it contributes a **distinct element**, such as a new task subtype, applicability boundary, tool-use sequence, or recurring failure mode, it is added as a new skill. Conflicts with existing skills are resolved before insertion by narrowing the candidate's scope, adding a do-not-use condition, or rewriting the procedure so that the skills apply to different situations. This process favors **specific, actionable skills** over broad reminders, preserving runtime selectivity because the agent initially

[2] We vary decoding configurations (see Appendix B) to increase the yield of mixed-outcome tasks; Fig. 15 in Appendix illustrates a resulting divergences.

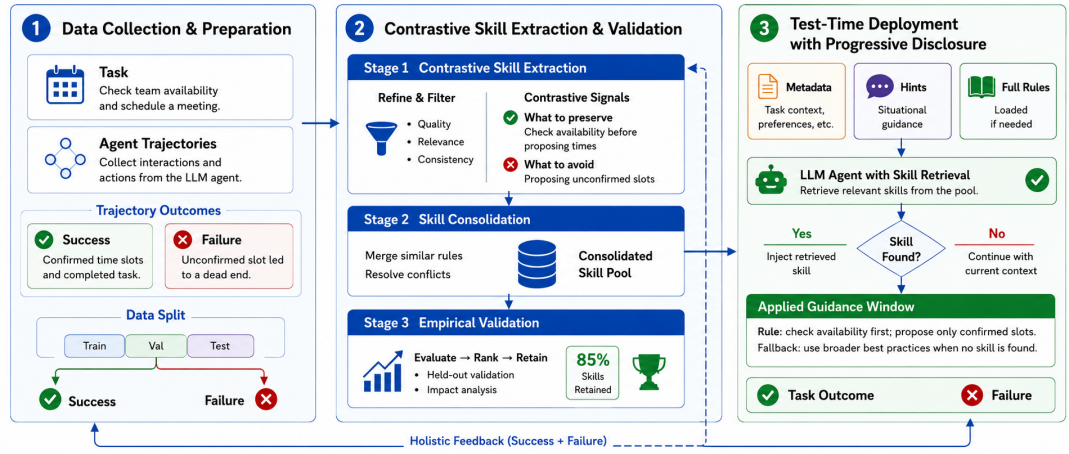

Figure 2: **Detailed workflow of SkillForge.** Starting from mixed-outcome trajectories, SkillForge prompts an LLM to extract procedural guidance by contrasting successful and failed runs. Guidance skills are then consolidated and validated. At test time, the agent reads only metadata and retrieves full skill content on demand only when the current context matches a relevant trigger condition.

decides what to load from skill metadata alone. A specific example of workflow is presented in Appendix D.1:

Stage 3: Empirical validation. To further reduce the variance, we evaluate each $s \in \mathcal{C}$ on the held-out validation split $\mathcal{T}_{\text{val}}$ (§2.1), comparing a baseline with no skill against a skill-augmented condition in which s is available for on-demand loading (§2.3). Let $r_t^{\emptyset}$ and r_t^{s} denote outcomes on task t without and with s, and define the per-task impact as:

$$\delta_t^{[s]} = r_t^{s} - r_t^{\emptyset}, \quad \delta_t^{[s]} \in \{-1, 0, +1\}. \tag{3}$$

A candidate s is retained if and only if it yields a strictly positive average impact across the tasks on which it is loaded,

$$\bar{\delta}^{[s]} = \frac{1}{|\mathcal{T}_{\text{val}}^{[s]}|} \sum_{t \in \mathcal{T}_{\text{val}}^{[s]}} \delta_t^{[s]} > 0, \tag{4}$$

where $\mathcal{T}_{\text{val}}^{[s]} \subseteq \mathcal{T}_{\text{val}}$ is the subset of tasks on which the agent chose to load s. The final retained set is $\mathcal{C}^* = \{s \in \mathcal{C} \mid \bar{\delta}^{[s]} > 0\}$.

2.3 Hierarchical On-Demand Retrieval

Preloading the full skill repository into the prompt wastes context and can hurt both reward and consistency (Fu et al., 2024). The failure mode is not merely token bloat: when many skills are exposed at once, surface keyword overlap between the user's request and unrelated skills can cause the agent to load and follow the wrong guidance, violating policy in ways that the no-skill baseline would have avoided (Fig. 16). SkillForge instead organises each skill into layers of increasing depth and lets the agent descend only as far as the current task requires. The shallowest layer holds only metadata (a short name and description), intermediate layers expose decision rules or diagnostic patterns, and terminal layers contain the heaviest resources such as code templates or step-by-step repair procedures. Each non-terminal layer specifies a *descent condition*, a natural-language description of when to load the next layer (e.g., "retrieve the repair template once the root cause is confirmed"). The number of layers varies by domain: latency-sensitive tasks typically need only one or two lightweight layers, while precision-sensitive tasks benefit from deeper hierarchies (see §B.6).

At episode start, only the metadata layer of each retained skill is appended to the system prompt; deeper layers reside as files in a dedicated skill directory. We expose a single tool, `load_skill(skill_name)`, which injects the next unobserved layer into the working context. At any turn, the agent decides whether to invoke this tool based on the context, including the most recently observed layer. No embedding index, vector database, or learned retriever is involved: the LLM's own judgment of whether the current context matches a descent condition is the sole gating mechanism.

3 Experiments

This section first describes our experimental setup, including benchmarks, models, baselines, and evaluation metrics (§3.1). We then present the main results, focusing on three questions: whether SkillForge improves task effectiveness, whether it improves repeated-run consistency, and whether these gains come with favorable efficiency trade-offs (§3.2), and finally present case analyses and discussions on practical usage scenarios (§3.3).

3.1 Experimental Setup

Benchmarks. We evaluate SkillForge on three primary benchmarks: **τ^2-Bench Airline** and **τ^2-Bench Retail**, which evaluate policy-constrained customer-service agents over multi-step tool-call dialogues; and **Terminal-Bench**, which evaluates software-engineering agents on fault-diagnosis and repair tasks requiring long action sequences. For the τ^2-Bench domains, we use the corrected task set from τ^2 (Barres et al., 2025), which resolves annotation errors and impossible constraints present in the original release (Yao et al., 2025). We omit the newly introduced Telecom domain, as new models like Gemini 3.0 Flash already saturate it (above 90% accuracy) and its 2,285 tasks make evaluation expensive. We additionally report results on **SkillsBench** in a supplementary analysis of proxy supervision (§3.3).

Models. Our main experiments use **Gemini 2.5 Flash** and **Gemini 3.0 Flash-Preview** (Gemini Team, Google, 2026), and **GPT-5-nano-20250807** (OpenAI, 2026) in selected experiments. We prioritize these lightweight models to ensure reproducibility and broad accessibility. Notably, Gemini 3.0 Flash already approaches state-of-the-art agentic performance at substantially lower cost than frontier models, making it a strong and practical backbone; the consistency gaps we observe even with this capable model underscore that reliability remains an open challenge regardless of raw capability. We expect SkillForge to remain beneficial for stronger models, since consistency failures stem from procedural variance rather than capability limitations, though the absolute gap may narrow. All ablations and diagnostic analyses use Gemini 3.0 Flash.

Baselines. We compare three methods: **No Skill**, **Self-Generated Skill** (skills generated from task descriptions without trajectory evidence, using the SkillsBench implementation (Li et al., 2026)), and **SkillForge** (our contrastively extracted, validation-filtered skills). We additionally include **Reasoning Bank** (Ouyang et al., 2026) in the ablation study (Fig.4) to test the overall design of our framework. While both our and their methods learn from past trajectories to improve future agent behavior, Reasoning Bank differs in how guidance is retrieved at inference time: it relies on external embedding-based retrievers to select relevant memories, whereas SkillForge exposes skills via metadata matching and an explicit bash grep-based tool call to retrieve the folders at pre-refined paths. Because the retrieval mechanisms are not directly comparable, we isolate their memory-generation and consolidation stages (which update their memory banks sequentially from individual trajectories) and keep our retrieval mechanism for both for a controlled comparison.

Metrics. We report single-trial performance via Pass@1 and repeated-run consistency via Pass^k. For each task t, with c_t successes over n independent executions, we follow τ-bench (Yao et al., 2025) and compute $\text{Pass}^k(t) = \binom{c_t}{k}/\binom{n}{k}$, averaged over tasks. In our main experiments, $n = 5$, so Pass^5 requires all five executions of a task to succeed. We further report *relative consistency*, $\text{RC} = \text{Pass}^5/\text{Pass@1} \in [0, 1]$, where $\text{Pass@1} = \text{Pass}^1$ and $\text{RC} = 1$ indicates perfect consistency. More setup details are in Appendix B.1.

3.2 Main Results

Overall performance and consistency. Table 1 reports results across three model families. On single-trial performance (Pass@1), SkillForge is broadly superior to the baselines, with consistent gains in all settings on different benchmarks and base models. The primary benefit lies in *consistency*: RC improves in every model–benchmark pair, with absolute gains ranging from 2.7 to 13.1 percentage points. The largest improvement is on τ^2-Bench Airline with Gemini 3.0 Flash (+13.1pp, 0.552→0.683, a 24% relative improvement).

