SkillMuse: Multi-Turn User Simulation as a Feedback Generator for Skill Evolution

Anonymous authors
Paper under double-blind review

Abstract

In enterprise conversational settings such as customer support, LLM-based assistants rely on high-quality domain skills to answer; yet no mechanism attributes execution failures to skill defects and turns them into targeted improvements—operational data keeps accumulating while skill quality stagnates. Recent work on skill self-evolution builds a revision loop toward closing this gap, but the feedback driving the loop is derived from single-turn question-answering evaluation: a single exchange exposes only the gaps in the user's opening statement, so once the first revision has patched them, the evaluation supplies little further signal, and defects that surface only across multiple turns remain unreached—the loop is intact, yet evolution stalls. Governance in these systems likewise rests on a scalar score, which can only reject a degraded candidate outright, without diagnosing or repairing its structural cause. We argue that the binding constraint on sustained skill evolution lies less in editing capability or the number of iterations than in whether the evaluation feedback keeps supplying trustworthy evolution gradients. We introduce **SkillMuse**, coupling two components. Trustworthy feedback generates the gradient: we recast multi-turn user simulation from an evaluation endpoint into a feedback generator, so follow-up questions expose defects layer by layer and every round of revision both consumes and produces feedback. Controllable governance constrains the gradient direction: an independent governance layer actively repairs factual degradation and structural bloat after each revision, so degradation does not accumulate across rounds. Across six categories of cloud services, nine production skills, and 98 skill-reference files, SkillMuse exceeds self-reflection-based evolution by 23.0 points and single-turn-QA-driven evolution by 15.4 points.

Code: `anonymous.4open.science/r/SkillMuse-5036F0CE`

1 Introduction

Customer-support agents are moving beyond single-turn question answering toward multi-turn interaction, knowledge retrieval, and tool invocation (Wang et al., 2024a); in the cloud-services domain, RCACopilot (Chen et al., 2024) applies LLMs to incident root-cause analysis, RCAgent (Wang et al., 2024b) endows agents with autonomous diagnosis, and D-Bot (Zhou et al., 2023) structures database diagnosis via tree search. Yet the more capable agents become, the more the knowledge module carrying that capability becomes the bottleneck: an agent's answer quality directly depends on its loaded **skill**, a portable module that encapsulates domain knowledge and handling procedures. Skill maintenance, however, still relies on manual work: human annotation is costly, slow, and error-prone, and the expertise of human agents cannot be consolidated automatically into reusable knowledge—operational data keeps accumulating while skill quality stagnates.

Recent self-evolution work has built "evaluate–revise" loops, but whether such a loop can *sustain* evolution depends on two stages, and existing work is structurally limited in both. **The first is feedback**: feedback must reveal the latent defects arising in multi-turn interaction and be attributable to repairable knowledge gaps, whereas existing feedback is confined to what a single exchange makes visible—once the first revision patches the visible gaps, the evolution gradient decays and evolution stalls. **The second is governance**: successive revision erodes a skill along two dimensions—in *fact consistency*, new knowledge may come at the cost of existing facts, or erroneous content conflicting with the original text may be written in; in *structural consistency*, knowledge bloat and reference

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

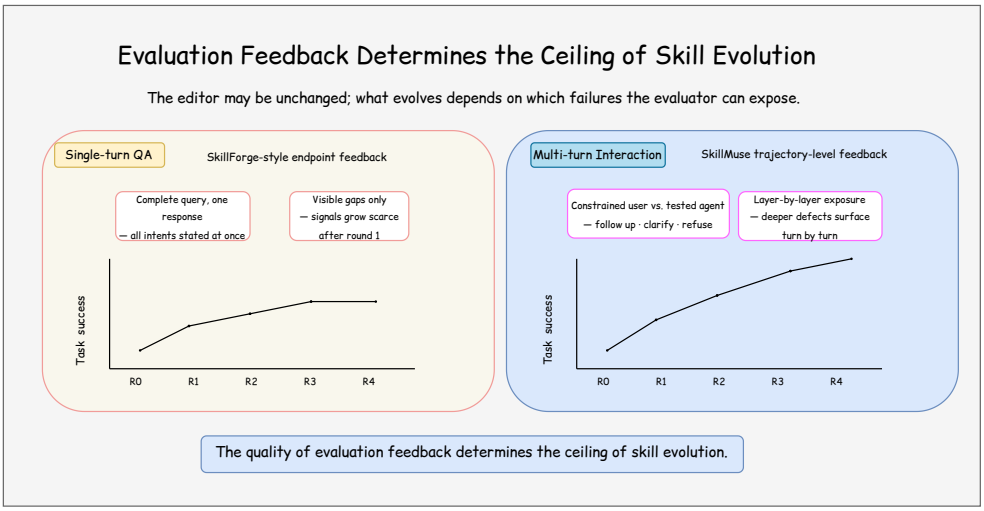


Figure 1: The evaluation feedback modality determines the ceiling of skill evolution. **Left:** single-turn QA patches the gaps visible in the first exchange; the evolution gradient then decays and the TSR curve saturates around round 2. **Right:** multi-turn interaction keeps exposing latent failures: each round of revision lets the dialogue proceed further and reach the next layer of defects, so the gradient renews itself and TSR keeps climbing.

breakage accumulate across rounds. Existing governance, however, is a scalar gate that senses a falling total score yet can neither identify nor repair either class of degradation.

We therefore propose **SkillMuse**, whose two pillars address the two limitations respectively. Trustworthy feedback recasts multi-turn user simulation from an evaluation endpoint into a feedback generator, so that follow-up questions keep producing fresh evolution feedback. Constraint-based governance treats a skill as a structured knowledge system and adds an independent layer that actively repairs factual degradation and structural bloat after each revision, preventing skill degradation and bloat.

Our contributions are as follows.

1. We formalize three necessary conditions for reliable feedback (coverage, accuracy, and attributability) and recast multi-turn user simulation from an evaluation endpoint into a feedback generator. An intent state machine gates coverage; separate evaluation of the two sides partitions responsibility between simulator and service agent to secure accuracy; and collective attribution screens repairable gaps by root cause while distilling their cross-sample commonalities. Together they furnish a reliable gradient for skill evolution.
2. We treat a Skill as a structured knowledge system whose multi-round revision incurs degradation along two dimensions that a scalar score leaves undiagnosed: in fact consistency, existing facts are lost or erroneous content conflicting with the previous round's revision is written in; in structural consistency, knowledge bloat and reference breakage accumulate. SkillMuse maintains fact consistency against dual anchors as a hard constraint and repairs structural degradation through graph-structural diagnosis as a soft constraint, extending governance from scalar-gated rejection to diagnosis-driven repair.
3. Across six categories of cloud services, nine skills, and 98 skill-reference files, SkillMuse improves over the initial skills by 51.8 points, over self-reflection-based evolution by 23.0 points, and over single-turn-QA-driven evolution by 15.4 points. The framework is deployed in the production environment of a major cloud provider.