Test-time scaling for skill construction. The memory-construction stage exposes a useful scaling rule that is consistent with prior

Model	Benchmark	No Skill		Self-Gen. Skill		SkillForge (Ours)	
		Pass@1	RC	Pass@1	RC	Pass@1	RC
Gemini 2.5 Flash	τ^2-Bench Airline	0.575	0.480	0.550	0.480	**0.600**	**0.567**
	τ^2-Bench Retail	0.629	0.455	0.621	0.428	**0.635**	**0.482**
	Terminal-Bench	0.264	0.413	0.261	0.395	**0.296**	**0.524**
Gemini 3.0 Flash	τ^2-Bench Airline	0.668	0.552	0.670	0.537	**0.701**	**0.683**
	τ^2-Bench Retail	0.761	0.599	0.742	0.597	**0.778**	**0.701**
	Terminal-Bench	0.512	0.670	0.528	0.667	**0.535**	**0.791**
GPT-5 Nano	τ^2-Bench Airline	0.435	0.389	0.440	0.359	**0.455**	**0.443**
	τ^2-Bench Retail	0.643	0.443	0.623	0.425	**0.645**	**0.519**
	Terminal-Bench	0.256	0.450	0.248	0.425	**0.273**	**0.525**

Table 1: Performance and consistency across three models. Pass@1 measures single-trial success, and RC = Pass^5/Pass@1 measures repeated-run consistency (higher is better). **Bold** denotes the best method for each comparison. SkillForge mainly improves RC, suggesting stronger stability across repeated executions. See Appendices B.4 and B.5 for statistics on significance, skill retrieval rate, and token usage.

work on dynamic sampling that balances exploration and exploitation in information gathering (Wan et al., 2025b). As shown in Fig.3, increasing sampling budget from 4 to 12 trajectories consistently improves repeated-run consistency over the no-skill baseline, because additional trajectories help find more intermediate-difficulty tasks, strengthening the contrastive signal available for extraction.

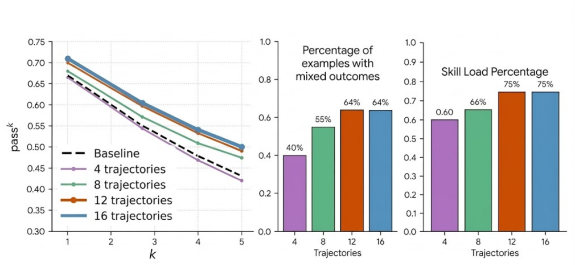

Figure 3: **Test-time scaling in memory construction.** Increasing the trajectory sampling budget improves consistency up to 12 trajectories. Beyond this, 16 trajectories yields no significant gain, as the proportion of mixed-outcome examples (middle) and the ratio of tasks where skill is loaded (right) both plateau, indicating that additional samples neither surface new intermediate difficulty tasks nor produce more helpful skills.

Beyond this, sampling with 16 trajectories performs comparably to 12 across all values of k. The middle and right panels of Fig.3 explain why: First, 12 and 16 trajectories yield the same proportion of mixed-outcome pairs, indicating that the additional samples no longer surface new intermediate difficulty tasks on which the agent behaves inconsistently. Second, the skill load percentage, the fraction of tasks in which the agent invokes at least one skill, also saturates at the same point (75% for both 12 and 16 trajectories), confirming that the extra trajectory budget does not produce skills that the agent finds relevant more often. Together, these results suggest that beyond a modest trajectory budget, additional samples yield diminishing contrastive signal and little additional skill utility. In practice, a moderate budget (here, 12 trajectories) captures most of the useful signal at reasonable sampling cost.

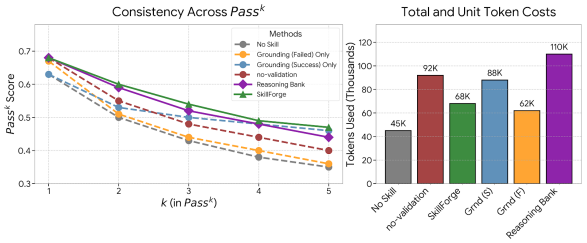

Figure 4: **Ablation Comparison of consistency and token efficiency across design variants.** SkillForge achieves the best overall trade-off, combining the strongest consistency and competitive task performance with the lowest token cost among the compared variants.

Ablation Studies. Fig. 4 breaks down the contributions of each SkillForge component along two axes: consistency across Pass^k (left) and total token cost (right). Among the trajectory-grounding variants, ***Grounding (Success) Only*** yields the most stable consistency curve, suggesting that successful trajectories provide the strongest signal for reusable behavior that transfers across runs; however, it does not improve over the no-skill baseline at $k = 1$ and incurs substantially higher token cost. ***Grounding (Failed) Only*** improves over the baseline at smaller k but degrades more quickly as k grows, indicating that failed trajectories contain useful error-avoidance cues yet are insufficient on their own to induce stable guidance.

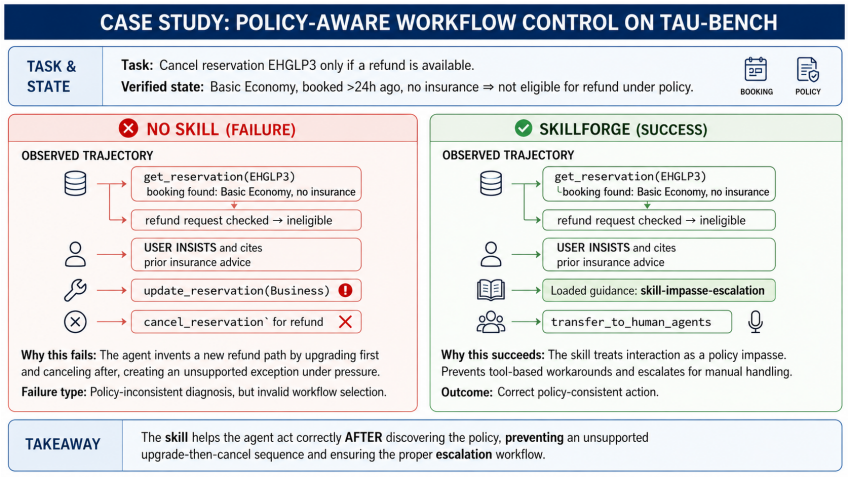

Figure 5: **Case study**: policy-aware workflow control under user pressure. Both agents correctly identify that reservation EHGLP3 is non-refundable, but the no-skill agent invents an invalid workflow, whereas SkillForge invokes the relevant procedural guidance and correctly addresses the issue.

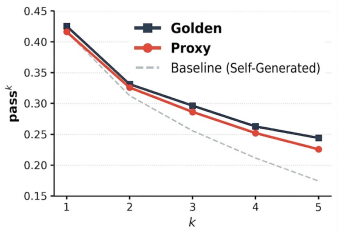

Figure 6: **Golden vs. LLM-judge proxy supervision for trajectory sampling**. Golden labels produce the strongest consistency, but proxy supervision remains a practical alternative, preserving the consistency gain across all values of k.

The contrast supports our design rationale: successful trajectories show *what to preserve*, failed trajectories show *what to avoid*, and their combination isolates the procedural differences that matter most. Removing the validation stage (***No-validation/filter***) matches SkillForge at $k = 1$ but its Pass^k drops sharply as k grows while token cost rises, confirming that unfiltered skills fail to sustain consistency and bloat context with low-utility guidance. Finally, ***Reasoning Bank*** (Ouyang et al., 2026), which sequentially updates memory on each trajectory without filtering, is outperformed by SkillForge for consistency and efficiency, with the gap widening at larger k. Together, these results indicate that contrastive grounding and validation filtering are complementary: grounding determines what guidance to generate, while filtering determines what guidance to keep. The prompt details are provided in Appendix C.

Efficiency trade-offs. SkillForge improves efficiency through both guidance quality and delivery. Fig. 4 (right) shows that lower-quality or poorly guidance can increase token usage while weakening performance. Skills extracted from successful or failed only trajectories without proper validation lack contrastive precision, which can mislead the agent and induce unproductive exploration or backtracking, as illustrated by the retail failure case in Appendix Fig. 14 and 16. Methods such as Reasoning Bank present a complementary limitation: because their guidance is not structured for progressive loading, they may expose all guidance upfront regardless of relevance, increasing context length without necessarily helping the current decision; SkillForge avoids both issues by using contrastive grounding and validation filtering to improve skill *content*, together with hierarchical on-demand loading to improve skill *delivery*. Consistent with this explanation, the delivery ablation in Appendix Table 3 shows that on-demand loading achieves the best reward, strongest Pass^k retention, and lowest token usage, reducing tokens by 16.7% over no-skill and 23.6% over full-prompt guidance.

3.3 Practical Analysis

Case Study: Workflow Control Under User Pressure. A recurring failure mode in τ^2-Bench is not misidentifying the correct policy, but abandoning it under sustained user pressure. Agents frequently recognize the policy-compliant action, then drift away from it when the user pushes back, citing prior advice, or reframes the request, producing unsupported workarounds that stitch together otherwise-valid tools into a sequence the policy does not sanction. SkillForge mitigates this class of failure by stabilizing *post-recognition decision-making*: once the agent has identified the applicable constraint, loading a relevant skill module anchors it to the sanctioned resolution path (e.g., escalation, refusal, or deferral) rather than improvising an alternative. The effect is less about improving *what* the agent knows and more about improving *whether it holds the line* when challenged. Fig. 5 illustrates this pat-

tern in a representative cancellation scenario: both the no-skill and SkillForge agents correctly determine that reservation `EHGLP3` is refund-ineligible, but only the SkillForge agent, by loading the `skill-impasse-escalation`, proceeds to `transfer_to_human_agents` instead of fabricating an upgrade-then-cancel workaround under user pressure. Additional case studies are provided in Appendix Figure 13.

Robustness to Approximate Supervision. Our main pipeline uses golden task outcomes to separate successful and failed trajectories, but such supervision is often unavailable in practice. To test whether approximate feedback is sufficient, we replace gold labels with an LLM judge and evaluate on SkillsBench (Li et al., 2026), a recent long-horizon benchmark designed to test skill usage in agents and unlikely to be contaminated in current models for fair evaluations on its labels. As shown in Fig.6, golden labels yield the strongest consistency, as expected, but proxy supervision remains a practical alternative. It consistently outperforms self-generated skills and preserves most of the consistency gain over the no-skill baseline. The proxy judge achieves 77% agreement with gold trajectory labels, comparable to agreement rates reported in prior work on LLM-based trajectory evaluation (Zheng et al., 2023). This suggests that moderate label noise does not substantially weaken the contrastive extraction stage.

4 Related Work

Experience Memory for Agent Self-Improvement. A central challenge for LLM agents is retaining and reusing past experience. Foundational approaches explore self-reflection and episodic memory retrieval (Shinn et al., 2023; Park et al., 2023; Packer et al., 2023; Wang et al., 2023). More recent work distills trajectories into structured artifacts such as lessons, workflows, and reasoning traces (Zhao et al., 2024; Wang et al., 2024; Ouyang et al., 2026; Suzgun et al., 2025), and explicitly manages abstraction boundaries across planning and execution (Ye et al., 2025; Zhou et al., 2025). However, these systems assume the stored experience is correct; recent studies show that incorrect or low-quality artifacts can degrade performance (Zhu et al., 2025; Li et al., 2026). Rather than filtering for correctness alone, curriculum-aware learning suggests that targeting informative, appropriately difficult examples yields more robust generalization (Wang et al., 2025; Qi et al., 2025). SkillForge builds on both observations: it selectively targets tasks where outcomes are mixed across runs and distills the resulting contrastive evidence into static, retrieval-free skill artifacts.

From Retrieval-Based Memory to Agent Skills. A key question in experience-based agent improvement is how stored knowledge is surfaced at inference time. Retrieval-augmented approaches query episodic memory banks via embedding-based similarity search (Lewis et al., 2020; Fan et al., 2024; Fang et al., 2025). An emerging alternative packages reusable procedures as declarative *skills* designed for progressive disclosure, loaded on-demand as targeted context when relevant (Zhou et al., 2026a; Zheng et al., 2025; Ye et al., 2026a; Ni et al., 2026; Zhang et al., 2026b,a; Lu et al., 2026), making artifacts architecture-agnostic, shareable, and efficient (Anthropic, 2025). However, high-quality skills remain mostly human-authored (Anthropic, 2026; Zhou et al., 2026b); self-generated, low quality skill can even hurt performance (Li et al., 2026). SkillForge bridges this gap by distilling contrastive evidence from past trajectories into curated declarative skills, improving procedural consistency without human curation or test-time retrieval.

5 Conclusion

We introduced **SkillForge**, a framework that distills reusable procedural guidance from contrastive past trajectories. By pairing successful and failed agent attempts, validating candidate skills, and loading retained guidance on demand, SkillForge identifies and deploys actionable procedural knowledge for reliable long-horizon agents. Across diverse agentic benchmarks, SkillForge improves repeated-run consistency without degrading performance in single-trial performance. These results show that contrastive grounding and selective deployment are effective ingredients for long-horizon agent reliability and learning from structured success–failure differences is a promising direction for automated, high-quality skill generation.

Limitations

SkillForge has several limitations that shape both its current scope and natural extensions.

Evaluation scope. Our experiments cover policy-constrained customer-service dialogue (τ^2-Bench Airline, Retail) and command-line software engineering (Terminal-Bench). These domains share a property critical to our analysis, but they do not exercise the full range of long-horizon agent settings. We do not evaluate on web agents, GUI agents, or open-ended generation tasks where success is harder to verify and where procedural patterns may be more diffuse. Extending the contrastive extraction principle to these settings, particularly to Deep-Research and GUI agents where mixed-outcome trajectories are common but reward signals are noisier, is a natural next step.

Failure modes addressed. SkillForge targets failures from recurring *procedural* errors: misordered tool calls, missed constraints, premature termination, or failure to reconcile conflicting state. Tasks dominated instead by missing factual knowledge, fundamentally hard reasoning gaps, or highly open-ended generation are less likely to benefit, since reusable procedural guidance cannot supply capabilities the base model lacks. The framework improves how agents *apply* their capabilities over long horizons rather than expanding those capabilities.

Retrieval bounded by model judgment. On-demand loading relies on the base model to recognize when stored guidance is relevant from metadata alone. Our per-skill diagnostics (Appendix B.5) show that invocations are usually appropriate when they fire, but a model that systematically under-loads or over-loads skills would dilute the benefit. Lightweight retrieval-assist mechanisms, e.g., trigger-match hints surfaced alongside metadata, could improve recall without reintroducing the cost of full embedding-based retrieval.

References

Anthropic. 2025. Agent skills specification. https://agentskills.io/specification. Accessed: 2026-03-21.

Anthropic. 2026. *Skill authoring best practices*. Anthropic PBC. Accessed: 2026-03-31.

Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. 2025. τ^2-bench: Evaluating conversational agents in a dual-control environment. *Preprint*, arXiv:2506.07982.

Minghao Chen, Yihang Li, Yanting Yang, Shiyu Yu, Binbin Lin, and Xiaofei He. 2024. Automanual: Constructing instruction manuals by LLM agents via interactive environmental learning. *Preprint*, arXiv:2405.16247.

Wenqi Fan, Yujuan Ding, Liangbo Ning, Shijie Wang, Hengyun Li, Dawei Yin, Tat-Seng Chua, and Qing Li. 2024. A survey on rag meeting llms: Towards retrieval-augmented large language models. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD '24, page 6491–6501, New York, NY, USA. Association for Computing Machinery.

Runnan Fang, Yuan Liang, Xiaobin Wang, Jialong Wu, Shuofei Qiao, Pengjun Xie, Fei Huang, Huajun Chen, and Ningyu Zhang. 2025. Memp: Exploring agent procedural memory. *Preprint*, arXiv:2508.06433.

Yao Fu, Dong-Ki Kim, Jaekyeom Kim, Sungryull Sohn, Lajanugen Logeswaran, Kyunghoon Bae, and Honglak Lee. 2024. Autoguide: Automated generation and selection of context-aware guidelines for large language model agents. *Preprint*, arXiv:2403.08978.

Zeyu Gan, Yun Liao, and Yong Liu. 2025. Rethinking external slow-thinking: From snowball errors to probability of correct reasoning. In *Forty-second International Conference on Machine Learning*.

Zhaolin Gao, Joongwon Kim, Wen Sun, Thorsten Joachims, Sid Wang, Richard Yuanzhe Pang, and Liang Tan. 2026. Prompt curriculum learning for efficient LLM post-training. In *The Fourteenth International Conference on Learning Representations*.

Gemini Team, Google. 2026. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. Technical Report.

Yuyang Hu, Shichun Liu, Yanwei Yue, Guibin Zhang, Boyang Liu, Fangyi Zhu, Jiahang Lin, Honglin Guo, Shihan Dou, Zhiheng Xi, Senjie Jin, Jiejun Tan, Yanbin Yin, Jiongnan Liu, Zeyu Zhang, Zhongxiang Sun, Yutao Zhu, Hao Sun, Boci Peng, and 28 others. 2026. Memory in the age of ai agents. *Preprint*, arXiv:2512.13564.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. In *Proceedings of the 34th International Conference*

on Neural Information Processing Systems, NIPS '20, Red Hook, NY, USA. Curran Associates Inc.

Xiangyi Li, Wenbo Chen, Yimin Liu, Shenghan Zheng, Xiaokun Chen, Yifeng He, Yubo Li, Bingran You, Haotian Shen, Jiankai Sun, Shuyi Wang, Binxu Li, Qunhong Zeng, Di Wang, Xuandong Zhao, Yuanli Wang, Roey Ben Chaim, Zonglin Di, Yipeng Gao, and 22 others. 2026. Skillsbench: Benchmarking how well agent skills work across diverse tasks. *Preprint*, arXiv:2602.12670.

Zhengxi Lu, Zhiyuan Yao, Jinyang Wu, Chengcheng Han, Qi Gu, Xunliang Cai, Weiming Lu, Jun Xiao, Yueting Zhuang, and Yongliang Shen. 2026. Skill0: In-context agentic reinforcement learning for skill internalization. *Preprint*, arXiv:2604.02268.

Mike A Merrill, Alexander Glenn Shaw, Nicholas Carlini, Boxuan Li, Harsh Raj, Ivan Bercovich, Lin Shi, Jeong Yeon Shin, Thomas Walshe, E. Kelly Buchanan, Junhong Shen, Guanghao Ye, Haowei Lin, Jason Poulos, Maoyu Wang, Jenia Jitsev, Marianna Nezhurina, Di Lu, Orfeas Menis Mastromichalakis, and 65 others. 2026. Terminal-bench: Benchmarking agents on hard, realistic tasks in command line interfaces. In *The Fourteenth International Conference on Learning Representations*.

Jingwei Ni, Yihao Liu, Xinpeng Liu, Yutao Sun, Mengyu Zhou, Pengyu Cheng, Dexin Wang, Xiaoxi Jiang, and Guanjun Jiang. 2026. Trace2skill: Distill trajectory-local lessons into transferable agent skills. *arXiv preprint arXiv:2603.25158*.

OpenAI. 2026. Openai gpt-5 system card. *Preprint*, arXiv:2601.03267.

Siru Ouyang, Jun Yan, I-Hung Hsu, Yanfei Chen, Ke Jiang, Zifeng Wang, Rujun Han, Long Le, Samira Daruki, Xiangru Tang, Vishy Tirumalashetty, George Lee, Mahsan Rofouei, Hangfei Lin, Jiawei Han, Chen-Yu Lee, and Tomas Pfister. 2026. Reasoningbank: Scaling agent self-evolving with reasoning memory. In *The Fourteenth International Conference on Learning Representations*.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. 2023. Memgpt: Towards LLMs as operating systems. *Preprint*, arXiv:2310.08560.

Joon Sung Park, Joseph C. O'Brien, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. 2023. Generative agents: Interactive simulacra of human behavior. *Preprint*, arXiv:2304.03442.