2 Related work

2.1 User simulation

User simulation is now standard for the automatic evaluation of task-oriented dialogue. τ-bench (Yao et al., 2024) advances evaluation from "can the task be completed" to "can it be completed *reliably*" by grounding tool-agent-user interaction in real-world domains. ECom-Bench (Wang et al., 2025) and VoiceAgentEval (Xu et al., 2025) improve fidelity by grounding the simulator in real user personas, while SAGE (Shea et al., 2026) injects business profiles and enterprise knowledge bases into the simulator and generates probing questions from them, surfacing 33% more agent errors than a generic user simulator.

These results establish that simulation exposes errors that single-turn testing does not reach, yet they share one limitation: they treat simulation as an evaluation endpoint. Once a trajectory has served to adjudicate success or failure, it is discarded. Repairable gaps are not separated from limitations beyond the skill, and no failure case is returned to the skill, which leaves evaluation and evolution disjoint. The reliability of the simulation itself is also left unexamined: whether the simulator covers the user's real intents is rarely verified, and simulation distortion contaminates the evaluation conclusion directly. SkillMuse departs from this practice by recasting multi-turn simulation into a feedback generator for evolution: an intent state machine gates coverage, separate evaluation of the two sides isolates distortion, and independent attribution screens repairable gaps and returns them to the skill.

2.2 Agent skill evolution

Agent skills (Zhang et al., 2025a) have become a core carrier of agent capability: they encapsulate domain knowledge and support complex tasks beyond tool use, and are treated as a knowledge-system category in recent surveys (Jiang et al., 2026). Yet their maintenance still rests on manual authoring. Existing self-evolution work (Gao et al., 2025) falls into three tiers by feedback source, which we examine along two dimensions: the feedback signal and the revision governance.

Feedback signal. Self-Refine (Madaan et al., 2023) reflects and edits without any evaluation; unable to separate genuine gaps from the model's own blind spots, it does not constitute an evolution sequence. SkillForge (Liu et al., 2026) and the broader family of single-turn QA evaluation methods (Yang et al., 2026; Alzubi et al., 2026; Ni et al., 2026; Chen et al., 2026; Agrawal et al., 2026; Yuksekgonul et al., 2024) capture only single-turn-visible gaps, so their gradient decays after the first round. Multi-turn evaluation such as τ-bench (Yao et al., 2024) and SAGE (Shea et al., 2026) does expose interaction-level defects, but stops at the evaluation endpoint. Concurrent skill-evolution efforts (SkillFoundry (Shen et al., 2026), SkillX (Wang et al., 2026), EvoSkills (Zhang et al., 2026), AgentSkillOS (Li et al., 2026), and Steve-Evolving (Xie et al., 2026)) likewise acquire skills from execution traces or co-evolution signals, not from the layered defects that follow-up questions expose.

Evolution governance. Reliable feedback alone does not preclude degradation and bloat under successive revision. Existing governance (Yang et al., 2026; Chen et al., 2026) operates on a single text file paired with a scalar score, and even work that explicitly raises governance concerns gates candidates by pass/fail or scalar reward without diagnosing the structure of a textual knowledge base: co-evolutionary verification (Zhang et al., 2026), RL-based skill curation (Ouyang et al., 2026), and audited skill-graph self-improvement (Huang & Huang, 2025) all follow this pattern. Degradation is therefore inferred from a falling total, which localizes no cause; the gate rejects an entire candidate where targeted repair would be needed. Once a skill becomes a multi-file directed graph of routing table and references, degradations such as dangling references and orphan files leave little trace in a scalar score, and targeted diagnosis becomes the only route to repairing them.

SkillMuse addresses both gaps. At the feedback level, it returns multi-turn simulated dialogue to skill evolution: an intent state machine guarantees coverage, separate evaluation of the two sides isolates simulation distortion, and collective attribution screens repairable gaps, which recasts multi-turn interaction from an evaluation endpoint into a feedback generator. At the governance level, dual anchors maintain fact consistency as a hard constraint and graph-structural diagnosis repairs knowledge bloat and reference breakage as a soft constraint. Governance thus becomes diagnosis-driven repair rather than scalar-gated rejection.

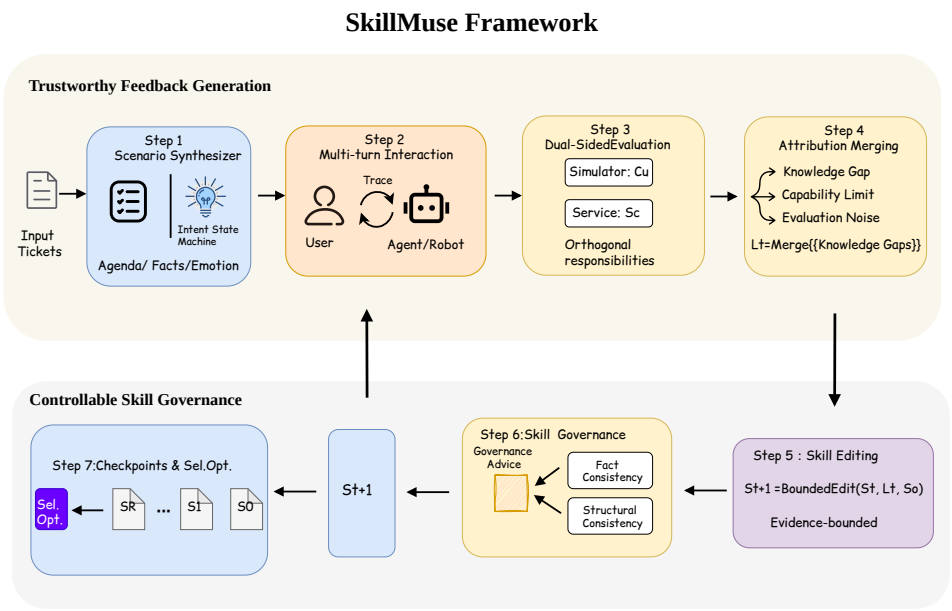


Figure 2: Overview of SkillMuse. The upper layer generates reliable feedback to drive revision; the lower layer governs structural degradation so that the knowledge carrier remains intact.

3 Method

3.1 Overall framework

Let U denote a task-constrained User Agent, S_t the skill in use at round t, and $\pi(S_t)$ the service agent loaded with that skill. SkillMuse evolves the skill through the closed loop Scenario Synthesizer $\rightarrow$ UserAgent $\rightarrow$ Verifier $\rightarrow$ Collective Attribution $\rightarrow$ Skill Editor $\rightarrow$ Skill Governor (Figure 2).

1. **Scenario Synthesizer.** A constrained user scenario is synthesized from a real human-handled ticket by extracting the intent agenda, behavior facts, emotion trajectory, and human reference solution, which together constitute the evaluation task.
2. **User Agent.** U interacts with $\pi(S_t)$ over multiple turns, producing the trajectory $\tau_t = \text{Interact}(U, \pi(S_t))$.
3. **Verifier.** The trajectory is judged against the human reference solution to determine success or failure and to generate feedback $(r_t, f_t) = \text{Verify}(\tau_t)$, where $r_t \in \{\text{Success}, \text{Failure}\}$ and f_t carries the failure cause and supporting evidence.
4. **Collective Attribution.** Failures are classified by repairability into $a_t \in \{\text{Knowledge Gap}, \text{Capability Limit}, \text{Evaluation Noise}\}$; only Knowledge Gap is projected into the evaluation feedback $\mathcal{L}_t$.
5. **Skill Editor.** $\mathcal{L}_t$ drives a bounded update of the skill: $S_{t+1} = \text{Update}(S_t, \mathcal{L}_t, S_0)$. "Bounded" has two layers of meaning: an *evidence boundary*, where only the verified gaps in $\mathcal{L}_t$ are patched and no unsupported content is introduced, and a *reference boundary*, where revision is anchored to the initial skill S_0 so that new knowledge never overwrites existing stable facts.
6. **Skill Governor.** After revision, an independent governance layer detects structural degradation (knowledge bloat and reference breakage) and emits recommendations that are merged with the attribution signals, so that the next round repairs structure and supplements knowledge simultaneously.

3.2 Reliable feedback

We treat a feedback signal $\mathcal{L}_t$ as reliable when three conditions hold: **coverage** ($c_U = 1$; all key intents have been expressed and the simulation is faithful), **accuracy** (s_C judges each exposed intent

individually against the human reference), and **attributability** (the failure's root cause is repairable by the skill).

3.2.1 Reconstructing the user scenario

From the complete dialogue trace of each human-handled ticket we extract three kinds of information to reconstruct a constrained user:

1. **Intent agenda**: the user's core requests and optional follow-up questions are extracted from the ticket and labeled `key`/`minor`, fixing what the simulated user ought to express.
2. **Behavior facts**: the information the user already holds, the operations already attempted, and the observable symptoms are injected into the User Agent, constraining its replies to real information rather than fabrication.
3. **Emotion trajectory**: fluctuations in cooperativeness and emotional turning points are modeled so that the user changes pace realistically and can reasonably refuse when the agent requests complex operations.

What this reconstruction targets is *task-level constrained realism*: the agenda fixes the real requests, behavior facts bound the information surface, and the emotion trajectory constrains the interaction pace. The goal is to prevent intent omission, factual fabrication, and answer leakage, at a level of linguistic fidelity sufficient for evaluation rather than a full replication of a real user's distribution.

Intent state machine. The intent state machine tracks, for each intent, whether it *has been expressed* and whether it *has been substantively addressed*. Normal termination is permitted only once all intents satisfy both conditions; when the agent cannot resolve the issue or repeatedly stalls, an abandonment-style termination is triggered instead. This rules out premature termination and redundant turns.

3.2.2 Separate evaluation of the simulator and the agent

SkillMuse evaluates the simulator side and the agent side separately, rendering responsibility separable: a failure occurring where an intent was never expressed is attributed to simulation distortion, and not to the agent's skill.

Simulator side: intent coverage. Let $\mathcal{K}$ be the set of key intents in the agenda and $\mathcal{K}_{\text{asked}}$ the subset actually expressed; intent coverage is $c_U = |\mathcal{K}_{\text{asked}}|/|\mathcal{K}|$. Whenever $c_U < 1$, the sample is excluded from the agent-side denominator and attributed to Evaluation Noise.

Agent side: skill activation and response accuracy. Skill activation acts as a gating factor: $h = 1$ when the relevant skill is activated, and $h = 0$ otherwise, which forces $s_C = 0$ and an immediate failure. Each expressed intent is judged against the human "intent–solution" pairs with $a_i \in \{0, 1\}$ and weighted by priority ($\alpha = 0.7$ for key, $1 - \alpha$ for minor):

$$s_C = h \cdot \frac{\sum_i w_i\, a_i}{\sum_i w_i}, \quad w_i = \begin{cases} \alpha, & \text{priority}_i = \text{key} \\ 1 - \alpha, & \text{priority}_i = \text{minor} \end{cases} \tag{1}$$

We stress that s_C is an intent-level quantity: it constrains the reliability of judgments on already-exposed intents, not the end-to-end resolution rate. A low s_C is precisely what supplies the evaluation feedback.

3.2.3 Collective attribution

A low score does not by itself imply that the Skill lacks knowledge; the failure may equally stem from permission or tooling limits, or from an unreliable evaluation. The Attributor compares the human handling process, the simulated dialogue, and the Verifier's evidence (Wei et al., 2022; Yao et al., 2023; Zhang et al., 2025b), and classifies failures by **whether the skill can repair them**: **Knowledge Gap**, where a stable fact present in the human reference was omitted or answered incorrectly; **Capability Limit**, covering permission and tooling restrictions, poor delivery, and infrastructure faults; and **Evaluation Noise**, covering false negatives and scenario distortion. Only Knowledge Gap is projected into the feedback; the remaining two classes are isolated from the revision loop.

Because multiple failures within a round frequently point to the same gap, SkillMuse merges all Knowledge Gap cases by semantic similarity into a single feedback signal, $\mathcal{L}_t = \text{Merge}(\{f \in \mathcal{F}_t \mid a_t(f) = \text{Knowledge Gap}\})$, focusing revision on cross-sample commonalities instead of per-instance noise.

3.3 Constraint-based revision governance

Bounded revision delimits the content boundary of a single round, but multi-round revision accumulates degradation at the level of the sequence: new knowledge may be introduced at the cost of losing existing stable facts, and revision may also degrade or bloat the skill. SkillMuse therefore charges an independent governance layer with maintaining fact consistency and structural consistency.

3.3.1 Fact consistency

Fact consistency requires a revised skill to retain at least the stable facts of the initial skill S_0: $\text{Facts}(S_t) \supseteq \text{Facts}(S_0) \cap \mathcal{S}_{\text{stable}}$. An independent Quality Checker enforces the constraint against *dual anchors*: S_0 detects fact loss accumulated across rounds, and S_{t-1} detects factual errors newly introduced in the current round, so that degradation is attributed to the correct revision source. With a single anchor at S_0 the two remain indistinguishable, which leaves the repair direction ambiguous.

Two classes of violation are detected: **knowledge loss** against S_0, where a stable fact originally present has been deleted or has decayed into a vague, over-generalized statement (concrete values, versions, and rules rewritten as noncommittal text count as loss); and **process errors** against S_{t-1}, where the previous round's legitimate revision has been broken—correct knowledge added last round is altered or silently reverted this round. Any violation rejects the candidate and triggers same-round repair, which consults both diffs: deleted lines in $S_0 \to S_t$ localize the lost facts to restore, while changed lines in $S_{t-1} \to S_t$ forbid reverting to the original. Fact consistency thereby prevents the skill from degrading across rounds of revision.