Zehan Qi, Xiao Liu, Iat Long Iong, Hanyu Lai, Xueqiao Sun, Jiadai Sun, Xinyue Yang, Yu Yang, Shuntian Yao, Wei Xu, Jie Tang, and Yuxiao Dong. 2025. WebRL: Training LLM web agents via self-evolving online curriculum reinforcement learning. In *The Thirteenth International Conference on Learning Representations*.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. *Preprint*, arXiv:2303.11366.

Charlie Victor Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. 2025. Scaling LLM test-time compute optimally can be more effective than scaling parameters for reasoning. In *The Thirteenth International Conference on Learning Representations*.

Mirac Suzgun, Mert Yuksekgonul, Federico Bianchi, Dan Jurafsky, and James Zou. 2025. Dynamic cheatsheet: Test-time learning with adaptive memory.

Guangya Wan, Mingyang Ling, Xiaoqi Ren, Rujun Han, Sheng Li, and Zizhao Zhang. 2025a. Compass: Enhancing agent long-horizon reasoning with evolving context. *Preprint*, arXiv:2510.08790.

Guangya Wan, Yuqi Wu, Jie Chen, and Sheng Li. 2025b. Reasoning aware self-consistency: Leveraging reasoning paths for efficient LLM sampling. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 3613–3635, Albuquerque, New Mexico. Association for Computational Linguistics.

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

Xinglin Wang, Shaoxiong Feng, Yiwei Li, Peiwen Yuan, Yueqi Zhang, Chuyi Tan, Boyuan Pan, Yao Hu, and Kan Li. 2025. Make every penny count: Difficulty-adaptive self-consistency for cost-efficient reasoning. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pages 6919–6932, Albuquerque, New Mexico. Association for Computational Linguistics.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. 2024. Agent workflow memory. *Preprint*, arXiv:2409.07429.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed H. Chi, Quoc V Le, and Denny Zhou. 2022. Chain of thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*.

Zidi Xiong, Yuping Lin, Wenya Xie, Pengfei He, Zirui Liu, Jiliang Tang, Himabindu Lakkaraju,

and Zhen Xiang. 2025. How memory management impacts LLM agents: An empirical study of experience-following behavior. *Preprint*, arXiv:2505.16067.

Yutao Yang, Junsong Li, Qianjun Pan, Bihao Zhan, Yuxuan Cai, Lin Du, Jie Zhou, Kai Chen, Qin Chen, Xin Li, Bo Zhang, and Liang He. 2026. Autoskill: Experience-driven lifelong learning via skill self-evolution. *Preprint*, arXiv:2603.01145.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik R Narasimhan. 2025. {τ}-bench: A benchmark for \underline{T}ool-\underline{A}gent-\underline{U}ser interaction in real-world domains. In *The Thirteenth International Conference on Learning Representations*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*.

Haoran Ye, Xuning He, Vincent Arak, Haonan Dong, and Guojie Song. 2026a. Meta context engineering via agentic skill evolution. *Preprint*, arXiv:2601.21557.

Rui Ye, Zhongwang Zhang, Kuan Li, Huifeng Yin, Zhengwei Tao, Yida Zhao, Liangcai Su, Liwen Zhang, Zile Qiao, Xinyu Wang, Yong Jiang, Pengjun Xie, Fei Huang, Siheng Chen, and Jingren Zhou. 2026b. Agentfold: Long-horizon web agents with proactive context folding. In *The Fourteenth International Conference on Learning Representations*.

Shicheng Ye, Chao Yu, Kaiqiang Ke, Chengdong Xu, and Yinqi Wei. 2025. H^2R: Hierarchical hindsight reflection for multi-task LLM agents. *Preprint*, arXiv:2509.12810.

Hanrong Zhang, Shicheng Fan, Henry Peng Zou, Yankai Chen, Zhenting Wang, Jiayu Zhou, Chengze Li, Wei-Chieh Huang, Yifei Yao, Kening Zheng, Xue Liu, Xiaoxiao Li, and Philip S. Yu. 2026a. Coevoskills: Self-evolving agent skills via co-evolutionary verification. *Preprint*, arXiv:2604.01687.

Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, Urmish Thakker, James Zou, and Kunle Olukotun. 2026b. Agentic context engineering: Learning comprehensive contexts for self-improving language models. In *The Fourteenth International Conference on Learning Representations*.

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

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

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. 2023. Judging LLM-as-a-judge with MT-bench and chatbot arena. In *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

Chenyu Zhou, Huacan Chai, Wenteng Chen, Zihan Guo, Rong Shan, Yuanyi Song, Tianyi Xu, Yingxuan Yang, Aofan Yu, Weiming Zhang, Congming Zheng, Jiachen Zhu, Zeyu Zheng, Zhuosheng Zhang, Xingyu Lou, Changwang Zhang, Zhihui Fu, Jun Wang, Weiwen Liu, and 2 others. 2026a. Externalization in llm agents: A unified review of memory, skills, protocols and harness engineering. *Preprint*, arXiv:2604.08224.

Yingli Zhou, Wang Shu, Yaodong Su, Wenchuan Du, Yixiang Fang, and Xuemin Lin. 2026b. A comprehensive survey on agent skills: Taxonomy, techniques, and applications. *Preprint*, arXiv:2605.07358.

Zijian Zhou, Ao Qu, Zhaoxuan Wu, Sunghwan Kim, Alok Prakash, Daniela Rus, Jinhua Zhao, Bryan Kian Hsiang Low, and Paul Pu Liang. 2025. MEM1: Learning to synergize memory and reasoning for efficient long-horizon agents. *Preprint*, arXiv:2506.15841.

Kunlun Zhu, Zijia Liu, Bingxuan Li, Muxin Tian, Yingxuan Yang, Jiaxun Zhang, Pengrui Han, Qipeng Xie, Fuyang Cui, Weijia Zhang, Xiaoteng Ma, Xiaodong Yu, Gowtham Ramesh, Jialian Wu, Zicheng Liu, Pan Lu, James Zou, and Jiaxuan You. 2025. Where llm agents fail and how they can learn from failures. *Preprint*, arXiv:2509.25370.

A Use of AI Assistance

We used large language models in two limited, non-scientific capacities during the preparation of this manuscript:

(1) *Code review*: identifying bugs, style inconsistencies, and potential edge cases in our experiment and evaluation scripts; and

(2) *Grammar and prose review*: proofreading draft text for grammatical errors, awkward phrasing, and typographical issues.

No AI tool was used for scientific ideation, experimental design, result interpretation, or the writing of substantive claims. All conceptual contributions, methodological decisions, analyses, and conclusions are solely the work of the authors. We believe this disclosure aligns with the transparency expectations of the research community.

B Implementation Details

B.1 Infrastructure and Evaluation Settings

Infrastructure. All experiments, skill induction runs, and agent evaluations were orchestrated using Google's Agent Development Kit (ADK), with the Vertex AI API used to access the Gemini model family. To evaluate on standardized environments, we adapted the original codebases of τ^2-Bench and Terminal-Bench to be natively compatible with the ADK framework.

Data splits. For τ^2-Bench, we follow the official train/test split provided by the benchmark. Because no official validation split is available, we randomly partition the training set into two equal halves: one for skill induction ($\mathcal{T}_{\text{contrast}}$) and one for empirical validation ($\mathcal{T}_{\text{val}}$), while keeping the official test set untouched for final evaluation. We apply the same procedure to Terminal-Bench. In all cases, $\mathcal{T}_{\text{test}}$ is never used during skill construction or validation. Note that we don't use any partial subset but full data for each of them.

Trajectory collection. To induce mixed-outcome trajectories for skill construction, we run the same task under a small set of inference-time configurations. In particular, we vary the decoding temperature from 0.5 to 1.0 and also vary the reasoning budget depending on the backbone model. For Gemini 2.5 Flash, we use reasoning budgets of 8192 tokens and `auto`; for Gemini 3.0 Flash, we use `low`, `medium`, and `high` thinking levels. These variations are used only during trajectory collection to increase the number of mixed-outcome tasks and expose diverse procedural behaviors on the same task. Configuration-specific artifacts are mitigated by the held-out validation stage, which retains only skills with positive downstream impact.

Evaluation settings. Unless otherwise stated, we evaluate agents at decoding temperature 1.0 and run 5 trials per task for consistency estimation. Exact model names, trial counts, and concurrency settings are reported alongside each experiment table, rather than assuming a single global configuration across all benchmarks.

Pipeline defaults. Skill generation uses contrastive mode by default, with up to 6 trajectories per outcome group (successful and failed) for contrastive drafting and up to 4 trajectories for single-outcome drafting. The generator LLM is run at low temperature (0.25 for drafting and 0.15 for batch refinement) with structured JSON outputs to ensure parseable skill candidates. The retained skill pool is capped at 50 skills, with up to 20 existing skills shown to the generator during drafting to reduce redundancy. Policy text is truncated at 6,000 characters to stay within context limits. Individual experiment scripts may override these defaults where noted; full configuration details are provided in the accompanying code repository.

Skills Generated. We summarize the final skill sets used by our method after generation, refinement, and filtering. For the airline domain, the final skill set used in our experiments contains 7 skills: `airline-baggage-allowance-logic`, `airline-cancellation-refund-validation`, `airline-disruption-compensation-and-disputes`, `airline-itinerary-search-optimization`, `airline-mirror-booking-workflow`, `airline-modification-financial-validation`, and `airline-rebooking-workflow`. These skills correspond to recurring airline-agent decision points, such as refund validation, rebooking, itinerary search, payment and fare-difference checking, and disruption-related compensation handling.

For coding and terminal tasks, the final Terminal-Bench skill set is stored under `Memory/Procedural_Memory/Outputs/tb_w8_skills`. It contains 5 skills: `glibc-compat-legacy-c`, `secret-sanitization-workflow`, `workflow-security-audit`, `qemu-initramfs-media-integration`, and `cli-tool-feature-extension`. These skills encode reusable software-engineering procedures such

as compatibility fixes, security auditing, secret removal, command-line feature extension, and QEMU/initramfs integration.