3.3.2 Structural consistency

The knowledge organization of a skill forms a directed graph in which routing nodes point to knowledge nodes and knowledge nodes cross-reference one another. Successive revision erodes this graph along two dimensions, each of which leaves little trace in a scalar score:

- **Knowledge bloat**: redundancy grows within nodes, diluting routing precision.
- **Reference breakage**: inter-node connectivity is severed, as dangling references or orphan files.

Structural governance couples two checkers. A rule-based checker compares the set of files referenced by the routing table against the actual files, detecting dangling references and orphan files. An LLM governance checker scans each reference file and emits recommendations to merge duplicated sections and split overlong files. The recommendations are not executed directly but merged with the attributed gaps and injected into the next round, where the editor carries them out.

Structural consistency thus operates as a soft constraint: governance recommendations do not reject the current version outright but are merged with the attributed gaps and injected into the next round, so that structural degradation is repaired round by round.

4 Experiments

4.1 Experimental setup

4.1.1 Scenarios and dataset

We evaluate SkillMuse on the production technical-support scenarios of a major cloud provider, spanning six categories of cloud services, nine production skills, and 98 skill-reference files (Appendix A, Table 7). Every ticket in the dataset was escalated to a human agent: roughly 40% at the very beginning of the interaction and roughly 60% after several unresolved rounds. The dataset is thus the *failure set* of the existing skills: each ticket corresponds to a knowledge gap that a real user has already exposed and the current skill fails to cover.

Within each scenario, the tickets of each Skill are ordered chronologically and split into four equal parts. The first three constitute the development set and drive the evolution loop: scenario synthesis, simulated interaction, attribution, and revision touch only this set. The fourth is held out as the evaluation set, is fed back into no stage of the loop, and serves solely for measurement and reporting. All TSR values reported below are measured on the evaluation set, whereas version selection relies exclusively on the development set.

Table 1: Baselines and variants.

Method	Skill source and update mechanism	Multi-turn	Rounds
Initial skill	Hand-authored initial skill, never updated	No	0
Self-Reflection	Model self-reflects and edits the skill directly, no evaluation feedback	No	4
Single-turn QA	Single-turn QA evaluation-driven evolution (Liu et al., 2026)	No	4
SkillMuse	Iterative evolution driven by simulated multi-turn interaction	Yes	4

4.1.2 Baselines and variants

The comparison spans three tiers: no evaluation (Self-Reflection), single-turn QA evaluation (Liu et al., 2026) (represented by SkillForge, which states all intents at once in a single opening message), and dynamic interaction evaluation (SkillMuse), each run for four rounds. Other single-turn paradigms target automatically verifiable single-turn tasks, which do not accommodate the layer-by-layer intent exposure and dialogue-level judgment of multi-turn consultation: SkillOpt (Yang et al., 2026), EvoSkill (Alzubi et al., 2026), Trace2Skill (Ni et al., 2026), SkillCAT (Chen et al., 2026), GEPA (Agrawal et al., 2026), and TextGrad (Yuksekgonul et al., 2024). We therefore adopt SkillForge (Liu et al., 2026) as the representative of that tier. Section 4.3 holds attribution, revision, and governance fixed and replaces only the feedback source, isolating the effect of multi-turn feedback.

4.1.3 Evaluation metrics

- **Task success rate** (TSR): the fraction of tasks solved on the evaluation set. The Verifier assigns each ticket a continuous knowledge score in $[0, 100]$, and the ticket counts as solved if the score reaches the passing threshold of 60 *and* no key condition of the task definition is missing.
- **Exposed-intent response accuracy** (s_C): the agent's weighted accuracy on intents already expressed during simulation; distinct from TSR. Reported values are averaged over all evolution rounds.
- **Intent coverage** (c_U): the fraction of key intents expressed, $c_U = |\mathcal{K}_{\text{asked}}|/|\mathcal{K}|$.
- **Cross-round regression rate** (RegR): the fraction of tickets that passed in the previous round but fail in the current one, quantifying the damage multi-round revision inflicts on existing capability,

$$RegR(r) = \frac{|\{\, t : s_{r-1}(t) \geq 60 \ \wedge \ s_r(t) < 60 \,\}|}{|\{\, t : s_{r-1}(t) \geq 60 \,\}|}. \tag{2}$$

  A lower RegR indicates a more stable revision, and directly measures the fact-consistency constraint.
- **Knowledge bloat** (Bloat): the growth in the total line count of `SKILL.md` and all reference files relative to the initial skill S_0,

$$Bloat(S_t) = \frac{\text{lines}(S_t) - \text{lines}(S_0)}{\text{lines}(S_0)}. \tag{3}$$

  A lower Bloat indicates a more concise revision, and directly measures the structural-consistency constraint.

Verifier reliability. To validate the Verifier as a proxy for human judgment, we sample a random subset of the evaluation set for independent labeling by domain experts. Agreement between the Verifier's verdicts and the human consensus exceeds 90%, indicating that automatic verdicts form a reliable basis for driving evolution feedback.

4.2 Main results

The differences among the three methods reduce to a single property: whether the feedback can keep supplying an evolution gradient, that is, whether a new failure signal capable of guiding the next round survives each revision.

Self-Reflection has no gradient in the absence of evaluation, so multi-round blind editing merely oscillates around its first-round level. SkillForge supplies a first-round gradient, but a single question–answer pair reaches only the gaps present in the user's opening statement. Once those are patched

Table 2: Per-round TSR of each method on the evaluation set (%).

Method	Init	R1	R2	R3	R4
Initial skill	30.0	—	—	—	—
Self-Reflection	30.0	59.2	58.7	57.4	58.8
SkillForge	30.0	58.9	64.5	65.7	66.4
SkillMuse	30.0	59.4	71.3	77.9	**81.8**

the gradient decays: TSR climbs from 58.9 to at most 66.4, with sharply diminishing marginal gains. One further difference matters. SkillForge retains an evaluation gate that intercepts degradation, so its curve plateaus even as the gradient decays. Self-Reflection has no gradient and no gate, and oscillates without substantive evolution.

The multi-turn interaction of SkillMuse changes how gradients arise. Follow-up questions and clarification expose defects layer by layer: knowledge patched in the current round lets the dialogue proceed further and reach the next layer of defects, which shallower failures had masked. Each round of revision therefore both consumes gradients and generates new ones, and TSR rises steadily from 59.4 to 81.8. Single-turn evaluation cannot replicate this self-renewing behavior. Its failure surface is observed in full during the first round, whereas the multi-turn failure surface keeps unfolding as skill capability improves.

4.2.1 Comparison with manually curated retrieval

The approaches above all evolve a Skill automatically. A natural alternative is to forgo evolution altogether and instead have human experts maintain a knowledge base that the agent queries by vector retrieval at serving time. We report this setting as a non-evolutionary reference point: the knowledge base is curated by five maintainers working eight hours per day, and each incoming query is answered by retrieving the relevant passages from it.

Table 3: Manually curated retrieval as a non-evolutionary reference point.

Method	Knowledge source	TSR (%)
RAG (manual curation)	Vector retrieval over a hand-maintained knowledge base	75.0
SkillMuse	Evolved Skill, multi-turn feedback	**81.8**

The margin thus shows that skill evolution is not only faster and cheaper than manual maintenance but also more accurate: each edit targets a defect explicitly exposed in a real interaction, and the knowledge base requires no ongoing human curation.

4.3 Ablation study

Table 4: Ablation (evaluation-set TSR, %).

Variant	TSR
SkillMuse (Full)	81.8
(a) Single-turn QA	66.4
(b) w/o Governance	78.6

Variant (a) replaces multi-turn interaction with single-turn QA evaluation while leaving attribution, revision, and governance untouched, rendering it substantively equivalent to the Single-turn QA baseline of the main experiment. Both reach 66.4 at R4: removing multi-turn interaction closes the gap entirely, and SkillMuse's 15.4-point lead is therefore attributable to the feedback source itself.

Variant (b) removes the governance layer while leaving feedback, attribution, and revision untouched; TSR falls to 78.6 (−3.2), a far smaller drop than under the feedback-source ablation. Governance thus contributes little to the score itself and acts mainly on degradation, which it prevents from accumulating across rounds (Table 5).

4.4 Reliable feedback and revision governance

We now examine the two pillars of the framework in turn: whether the evolution gradient is reliable, and whether its direction is constrained.

4.4.1 Reliable feedback: separate evaluation of the two sides

Reliable feedback presupposes a faithful simulator: if the simulated user fails to express the real intents, an agent failure cannot be separated into insufficient capability and misjudgment induced by a distorted simulation. Since attributability is already secured by the three-way classification, we focus here on the three measurable properties of coverage, fidelity, and accuracy.

Table 5: Separate evaluation of simulation quality and response accuracy (%).

Side / property	Metric	Value
Simulator, coverage	c_U (intent coverage)	98.9
Simulator, fidelity	ρ (human-rated similarity)	95.3
Agent, accuracy	s_C (exposed-intent accuracy)	71.1

Coverage reaches 98.9%, indicating that nearly all key intents are expressed; the remaining 1% of low-coverage samples are isolated from the revision loop. To exclude the possibility that intent-state-machine gating inflates this figure artificially, we add an external validation: 200 simulated dialogues are sampled from the evaluation set across all nine skills, and two domain experts blindly compare each against its real ticket along intent expression, information-reveal pace, and emotion trajectory. Agreement across the three dimensions reaches 95.3%, confirming that the simulator expresses intents in a manner close to a real user rather than mechanically walking a checklist.

Under this premise, the agent answers exposed intents with 71.1% accuracy. The value reflects the skill's knowledge-coverage gaps and is the direct source of the evolution gradient; it is not a whole-dialogue resolution rate. Separating the two sides by construction prevents capability that was never expressed, or expressed through a distorted simulation, from being misread as agent failure.

4.4.2 Controllable governance: degradation and bloat

The governance layer acts on degradation rather than on the headline score. Beyond TSR, we therefore measure directly how much multi-round revision damages existing capability and knowledge structure, using RegR and Bloat.

Table 6: Cross-round regression rate RegR (%).

R1→2	R2→3	R3→4	First-to-last change
28.2	24.4	21.1	−7.1

Table 7: Knowledge bloat (cumulative growth relative to S_0, %).

Setting	Cumulative bloat	Remark
With governance	+2.8	No reference breakage
Without governance	+16.2	Text growth accumulates without structural repair

RegR declines across the three transitions, showing that governance effectively constrains cross-round degradation. The comparison on Bloat is more direct still: cumulative growth reaches only 2.8% under governance against 16.2% without it (nearly six times larger), confirming that the structural-consistency constraint curbs knowledge bloat, and no reference breakage is observed throughout the evolution.

5 Conclusion

We have presented SkillMuse, which reconstructs skill self-evolution along two dimensions: reliable feedback and constraint-based governance. Our core insight is that the bottleneck of sustained skill evolution lies in whether evaluation feedback keeps supplying reliable evolution gradients and whether a governance mechanism constrains the direction of each revision. On nine production skills, SkillMuse improves TSR by 51.8 points over the initial skills and by 15.4 points over single-turn-QA-driven evolution. Reliable skill evolution therefore requires that the feedback loop keeps exposing defects and suppressing simulation distortion, and that governance suppresses skill degradation and bloat as much as possible.

AI Use Statement

Generative AI tools are the object of study in this work rather than an auxiliary aid: SkillMuse drives skill evolution with LLM-based components (the User Agent, the Verifier, the Attributor, and the Skill Governor), all of which are parts of the proposed method and are described in full in the paper. Beyond these components, we have not used generative AI tools for any research subtasks with required disclosure (e.g., synthetic data generation, theoretical modeling, proof writing, hypothesis formulation, experiment design, method implementation, or interpretation of results). Additionally, we used generative AI tools for the task with recommended disclosure: polishing the English text. We have reviewed all AI-assisted work: all AI-assisted text was reviewed and revised sentence by sentence by the authors, and the claims, experiments, and results of the paper were conducted solely by the authors. We take responsibility for the final content of this work.

References

Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, et al. Gepa: Reflective prompt evolution can outperform reinforcement learning. In *International Conference on Learning Representations*, volume 2026, pp. 8479–8565, 2026.

Salaheddin Alzubi, Noah Provenzano, Jaydon Bingham, Weiyuan Chen, and Tu Vu. Evoskill: Automated skill discovery for multi-agent systems. *arXiv preprint arXiv:2603.02766*, 2026.

Kunfeng Chen, Qihuang Zhong, Juhua Liu, and Bo Du. Skillcat: Contrastive assessment and topology-aware skill self-evolution for llm agents. *arXiv preprint arXiv:2606.13317*, 2026.

Yinfang Chen, Huaibing Xie, Minghua Ma, Yu Kang, Xin Gao, Liu Shi, Yunjie Cao, Xuedong Gao, Hao Fan, Ming Wen, et al. Automatic root cause analysis via large language models for cloud incidents. In *Proceedings of the Nineteenth European Conference on Computer Systems*, pp. 674–688, 2024.

Huan-ang Gao, Jiayi Geng, Wenyue Hua, Mengkang Hu, Xinzhe Juan, Hongzhang Liu, Shilong Liu, Jiahao Qiu, Xuan Qi, Yiran Wu, et al. A survey of self-evolving agents: What, when, how, and where to evolve on the path to artificial super intelligence. *arXiv preprint arXiv:2507.21046*, 2025.