For tau2-retail, the final refined retail library contains 25 skills in `skills/tau2_retail_cheatsheets_v2`. These skills capture common retail-agent failure patterns including multi-intent clarification, replacement-item confirmation, transaction-scope confirmation, irreversible-action confirmation, and information verification before transactional updates. As a concrete example, the skill `address-multi-part-requests` is available at: https://anonymous.4open.science/r/adk_agentskill-DD64/skills/tau2_retail_cheatsheets_v2/address-multi-part-requests/SKILL.md. One example of full skill card is shown below.

```
---
name: address-multi-part-requests
description: Ensures all distinct components
  of a user's multi-part request are
  acknowledged and addressed, preventing
  partial fulfillment.
metadata:
  benchmark: tau2_retail_cheatsheets_v2
  source: /data/Research_dir/Memory/...
    Procedural_Memory/Outputs/...
    tau2_pairable_skill_tool.json
---

## Layer 1: Metadata & Description
- Description: Ensures all distinct components
  of a user's multi-part request are
  acknowledged and addressed, preventing
  partial fulfillment.

## Layer 2: Guidelines
### General Guidelines
- Always parse the user's initial statement
  for all explicit and implicit intents.
- Confirm understanding of all parts of a
  multi-part request.
- Do not assume prioritization of one request
  type over another.
- Seek user clarification on which part to
  address first if multiple intents are present.
- Avoid proceeding with any part of a request
  until all components are acknowledged.

### Specific Guidelines
- If a user asks questions AND requests an
  action, acknowledge both and ask, 'I
  understand you have questions about products
  and also want to initiate returns. Which
  would you like to tackle first?'
- Before initiating any transactional action
  (like returns), explicitly confirm the
  specific items or details involved.
- If an informational query is present, ensure
  it is addressed or explicitly deferred with
  user consent.
- Avoid immediately executing a transactional
  request if an informational query is also
  present.
- Use a summary statement to confirm all
  understood parts of the request before
  proceeding with any action.

## Layer 3: Context
- Problem Pattern: Failing to address all
  distinct components of a multi-part user
  request.
- State Signature: User's initial statement
  contains multiple, distinct requests or
  intents (e.g., informational query and
  transactional action).
- Bypass Strategy:
  1. Identify all distinct intents within
     the user's utterance.
  2. Acknowledge each identified intent to
     the user.
  3. Explicitly ask the user which component
     they would like to address first.
  4. Before proceeding with any transactional
     component, confirm specific details
     related to that transaction.
```

For mode details, please see our code attached in the supplemental materials.

B.2 τ^2-Bench Revisions

For our τ^2-Bench experiments, we used a revised benchmark configuration based on the public `tau2-bench-revised` release.[3] According to the released revision notes, this resource provides two benchmark-side changes:

1. **Airline domain task fixes**, documented in the airline task revision file.
2. **Retail user simulation prompt improvements**, intended to reduce premature task termination.

We used these revised benchmark files as replacements for the corresponding original τ^2-Bench resources.

B.3 Structured Skill Format Across Benchmarks

All skills originate from a primary `SKILL.md` file that includes a standardized YAML frontmatter for dynamic metadata extraction:

```
–
name: <skill-name>
description: <when to load this skill
and what it helps with>
–
```

For **Tau-Bench** style conversational skills, the guidance is typically self-contained within

[3] https://github.com/AGI-Eval-Official/tau2-bench-revised

this single document. The body focuses on reusable tool-call procedures and policy-sensitive decision rules, organized into sections such as:

```
## When to Use
## Key Sequence Pattern
## Common Failure Variants
## Preferred Procedure
## Capability Reconciliation
```

For **SWE-Bench and Terminal-Bench** style coding skills, the organization is inherently *hierarchical* to support progressive disclosure. The root `SKILL.md` acts as a high-level entry point or playbook:

```
## Context
## Why This Is Challenging
## Implementation Strategy
## Code Patterns
## Debugging Approach
## Common Mistakes
## Verification
```

Crucially, to prevent overwhelming the agent's context window, this root document does not contain every dense script or code snippet. Instead, sections like `## Debugging Approach` or `## Code Patterns` provide high-level summaries alongside explicit file paths pointing to deeper, specialized sub-documents. If the agent determines that a particular debugging strategy is relevant to its current context, it can issue subsequent tool calls to load those targeted, longer code patterns and verification techniques. This allows the agent to dynamically drill down into complex procedures only when necessary, maximizing both context space and token efficiency.

B.4 Statistical Analysis

Significance Calculation. Because consistency is reported as a curve over Pass^1 (e.g. Pass @ 1)–Pass^5, we quantify uncertainty at two levels. For each plotted Pass^k value, we report a 95% confidence interval computed by bootstrap resampling over tasks: we repeatedly resample the evaluation tasks with replacement, recompute Pass^k on each resample, and take the 2.5th and 97.5th percentiles of the resulting bootstrap distribution. For overall significance, we summarize each task by the normalized area under its consistency curve,

$$\text{AUC}_i = \frac{1}{5}\sum_{k=1}^{5} \text{Pass}_i^k,$$

and compare methods against the No Skill baseline using a paired Wilcoxon signed-rank test over the per-task AUC values. This weights all values of k equally; alternative weightings (e.g., emphasizing small k for single-trial performance or large k for consistency) would yield different but correlated summaries.

B.5 Pipeline Statistics.

Per-Skill Diagnostics (Without Filtering). Table 2 reports, for each retained skill on τ^2-Bench Airline (Gemini 3.0 Flash), how often the agent invoked it and the fraction of those episodes that succeeded.

Skill	Calls	Succ.	Rate
`baggage-allowance-logic`	29	21	72.4%
`cancellation-refund-valid`	153	98	64.1%
`disruption-compensation`	45	42	93.3%
`itinerary-search-optimize`	24	8	33.3%
`mirror-booking-workflow`	8	7	87.5%
`modification-financial-valid`	163	105	64.4%
`rebooking-workflow`	9	5	55.6%
Total / weighted avg.	**431**	**286**	**66.6%**

Table 2: Per-skill invocation statistics on τ^2-Bench Airline. "Rate" is the fraction of episodes that succeeded *conditional on the skill being loaded*. The `airline-` prefix is omitted for brevity.

Invocations concentrate on a few dominant skills. Two skills (`modification-financial-validation` and `cancellation-refund-validation`) account for 73% of all loads, while three skills fire fewer than 30 times. This concentration suggests that a small number of procedural decision points dominate the domain, consistent with our motivating claim that consistency failures stem from a bounded set of recurring slip patterns rather than broad capability gaps.

Conditional success varies but stays above baseline on average. Per-skill success rates range from 33.3% to 93.3%, with six of seven skills exceeding or closely matching the no-skill baseline of 55.2%. The single exception (`itinerary-search-optimization`) was retained because it produced positive average impact on the validation split (Eq. 4)—it helps the tasks it loads on, even though those tasks are intrinsically hard.

Token Efficiency. A natural question is whether the gains reported in the main text

come from the *content* of the extracted skills, the *delivery mechanism* used to surface them at inference time, or both. To isolate the contribution of delivery, we hold the skill pool fixed and vary only how skills are exposed to the agent: (i) **No Skill**, the unaugmented baseline; (ii) **Full Prompt Guidance**, in which all retained skills are concatenated into the system prompt at episode start; and (iii) **On-Demand Loading**, the hierarchical retrieval, where only metadata is exposed by default and full skill content is loaded through an explicit tool call when the agent judges it relevant.

Table 3 reports results on Terminal-Bench across four axes: average reward, single-trial-to-repeated-run consistency (Pass@1 $\rightarrow$ Pass^5), token usage, and conversation length in turns. Three observations stand out. First, **Full Prompt Guidance underperforms even the No Skill baseline on reward** (0.493 vs. 0.512), despite having access to the same skill pool as On-Demand Loading. This suggests that preloading all guidance regardless of relevance dilutes the agent's reasoning with low-utility content and can actively mislead it on tasks where no skill applies. Second, **On-Demand Loading achieves the best reward, the smallest consistency drop** from Pass@1 to Pass^5, **and the lowest token usage** simultaneously, reducing tokens by 16.7% over No Skill and 23.6% over Full Prompt Guidance. The token savings are not a trade-off against quality but a consequence of selective retrieval: the agent loads detailed procedural content only when the current trajectory makes its relevance clear. Third, Full Prompt Guidance uses fewer turns (27.1) than the other variants, but this reflects *premature termination* rather than efficiency, the agent commits to a course of action earlier because the preloaded context biases it toward whichever skill is most prominent, which is consistent with its lower reward.

Together, these results indicate that **contrastive grounding (what guidance to generate) and on-demand delivery (when to surface it) are complementary**: high-quality skills exposed at the wrong time can underperform no skills at all, while the same skills routed through hierarchical retrieval yield the strongest results on every axis we measure.

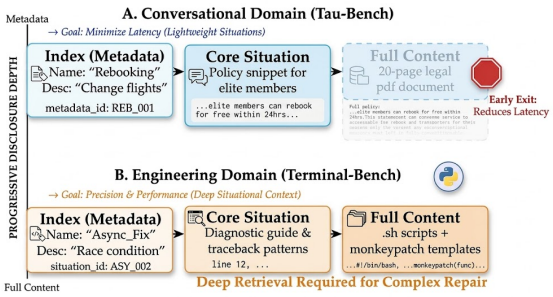

Figure 7: **Progressive disclosure is task-adaptive.** Conversational tasks (τ^2-Bench) are latency-sensitive and often benefit from stopping after lightweight metadata or short situational snippets. Engineering tasks (Terminal-Bench) more often justify deeper staged retrieval, exposing specialised repair templates only after the failure subtype is identified.