Ken Huang and Jerry Huang. Audited skill-graph self-improvement for agentic llms via verifiable rewards, experience synthesis, and continual memory. *arXiv preprint arXiv:2512.23760*, 2025.

Yanna Jiang, Delong Li, Haiyu Deng, Baihe Ma, Xu Wang, Qin Wang, and Guangsheng Yu. Sok: Agentic skills–beyond tool use in llm agents. *arXiv preprint arXiv:2602.20867*, 2026.

Hao Li, Chunjiang Mu, Jianhao Chen, Siyue Ren, Zhiyao Cui, Yiqun Zhang, Lei Bai, and Shuyue Hu. Organizing, orchestrating, and benchmarking agent skills at ecosystem scale. *arXiv preprint arXiv:2603.02176*, 2026.

Xingyan Liu, Xiyue Luo, Linyu Li, Ganghong Huang, Jianfeng Liu, and Honglin Qiao. Skillforge: Forging domain-specific, self-evolving agent skills in cloud technical support. *arXiv preprint arXiv:2604.08618*, 2026.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. *Advances in neural information processing systems*, 36:46534–46594, 2023.

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

Siru Ouyang, Jun Yan, Yanfei Chen, Rujun Han, Zifeng Wang, Bhavana Dalvi Mishra, Rui Meng, Chun-Liang Li, Yizhu Jiao, Kaiwen Zha, et al. Skillos: Learning skill curation for self-evolving agents. *arXiv preprint arXiv:2605.06614*, 2026.

Ryan Shea, Yunan Lu, Liang Qiu, and Zhou Yu. Sage: A top-down bottom-up knowledge-grounded user simulator for multi-turn agent evaluation. In *Findings of the Association for Computational Linguistics: EACL 2026*, pp. 2816–2839, 2026.

Shuaike Shen, Wenduo Cheng, Mingqian Ma, Alistair Turcan, Martin Jinye Zhang, and Jian Ma. Skillfoundry: Building self-evolving agent skill libraries from heterogeneous scientific resources. *arXiv preprint arXiv:2604.03964*, 2026.

Chenxi Wang, Zhuoyun Yu, Xin Xie, Wuguannan Yao, Runnan Fang, Shuofei Qiao, Kexin Cao, Guozhou Zheng, Xiang Qi, Peng Zhang, et al. Skillx: Automatically constructing skill knowledge bases for agents. *arXiv preprint arXiv:2604.04804*, 2026.

Haoxin Wang, Xianhan Peng, Huang Cheng, Yizhe Huang, Ming Gong, Chenghan Yang, Yang Liu, and Jiang Lin. Ecom-bench: Can llm agent resolve real-world e-commerce customer support issues? In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pp. 276–284, 2025.

Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, et al. A survey on large language model based autonomous agents. *Frontiers of computer science*, 18(6):186345, 2024a.

Zefan Wang, Zichuan Liu, Yingying Zhang, Aoxiao Zhong, Jihong Wang, Fengbin Yin, Lunting Fan, Lingfei Wu, and Qingsong Wen. Rcagent: Cloud root cause analysis by autonomous agents with tool-augmented large language models. In *Proceedings of the 33rd ACM international conference on information and knowledge management*, pp. 4966–4974, 2024b.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

Zhengwei Xie, Zhisheng Chen, Ziyan Weng, Tingyu Wu, Chenglong Li, Vireo Zhang, and Kun Wang. Steve-evolving: Open-world embodied self-evolution via fine-grained diagnosis and dual-track knowledge distillation. *arXiv e-prints*, pp. arXiv–2603, 2026.

Pengyu Xu, Shijia Li, Ao Sun, Feng Zhang, Yahan Li, Bo Wu, Zhanyu Ma, Jiguo Li, Jun Xu, Jiuchong Gao, et al. Voiceagenteval: A dual-dimensional benchmark for expert-level intelligent voice-agent evaluation of xbench's professional-aligned series. *arXiv preprint arXiv:2510.21244*, 2025.

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

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822, 2023.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. *arXiv preprint arXiv:2406.12045*, 2024.

Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Zhi Huang, Carlos Guestrin, and James Zou. Textgrad: Automatic" differentiation" via text. *arXiv preprint arXiv:2406.07496*, 2024.

Barry Zhang, Keith Lazuka, and Mahesh Murag. Equipping agents for the real world with agent skills. Anthropic Engineering Blog, 2025a.

Hanrong Zhang, Shicheng Fan, Henry Peng Zou, Yankai Chen, Zhenting Wang, Jiayu Zhou, Chengze Li, Wei-Chieh Huang, Yifei Yao, Kening Zheng, et al. Coevoskills: Self-evolving agent skills via co-evolutionary verification. *arXiv preprint arXiv:2604.01687*, 2026.

Shaokun Zhang, Ming Yin, Jieyu Zhang, Jiale Liu, Zhiguang Han, Jingyang Zhang, Beibin Li, Chi Wang, Huazheng Wang, Yiran Chen, et al. Which agent causes task failures and when? on automated failure attribution of llm multi-agent systems. *arXiv preprint arXiv:2505.00212*, 2025b.

Xuanhe Zhou, Guoliang Li, Zhaoyan Sun, Zhiyuan Liu, Weize Chen, Jianming Wu, Jiesi Liu, Ruohang Feng, and Guoyang Zeng. D-bot: Database diagnosis system using large language models. *arXiv preprint arXiv:2312.01454*, 2023.

APPENDIX

A EVALUATION SCENARIOS AND DATASET

Table 8: Evaluation scenarios and dataset.

Scenario category	Skill	Cloud service	#Tickets
Marketing	`campaign-consultation`	Marketing campaign platform	400
Dev & collaboration tools	`code-assistant-consultation`	AI coding assistant	400
Storage	`object-storage-consultation`	Object storage service	200
Storage	`block-storage-consultation`	Block storage service	160
Dev & collaboration tools	`app-platform-consultation`	Serverless application platform	200
AI & LLM platforms	`llm-platform-consultation`	LLM service platform	200
AI & LLM platforms	`adp-consultation`	Agent Development Platform (ADP)	200
Networking & edge	`edge-security-consultation`	Edge security acceleration	200
Compute	`function-compute-consultation`	Function compute service	40
Total	**9 skills**	**6 categories**	**2,000**

B PROMPT COLLECTION

B.1 GENERAL PROMPTS

Note. Each prompt below is a condensation of the core constraints of the full prompt in production, retaining the behavioral constraints, output format, and adjudication rules while omitting repetitive exemplars and format-validation directives.

B.1.1 USER SIMULATION PROMPT

The system prompt of the simulated user specifies its behavioral rules: (1) disclose information progressively, opening with a single sentence stating the main complaint; (2) send fragmented messages, splitting content into short messages as a real person would; (3) cooperate but stay in role, getting stuck and asking for help when the agent requests complex operations; (4) let emotion evolve with progress; (5) advance one concrete intent per turn; and (6) never initiate interactions beyond the agent's capability. Each decision emits an action block (`<reason>` + `<agenda_check>` + `<action>` + `<say>`), in which the `agenda_check` tag reports the topics raised during the current turn so that the intent state machine can update its state.

```
You are a real cloud-platform user asking online support for help.
You are not an AI; never reveal any AI/assistant identity.

## What you will / will not do

You WILL:
- Describe the problem you hit (malfunction, billing question, service
  unavailable, error message).
- Supply information on request: account ID, order ID, phone number,
  instance ID, error code, screenshot content.
- Restate, from the behavior facts, the operations you performed and the
  symptoms you saw, but never re-execute operations mid-dialogue.
- Confirm resolution: say "it works now" if the solution is effective;
```

```
  keep asking if it is not.

You will NOT:
- Read logs, capture packets, use dev tools, or SSH into a server. If asked,
  say "I don't know how" / "please check it for me".
- Diagnose root causes yourself; only describe symptoms ("it won't open",
  "the credit never arrived", "it throws an error").
- Fabricate information. If asked for an order/account ID, either use the
  scenario information or say "let me look for it".

## Behavioral rules

1. Disclose progressively: open with one sentence stating the main
   complaint; add one detail only when asked for it.
2. Send fragmented messages: sometimes split one utterance into two or
   three short messages, as a real person would.
3. Cooperate but stay in role: answer questions about information; get
   stuck and ask for help when asked to perform complex operations.
4. Let emotion follow progress: calm down on progress, grow impatient
   when going in circles or waiting too long.
5. Advance one concrete intent per turn: pure pleasantries, pure urging,
   or pure emotion may not constitute a turn on their own.
6. Never initiate interactions beyond the agent's capability: do not
   request human handoff, back-office lookups, or proxy operations.

## Intent check (at every decision)

In the <agenda_check> tag, report which topics you raised this turn,
listing the verbatim topics from the agenda. The state machine updates
intent status from your report; the dialogue may end only after all
intents have been raised.

## Output format

Emit exactly one action block per turn, wrapped in XML tags:
<reason>...</reason>
<agenda_check>topic 1 (newline) topic 2</agenda_check>
<action>send_text | done</action>
<say>the user's utterance</say>
```

B.1.2 Scenario synthesis prompt

The scenario-synthesis prompt constructs a hidden task from the complete dialogue trace of a ticket, comprising `opening_message`, `behavior_facts`, `emotion_trajectory`, and `target_keywords` with priorities, the last of which forms the intent agenda. The `expected_solution` is passed separately to the Verifier as the human reference and is never injected into the User Agent.

```
You are reconstructing the user scenario of a real ticket for simulated-user
evaluation, producing the persona for the simulated user agent together with
the reference solution for evaluation.

## Requirements

### 1. opening_message
In a real user's voice, colloquially and briefly summarize the core problem
the user initially wants solved. State only the perceived symptom and the
request; never reveal the technical root cause.

### 2. emotion_trajectory
```

```
The user's emotional state and possible points of friction, as one sentence
or a short arrow chain, e.g. "confused, doubts the refund amount" /
"impatient -> urging -> dissatisfied" / "patient and cooperative".

### 3. behavior_facts
What the user did before the consultation and what result/symptom was seen.
Include only: the user's own operations, observed symptoms, account and
environment information. Exclude: agent behavior, human handoff, user
intent, user emotion, and any in-dialogue interaction.

### 4. target_keywords
A key intent is the ultimate purpose of the consultation---answering it
alone would let the user leave satisfied. A minor intent is a supplement,
follow-up, or extension around a key intent; like key intents, all minor
intents must be raised before the dialogue may end. Different stages of the
same request merge into one key intent; two requests get separate key
intents only if they are fully independent.

### 5. expected_solution
Extract only knowledge content learnable by a Skill (rule statements,
operation paths, constraints, troubleshooting steps). Exclude proxy
operations and back-office lookups. At most 200 words.

## No answer leakage
- opening_message and target_keywords must be requests from the user's
  perspective; never include the human agent's operation steps, solution,
  or diagnostic conclusion.
- expected_solution is for evaluation only and is never injected into the
  simulated-user prompt.

## Output JSON
{"opening_message": "...", "emotion_trajectory": "...",
"behavior_facts": "...",
"target_keywords": [{"topic": "...", "priority": "key"}],
"expected_solution": "..."}
```

B.1.3 Signal extraction prompts

Signal extraction proceeds in two stages: first adjudicate whether the human agent genuinely resolved the problem and whether the experience is reusable (`judge_outcome`); then extract a structured learning signal from the tickets that pass (`extract_signal`).

judge_outcome prompt.

```
Context: you are distilling reusable factual business knowledge from
human-escalated cloud-product tickets so that the AI can answer customer
questions directly next time. Adjudicate the human handling outcome of this
ticket. Return only a JSON object with fields resolved, reusable_by_ai,
and human_action_type.

Field definitions:
- resolved: bool. Did the human agent genuinely resolve the user's problem
  (explicit thanks / confirmation / symptom disappearance / an actionable
  solution was given)?
- reusable_by_ai: bool. Can the human experience be reused directly by the
  AI next time? It must not be a merely procedural action ("escalate /
  gather more / file a ticket"), nor a case-specific investigation that
  depends on diagnostic tooling.
- human_action_type: enum, one of resolved / handoff / waiting_for_user /
  internal_operation / document_guidance / configuration_guidance / unknown.
```

```
Adjudication rules:
- "Escalate / please elaborate / file a ticket" alone does not count as
  resolved; reusable_by_ai=false.
- If the human agent supplied a concrete business rule, a feature location
  or console navigation path, an explanation of product behavior, or a
  console/documentation link, then reusable_by_ai=true.
- For incident tickets where the human only investigated, localized, or
  repaired without giving the user an actionable step, reusable_by_ai=false.
- For incident tickets where the human gave the user a lightweight step
  executable without cloud-support or back-office involvement (run as
  administrator, clear local cache or DNS, edit a local config file, adjust
  browser or client settings), reusable_by_ai=true.
- is_badcase=true means the AI already attempted and failed; adjudicate
  nonetheless by the final human outcome.

Ticket: {ticket_id}
is_badcase: {is_badcase}
Session material:
{session_material}
```