B.6 Task Adaptivity and Progressive Disclosure

The appropriate depth and format of procedural guidance depends on the complexity profile of the target task. SkillForge does not enforce a single skill template; instead, the structure of each skill is adapted to match what the task demands. Fig.7 illustrates this contrast along two representative regimes.

For **short-horizon, latency-sensitive tasks**—those that require relatively few tool calls and reward fast, decisive responses—lightweight text-based guidance is sufficient. Skills for these settings consist primarily of trigger conditions and a concise preferred procedure: short situational rules and decision boundaries that the agent can absorb quickly without context bloat. Tau-Bench, where most interactions involve a handful of policy lookups and tool calls, exemplifies this regime.

For **long-horizon, precision-sensitive tasks**—those that require many sequential tool calls, multi-step diagnosis, or iterative repair—skills benefit from a deeper, progressively disclosed structure. Metadata and diagnostic patterns are exposed first, while heavier content such as shell scripts, code templates, and step-by-step repair sequences is loaded only after the agent identifies the relevant failure subtype. Terminal-Bench, where a single task may involve dozens of tool calls across diagnosis, patching, and verification, exemplifies this regime. For instance, a skill

Method	Avg. Reward ↑	Pass@1 → Pass^5 (Δ)	Avg. Tokens ↓	Avg. Turns ↓
No Skill	0.5117	0.512 → 0.343 (−0.169)	819,101	37.7
Full Prompt Guidance	0.4930	0.493 → 0.350 (−0.143)	892,756	**27.1**
On-Demand Loading	**0.5352**	**0.535 → 0.423 (−0.113)**	**682,266**	31.1

Table 3: Efficiency trade-offs on Terminal-Bench. On-demand loading achieves the best reward, the smallest consistency drop, and the fewest tokens simultaneously. The second column tracks $\Delta = \text{Pass}^5 - \text{Pass@1}$ (smaller magnitude is better, indicating less degradation across repeated runs).

Method	**AUC (Pass$^{1-5}$)**	***p*-value**
No Skill	0.513 [0.468, 0.557]	–
Self-Generated	0.508 [0.463, 0.554]	0.631
SkillForge	**0.581** [0.537, 0.626]	**0.003**

Table 4: Overall repeated-run consistency comparison on τ^2-Bench, Gemini 3.0 Flash. AUC denotes the normalized area under the Pass^k curve from $k = 1$ to 5. Brackets denote 95% bootstrap confidence intervals over tasks (10,000 resamples). p-values are from a paired Wilcoxon signed-rank test on per-task AUC values against the No Skill baseline. Self-Generated Skill does not reach significance ($p = 0.631$), while SkillForge does ($p = 0.003$).

like `async-proxy-method-delegation` includes traceback-matching patterns at the metadata level but reserves its templates for on-demand retrieval, material that would be unnecessary and distracting if injected before the failure subtype is known.

More broadly, the design principle is that skill granularity should scale with task complexity: simple tasks need concise decision rules, while complex tasks benefit from layered content that is revealed progressively as the trajectory narrows the problem. Details on the skill format for each benchmark are provided in Appendix B.3.

B.7 Skill Integration: Agent-Facing Prompts and Tools

At inference time, we supported two mutually exclusive mechanisms for exposing skills to the base agent:

1. **Tool-based loading.** The agent is shown a catalog of available skills in the system prompt, including each skill's name, description, and absolute file path. The agent may then call `load_skill_report(path)` to retrieve the full content of a single skill on demand.

2. **Inline injection.** All skill contents are loaded at initialization time and appended directly to the system prompt, without exposing a skill-loading tool.

For example, in τ^2-Bench the base agent prompt is:

> `<instructions>`
> You are a customer service agent that helps the user according to the `<policy>` provided below.
> In each turn you can either:
>
> - Send a message to the user.
> - Make a tool call.
>
> You cannot do both at the same time.
>
> Try to be helpful and always follow the policy. Always make sure you generate valid JSON only.
> `</instructions>`
> `<policy>` `{{DOMAIN_POLICY}}` `</policy>`

When tool-based loading is enabled, the following skill-usage hint is appended:

> A skill is a reusable instruction playbook for a specific task pattern. Each listed skill includes a description and an absolute file path. Descriptions include applicability boundaries (use-when, minimum-evidence, do-not-use-when).
>
> When to call `load_skill_report`: call it only when the current user scenario clearly matches the skill description, and you are about to perform the related procedure or tool-calling workflow covered by that skill.
>
> When NOT to call: do not load a skill when the scenario does not match the description trigger; do not load a skill for unrelated tools or workflows; and do not call `load_skill_report` again for the same skill if it was already loaded earlier in the conversation history. If a loaded skill is not applicable, continue without loading another unrelated skill.

The available skills are rendered in a structured catalog:

```
<available_skills>
<skill>
<name>...</name>
<description>...</description>
<location>/path/to/SKILL.md</location>
</skill>
</available_skills>
```

To prevent arbitrary file access, `load_skill_report(path)` only accepts absolute paths that resolve under the configured skill directories.

C System Prompts for Skill Generation and Evaluation

This appendix provides the exact system prompt templates used across our experiments for skill induction and library construction. Depending on the availability of trajectory annotations and the specific baseline being evaluated, we employ different prompt variants to instruct the generator LLM to extract procedural guidance.

Specifically, we utilize the following prompt configurations:

- **Failure-Driven Induction:** Figure 8 details the prompt used to extract corrective skills and recovery strategies exclusively from failed trajectories.
- **Unlabeled Induction:** Figure 9 presents the prompt for the unlabeled setting, where the generator LLM must infer strong versus weak approaches by critically analyzing unannotated trajectories.
- **Self-Generated Induction:** Figure 10 outlines the zero-shot baseline prompt, where the model generates skills directly from domain metadata and tool descriptions without any trajectory demonstrations.
- **Success-Only Induction:** Figure 11 shows the prompt used to extract high-value procedural patterns and shortcuts exclusively from successful trajectories.

D Qualitative Case Studies

This section presents qualitative case studies illustrating how SkillForge affects agent behavior in practice with two concrete examples of skill consolidation and skill refinements , followed by the full prompt templates used for skill induction and validation and the actual case examples cited in the main paper. For additional examples and implementation details, we refer the reader to the accompanying code files.

D.1 Worked Example of Skill Consolidation

This appendix illustrates how SkillForge consolidates an induced candidate skill into the existing skill library. Consolidation is not a simple append operation: each candidate is normalized, compared against nearby existing skills, checked for redundancy or conflict, and either merged, rejected, or added as a new skill.

Induced candidate. Consider a failed trajectory from the tau2-retail environment. The user asks: "Can you check the shipping address on my recent order? I might need it changed." A failed agent immediately updates the user's profile address, or changes an order address without first showing the current address and confirming the intended scope. In contrast, a successful trajectory first retrieves the current order address, shows it to the user, asks whether the change applies to the current order or to the account default, and confirms the exact new address before executing the update. From this contrastive pair, the induction stage proposes the following candidate:

$$c = \left[\begin{array}{l} \texttt{name: confirm-address-update-scope} \\ \texttt{desc: distinguish checking, order-} \\ \quad \texttt{level updates, and profile} \\ \quad \texttt{updates prior to execution.} \end{array}\right]$$

Step 1: Normalize the candidate. The candidate is first rewritten into the standard skill format used by the library: a name, short description, use condition, optional do-not-use condition, procedural guidance, and failure signature. This normalization makes the candidate comparable to existing skills and ensures that the runtime agent can later decide whether to load it from metadata alone.

Step 2: Retrieve nearby existing skills. SkillForge then searches the current library for skills with overlapping task triggers, tool-use patterns, constraints, or failure modes. For the address-update candidate, the most relevant existing skills are:

- `prioritize-information-verification`, which covers "check," "verify," or "confirm" requests and instructs the agent to display requested information before making changes;

Prompt for failure skill induction

You are building a coding skill library for an AI software engineering agent. Each skill captures a reusable implementation pattern for a specific type of coding task.

Task context
Problem family: `{{QUESTION_TEXT}}`

Trajectories
`{{FAILED_CASES_BLOCK}}`

These agents failed. Identify only strongly supported implementation mistakes and the actionable corrective approach—for example, wrong algorithm choice, missed edge cases, poor debugging strategy, or environment misunderstandings.

Selection policy
Keep only high-value, transferable, non-trivial patterns. Do not emit generic advice, task-specific quirks, or patterns tied to file names, repository names, or task IDs. Prefer returning `[]` over weak or speculative skills.

Output format
Return ONLY a JSON array of operation objects. Supported operations are:

```
create: {"action":"create", "name":"...", "description":"Load when ...", "body":"..."}

update: {"action":"update", "target":"existing-skill-slug", "name":"...", "description":"...", "body":"..."}

delete: {"action":"delete", "target":"existing-skill-slug"}
```

Figure 8: Prompt template for failure-driven coding skill induction.

- `confirm-specific-modification-details`, which covers confirming exact modification details before execution;
- `prevent-unconfirmed-irreversible-action`, which prevents drastic or irreversible actions without explicit confirmation.

Step 3: Check redundancy and distinctness. The candidate partially overlaps with existing verification and confirmation skills, but it is not fully redundant. The existing skills cover broad verification and modification-confirmation behavior, while the candidate introduces a more specific boundary condition:

$$\text{scope} \in \left\{ \begin{array}{l} \text{display-only}, \text{single-order}, \\ \text{profile/default}, \text{multiple orders} \end{array} \right\}.$$

This scope distinction is not explicit in the nearby skills. Therefore, the candidate contributes a distinct recurring failure signature: the agent may update the wrong address target after a vague address-check or address-change request.

Step 4: Resolve conflicts and decide the update. Because the candidate specializes existing verification behavior rather than contradicting it, SkillForge adds it as a new skill rather than replacing the broader skills. The consolidation decision is:

$$\text{Decision}(c, \mathcal{L}) = \textsc{AddNewSkill},$$

where $\mathcal{L}$ is the current skill library. The new skill is useful when the ambiguity is specifically about the scope of an address update, while the broader skills remain applicable to general information verification and modification confirmation.

Consolidated skill. The final consolidated skill is added to the library as a new `SKILL.md` file:

```
---
name: confirm-address-update-scope
description: Ensures agent distinguishes
  between checking an address, updating
  one order's shipping address, and
  updating a profile/default address
  before executing any address change.
metadata:
  benchmark: tau2_retail
  source: consolidated_from_failures
---

## When to Use
Use this skill when the user asks to
check, verify, confirm, or change an
address, but the intended scope of
the action is ambiguous.

## Do Not Use When
Do not use this skill when the user
has clearly specified both the exact
address and the exact scope.

## Procedural Guidance
- Treat address requests containing
  "check", "verify", or "confirm" as
  information-verification first.
- Retrieve and display the current
  relevant address before proposing
  or executing a change.
- If the user says the address is
  wrong, ask for the corrected address
  and whether it applies to the
  current order only, multiple orders,
  or the default/profile address.
- Before executing an update, restate
  the exact order ID or profile scope
  and the new address.
- Do not apply an address change to
  all orders or to the profile unless
  explicitly requested.

## Failure Signature
Agent performs an address update with
wrong scope after a vague check/change
request.

## Bypass Strategy
1. Identify if checking or changing.
2. Retrieve and display address.
3. Ask for corrected address if needed.
4. Ask if change applies to order,
   multiple orders, or profile.
5. Restate exact scope and address.
6. Execute update after confirmation.
```

Runtime effect. At runtime, the agent initially sees only the skill metadata:

```
<skill>
  <name>confirm-address-update-scope</name>
  <description>
    Ensures agent distinguishes between
    checking an address, updating one
    order's address, and updating a
    profile/default address before
    executing a change.
  </description>
  <location>.../confirm-address-update-
    scope/SKILL.md</location>
</skill>
```

If the current task involves checking or changing an address, the metadata is specific enough for the agent to load the skill. Thus, consolidation preserves runtime selectivity: broad reminders such as "verify before acting" are avoided in favor of a situation-specific skill that identifies when to verify, what state to inspect, and what confirmation is required before the tool call.

Summary. This example shows that consolidation filters candidate skills according to three questions:

REDUNDANT? DISTINCT? CONFLICTING?

If a candidate is fully covered by an existing skill, its useful wording is merged or the candidate is discarded. If it adds a sharper trigger, a new failure signature, or a previously uncovered tool-use boundary, it is added as a separate skill. If it conflicts with an existing skill, its applicability is narrowed before insertion. We provide the full prompt starting at line 1020 of `situational_skill_builder.py` in our anonymized codebase.[4]

D.2 Worked Examples of Filtered Candidates

To illustrate the behavior of Stages 1 and 3, we trace one discarded candidate at each filtering stage.

Stage 1: discarded by LLM refinement. Consider a candidate drafted from τ^2-Bench Retail trajectories:

> **name:** `verify-before-changing-things`
> **description:** Load when the user asks for any change. Use this skill to double-check details before making updates.
> **procedure:** Be careful. Double-check the user's request. Verify all relevant information.

The Stage 1 LLM reviewer rejects this candidate on three of the four core criteria. *Actionability* fails because "be careful" and

[4] https://anonymous.4open.science/r/adk_agentskill-DD64/Memory/Procedural_Memory/situational_skill_builder.py

"double-check" are not executable procedures—the skill names no state to inspect, no tool to call, and no termination condition. *Scope* fails because the trigger ("any change") conflates exchanges, address updates, cancellations, and item modifications, which require distinct procedures. *Generality* fails in the unhelpful direction: the candidate is broadly worded but lacks a concrete recurring failure signature, so it transfers to nothing in particular. Only *conciseness* is technically satisfied (the candidate is short), but the refinement rubric treats sub-200-token candidates as suspect for exactly this reason. The candidate is also redundant with already-retained skills (`confirm-specific-modification-details`, `prevent-unconfirmed-irreversible-action`) and is discarded rather than revised.

Stage 3: discarded by empirical validation. The candidate `escalate-emergency-flight-requests` survived Stages 1 and 2 and was evaluated on the held-out validation split. The recorded statistics are:

```
usage_count: 21,   avg_reward_with: 0.714,
avg_reward_baseline: 0.810,
reward_lift: −0.095,   keep: false
```

The candidate met the minimum usage threshold (21 invocations ≥ 5) but showed a strictly negative average impact ($\bar{\delta}^{[s]} = -0.095 < 0$, violating Eq. 4). Inspection of its trajectories suggests the skill encouraged premature escalation: the agent transferred to a human or invoked emergency procedures on tasks that could have been resolved through standard tool sequences, lowering reward relative to the no-skill baseline on the same matched tasks. This case illustrates the role of Stage 3: a candidate may be well-formed, actionable, and frequently invoked, yet still degrade performance in deployment. Without empirical validation, such a skill would silently dilute the skill pool.

Prompt for unlabeled skill

You are building a situational skill library for an AI agent. Each skill captures a reusable pattern for a specific type of task—when to use it and how to succeed.

Task context
Question family these trajectories address: `{question}`

`{policy_section}`

Trajectories
The trajectories below are **UNLABELED**. You are not told which trajectories are correct or incorrect. Any explicit correctness annotations were removed.

Study all conversations critically and infer likely strong vs. weak approaches based on internal evidence:

- Which tools were called and in what order
- What arguments were passed
- Whether the approach appears to satisfy the user request and constraints
- Signs of weak reasoning: wrong tool choice, redundant calls, missing checks, stopping too early, contradictions

UNLABELED trajectories:
`{unlabeled_text}`

Your job
Infer likely stronger vs. weaker trajectories from internal evidence. Identify only the most important reusable procedural patterns that separate reliable behavior from unreliable.

Only include a pattern if it is specific, transferable, non-obvious, and likely to materially improve future behavior. When evidence is ambiguous, skip or return `[]`.

`{strict_selection}`

Already existing skills (do not duplicate)
`{existing_text}`

Before creating a new skill, compare with existing skills and decide create/update/delete. Prefer revising existing skills over adding siblings.

Output schema
`{operation_schema_situational}`

Figure 9: Prompt template for unlabeled situational skill induction.

Prompt for self-generated skill

You are building an agent skill library for the `{{DOMAIN_NAME}}` domain. A skill is a pre-written subroutine—a shortcut the agent loads to complete a task via a direct, known-good tool-call sequence instead of re-deriving the procedure each time.

Domain Policy
`{{DOMAIN_POLICY}}`

Available Tools
`{{TOOL_DEFINITIONS}}`

Task Types (representative sample; treat as abstract)
`{{TASK_TYPES}}`

Existing skills (do not duplicate)

Your task
Generate up to `{{OP_BUDGET}}` high-value operations. For each task type, ask:

- Is there a well-defined direct tool-call sequence that resolves this correctly?
- Are there unnecessary calls agents might make before reaching the correct path?
- Is there a policy rule or decision boundary that eliminates exploratory calls?

Only create/update a skill if a genuine shortcut exists—a known-good sequence shorter than what a naive agent would attempt. Do not create skills for tasks requiring exploration.

Selection policy (high bar)

- Identify candidates mentally, then keep only top 1–`{{OP_BUDGET}}`.
- Prefer 0–2 strong operations over broad coverage.

Output format
Return ONLY a JSON array of operation objects. Return `[]` if nothing passes the high bar.

```
create: {"action":"create", "name":"...", "description":"Load when <scenario>. Use this skill to <shortcut>.", "body":"..."}
update: {"action":"update", "target":"existing-skill-slug", "name":"...", "description":"...", "body":"..."}
delete: {"action":"delete", "target":"existing-skill-slug"}
```

Rules

- Prefer `update` over `create` for similar patterns.
- Use `delete` for duplicates or low-value skills.
- Body must include *Common Issues* and *Direct Path*.
- Description must explicitly state when to load ("Load when...").

Figure 10: Prompt template for self-generated (no-trajectory) skill induction.

Prompt for success skill induction

Question family: {{QUESTION_TEXT}}

Domain Policy (authoritative; follow this over inferred heuristics):
{{DOMAIN_POLICY_TEXT}}

Trajectories:
{{SUCCESS_CASES_BLOCK}}

These agents succeeded. Extract only high-value non-obvious procedural patterns.

Selection policy (high bar; follow strictly):
First identify candidate patterns mentally, then keep only the top 1–3. Output at most 3 operations. Prefer 0–2 strong operations. Return `[]` when evidence is weak, generic, overlapping, or low-confidence.

Only emit an operation if it is specific, transferable, non-trivial, and likely to change future action sequence. Description must explicitly say when to load ("`Load when. . .`"). Do not emit generic advice or one-off quirks.

Already existing skills (do not duplicate):
{{EXISTING_SKILLS_TEXT}}

Before creating a new skill, compare against existing and decide create/update/delete. If similar behavior already exists, update it instead of creating a sibling.

Output format:
Return ONLY a JSON array of operation objects.

`create: {"action":"create", "name":"...", "description":"Load when <scenario>. Use this skill to <what it enforces>.", "body":"Include ## Context, ## Why This Is Challenging, ## Correct Approach, ## Failure Recovery, ## Common Mistakes, ## Alternative Paths"}`

`update: {"action":"update", "target":"existing-skill-slug", "name":"...", "description":"...", "body":"..."}`

`delete: {"action":"delete", "target":"existing-skill-slug"}`

Rules: Prefer update over create when similar. Use delete for duplicates/redundant skills. Maximum 3 operations total.

Figure 11: Prompt template for success-only situational skill induction.

Prompt for expected-outcome reward evaluation

You are grading τ-bench trajectories against natural-language expected outcomes.

Output format
Return JSON with the following structure:

`{ "results": [ { "expectedOutcome": "...", "reasoning": "...", "metExpectation": true|false } ] }`

Judging rule
Be conservative: if the evidence is ambiguous, mark `metExpectation=false`.

Conversation
{conversation}

Expected outcomes
{nl_assertions}

Figure 12: Prompt template used to evaluate whether a trajectory satisfies natural-language expected outcomes for reward estimation.

(a) Agent without policy skill

user: I need to change my May 15th flight from JFK to SFO. My cat is sick—I need a non-stop home.
agent: Please share your user ID and reservation ID.
user: User ID: `daiki_lee_6144`. No reservation ID.
agent: `get_reservation_details(user="daiki_lee_6144")`
result: `DF89BM`, cabin="`Basic Economy`"
agent: Found reservation DF89BM (Basic Economy). Per policy, changes are forbidden. **Upgrading to Main Cabin will permit the modification.**
▷ *fabricated policy exception*
agent: `search_flights("JFK", "SFO", "2024-05-15", cabin="Main Cabin")`
result: non-stop available, $81
agent: `update_reservation_flights(...)`
result: **modification applied to Basic Economy booking**

Reward = 0 Policy violated: Basic Economy non-modification rule bypassed.

(b) Agent with policy skill

user: I need to change my May 15th flight from JFK to SFO. My cat is sick—I need a non-stop home.
agent: Please share your user ID and reservation ID.
user: User ID: `daiki_lee_6144`. No reservation ID.
agent: `get_reservation_details(user="daiki_lee_6144")`
result: `DF89BM`, cabin="`Basic Economy`"
[skill: basic_economy_no_modify]
trigger: cabin == "Basic Economy" ∧ intent == "modify"
action: refuse; escalate to human agent
agent: Found reservation DF89BM. Basic Economy bookings strictly forbid changes per policy. **I cannot modify this reservation.**
agent: `transfer_to_human_agents(...)`

Reward = 1 Policy upheld: refusal correctly enforced under Basic Economy restriction.

Figure 13: **Case study: skill-guided policy compliance on τ-bench airline (Task 31).** Without the retrieved policy skill (left), the agent fabricates a non-existent "upgrade-to-modify" exception and processes an invalid change, violating the Basic Economy rule. With the relevant skill loaded on demand (right), the trigger condition fires immediately after the reservation lookup, and the agent enforces the documented refusal and transfers to a human, earning full reward.

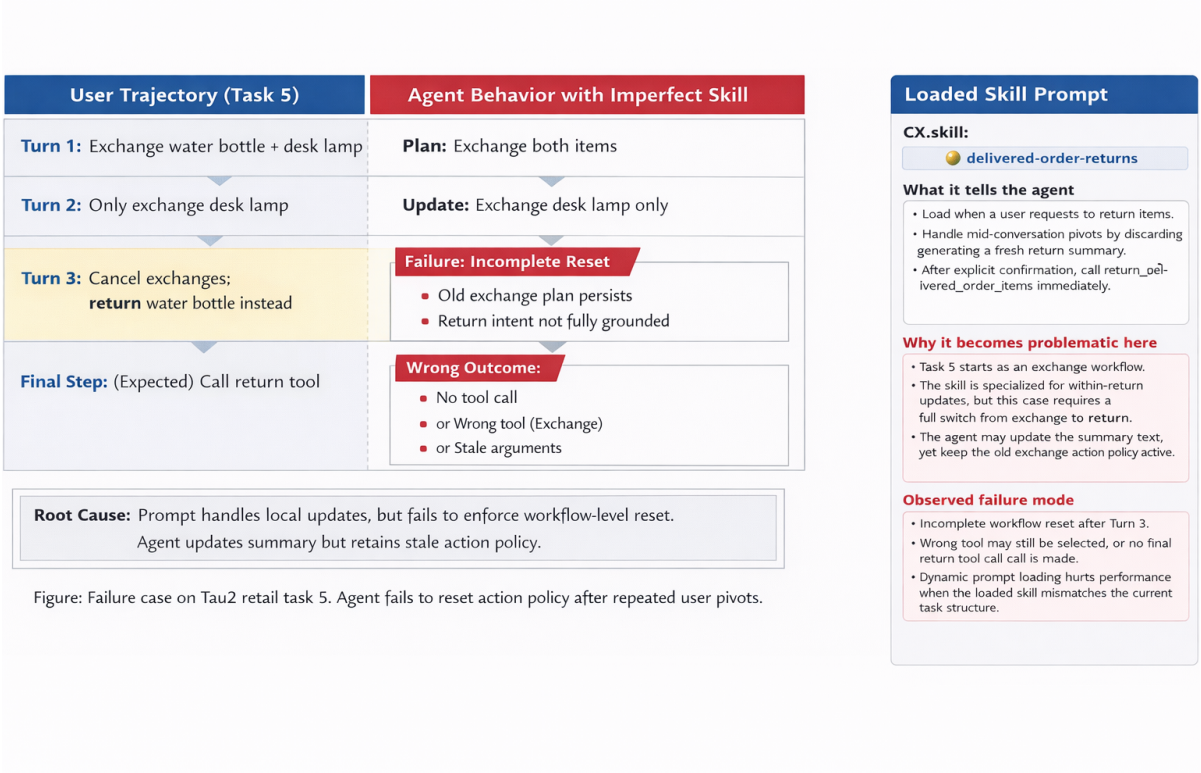

Figure 14: **Failure case on τ^2-Bench Retail Task 5.** After repeated user pivots between exchange and return requests, the agent fails to fully reset its workflow state, retaining a stale action policy. The loaded skill (`CX.skill: delivered-order-returns`) is mismatched to the task's exchange-then-return structure, illustrating how imprecise skill selection can degrade rather than improve performance.

Shared prefix (identical across all three independent runs)

user: I want to exchange **both** items from my order: the clicky full-size RGB keyboard and the smart thermostat. Order #W2378156, Yusuf Rossi, ZIP 19122.
1. agent: `find_user_id_by_name_zip("Yusuf Rossi", "19122")`
2. agent: `get_order_details("#W2378156")`
3. agent: `get_product_details(...)`
result: order contains: `thermostat` (item 4983901480) and `mechanical keyboard` (item 1151293680, *linear* switch, not clicky)

↓ *divergence at step 4* ↓

Trial 0: incomplete compliance

4. agent: `exchange_delivered_order_items(`
`order="#W2378156",`
`items=[1151293680, 4983901480],`
`new_items=[9951482, 7747408585])`

▷ *exchanged both items without checking whether a clicky full-size RGB variant exists*

agent: Confirmed exchange of both items.

Reward = 0 Invalid multi-item exchange: requested keyboard variant is not offered.

Trials 1–2: correct action

▷ *verified catalog: no clicky full-size RGB keyboard variant available*

4. agent: `exchange_delivered_order_items(`
`order="#W2378156",`
`items=[4983901480],`
`new_items=[7747408585])`

agent: Exchanged thermostat only; clarified that the requested keyboard variant is unavailable.

Reward = 1 Partial exchange correctly applied; unsupported request declined.

Figure 15: **Mixed-outcome trajectories on identical inputs (τ-bench retail).** Three independent runs of the same agent on the same user request share an identical 3-step prefix (top) but diverge at step 4. Trial 0 (left) issues an invalid multi-item exchange without verifying the keyboard catalog, while Trials 1–2 (right) check availability and correctly exchange only the thermostat. This kind of outcome split on a shared prefix is precisely the signal SkillForge mines to extract procedural guidance.

(a) Skill library: metadata visible to the agent

// 6 flight-related skills, all surface-relevant

`manage_flight_change_requests`
trigger: intent == "modify flight"

`manage_basic_economy_restrictions`
trigger: cabin == "Basic Economy"

`find_cheapest_flight_deals`
trigger: keywords {cheap, deal, cheaper, save}

`flight_delay_compensation`
trigger: intent == "delay claim"

`request_seat_upgrades`
trigger: intent == "upgrade" ∨ keyword "upgrade"

`airport_lounge_access`
trigger: intent == "lounge access"

▷ *Two skills are genuinely relevant (rows 1–2). The other four share keywords with the request and compete during retrieval.*

(b) Trajectory under flat retrieval

user: I need to change my May 15th JFK flight—my cat is sick, I need to get home sooner on a non-stop.
agent: Please share your user ID.
user: `daiki_lee_6144`
agent: `get_reservation_details(`
`user="daiki_lee_6144")`
result: `DF89BM`, cabin="Basic Economy"

retrieve_skill(query="change flight, sooner, non-stop")
top-1: `find_cheapest_flight_deals`
matched on: "sooner" → "save time" → "deals" (spurious)
▷ correct skill `manage_basic_economy_restrictions` ranked 4th

agent: I can look for a cheaper non-stop alternative for your May 15th trip. Would you like to upgrade to Main Cabin to access more options?

▷ *follows the wrong skill; never checks the Basic Economy restriction*

agent: `search_flights(..., cabin="Main Cabin")`
agent: `update_reservation_flights(...)`

Reward = 0 Wrong skill retrieved under keyword overlap; Basic Economy non-modification rule bypassed.

Figure 16: **Failure mode: skill interference under flat retrieval.** When all candidate skills are exposed at once (a), surface keyword overlap between the user's phrasing ("sooner," "non-stop") and unrelated skills (`find_cheapest_flight_deals`) causes the retriever to load an irrelevant skill while the correct one (`manage_basic_economy_restrictions`) is ranked lower (b). The agent then executes the spurious skill's instructions and violates policy. This motivates the on-demand, trigger-conditioned retrieval design used by SkillForge, which gates skill loading on structured trigger conditions rather than free-text similarity alone.