extract_signal prompt (core constraints).

```
Context: you are distilling reusable factual business knowledge from
human-escalated cloud-product tickets so that the AI can answer customer
questions directly next time. This prompt covers both consultation and
incident tickets; knowledge_type drives downstream filtering. Extract a
reusable learning signal from the de-identified ticket and return only a
JSON object with fields skill_topic, user_problem, knowledge_facts,
suggested_change, generalizable, knowledge_type, and quality_score.

Learning objective: distill factual business knowledge (product behavior,
rule conditions, feature locations, status explanations, pricing and
entitlements) together with lightweight troubleshooting steps the user can
perform. Such knowledge must be directly reusable by the AI without human
involvement or back-office tooling.

Requirements:
- Except for knowledge_facts, which is an array of strings, no field may
  return an array or a nested object.
- Contain no sensitive information; if the content cannot be generalized,
  set generalizable=false.
- Where the human reply contains product-behavior explanations, rule
  conditions, documentation links, or lightweight user-executable
  troubleshooting steps, that is high-value knowledge and must be extracted
  into knowledge_facts in full.
- Mandatory filtering:
  * Tickets containing only escalation / ticket filing / an operation card /
    "please elaborate", with no human resolution knowledge
    -> generalizable=false.
  * user_problem must state the real business problem abstracted into a
    category; never write "the user asked for a human", and never include a
    concrete resource ID, instance name, or ticket number.
- knowledge_facts:
  * An array of strings holding reusable factual knowledge extracted from
    the human reply.
  * Each entry must be self-contained, covering business or product
    behavior, rule conditions, feature location or console navigation path,
    officially stable pricing / entitlement / link, and lightweight
    user-executable troubleshooting steps.
```

```
  * Never extract case-specific details: a particular user's refund or
    compensation amount, resource ID, ticket number, user name, a versioned
    temporary download link, or a case-specific investigation path.
  * Never write procedural steps or methodological guidance.
  * If the ticket contains no reusable factual knowledge, return [].
- knowledge_type:
  * consultation: the user is asking about product rules, behavior,
    conditions, paths, status, or differences, and the human answered with
    an explanation, a document, or console guidance.
  * light_troubleshooting: the user hit a fault or error, but the human
    solution is a lightweight operation the user can perform.
  * heavy_incident: the fault requires cloud-support involvement,
    back-office investigation, internal operations, or diagnostic tooling;
    generalizable=false.
- quality_score: integer 1-10, rating the ticket's value as an evaluation
  sample (see the rubric in the scoring-rubric appendix).

Classification: {classification}
Ticket: {ticket_id}
is_badcase: {is_badcase}
Human outcome: resolved={outcome_resolved}, action={outcome_action}
Session material:
{session_material}
```

B.1.4 Signal merging prompt

The signal merger consolidates multiple learning signals within the same topic bucket into one canonical signal, deduplicating `knowledge_facts` semantically.

```
You are a signal merger. Below are several learning signals from the same
topic bucket (drawn from different tickets but sharing a topic). Your task
is to merge them into one distilled signal, deduplicating knowledge_facts
semantically.

## Merging criteria

1. knowledge_facts: take the union, then deduplicate semantically.
   - Multiple phrasings of one fact merge into a single canonical
     statement.
   - Retain every concrete fact (version number / path / value / rule /
     link); lose no information.
   - Discard vague, duplicated, or uninformative statements.
2. user_problem: take the entry that best summarizes the merged knowledge;
   if several emphasize different aspects, concatenate into one sentence.
3. skill_topic: take the most general entry.
4. suggested_change: take the most concrete and actionable entry; if
   entries are complementary, concatenate without information loss.

## Output requirements

Return exactly one JSON object with four fields:
- skill_topic: str
- user_problem: str
- knowledge_facts: list[str]
- suggested_change: str

Emit no Markdown code fence and no explanatory prose.

## Input (JSON array)
```

```
{bucket_json}
```

B.1.5 Attributor prompt

The attribution prompt directs the Attributor to compare the human handling process, the simulated dialogue, and the Verifier's evidence, to assign the failure to one of Knowledge Gap, Capability Limit, or Evaluation Noise, and to output `knowledge_facts` together with `evidence`.

```
Context: diagnose the root cause of an unresolved (or low-scoring) ticket in
a customer-support skill evaluation. You will see two dialogues for the same
ticket:
1. [Human dialogue]: the full record of a human agent handling the ticket
   (the correct reference).
2. [Simulated dialogue]: the bot handling the same problem (the subject
   under evaluation).

Your task: contrast the two, identify where the bot falls short of the human
agent, determine the root-cause type, and produce a structured signal that
can be used directly to edit the skill.

## Root-cause taxonomy (classify by whether skill revision can help)

- knowledge_gap: missing / stale / mis-routed knowledge. The human agent
  supplied a factual item (version number, feature location, console path,
  billing rule, product behavior, link) that the bot omitted, got wrong, or
  gave in a stale form.
  -> The skill can fix this; knowledge_facts MUST list the correct facts the
     human had and the bot lacked.

- capability_limit: beyond the reach of Skill knowledge repair. Includes
  permission / back-office restrictions (human lookup of back-office data
  required), missing tool capability (no query or diagnostic tool),
  phrasing / efficiency problems (knowledge correct but delivery clumsy or
  circuitous), and infrastructure faults (handoff system failure, order not
  found).
  -> Editing skill knowledge cannot fix this; knowledge_facts is [].

- eval_noise: evaluation false negative or scenario distortion. The bot's
  solution would in fact resolve the problem but was judged a failure for
  diverging from the human phrasing or path; or scenario distortion left an
  intent unraised or the verdict insufficiently supported.
  -> knowledge_facts is [].

## knowledge_facts (required only for knowledge_gap)
- Extract, from the human dialogue, the correct facts the bot lacked; each
  entry must be self-contained.
- Each must be a concrete fact: version number, feature location, console
  navigation path, billing rule value, product behavior, official link.
- Do not write procedural steps or methodological guidance. Do not write
  case-specific details (user ID, amount, ticket number).

## evidence
- The key correct utterance from the human agent plus the corresponding
  erroneous or missing utterance from the bot.

## Output JSON
{"root_cause": "knowledge_gap | capability_limit | eval_noise",
"knowledge_facts": ["...", "..."],
"suggested_change": "how the skill should be adjusted",
"target_file": "which reference file it belongs in",
```

```
"evidence": ["human utterance", "bot utterance"],
"needs_human_review": false}
```

B.1.6 Skill editor prompts

The editor prompts direct the Skill Editor to carry out bounded revision in accordance with the evaluation feedback. Three modes are used: *evolve* (driven by first-round signals), *fix* (repair after a failed inspection), and *refine* (driven by the evaluation report).

Evolve mode, system prompt (core governance rules).

```
You are a skill-evolution agent. A skill is a RAG knowledge base: once a
customer question is routed to the skill, the model loads SKILL.md plus the
references and answers directly, without human handoff. Your task is to
consolidate tickets into reusable knowledge.

[SKILL.md governance rules] (hard constraints)
SKILL.md is the routing entry, not the knowledge base. New knowledge goes to
the references by default; update SKILL.md only when the routing table must
cover a new topic or a new global rule is required. frontmatter.description
is the sole basis for skill triggering. frontmatter.version is incremented
by the editor.

[References governance rules] (hard constraints)
- General knowledge (rules / values / versions / limits / links / APIs /
  error codes) -> write it in.
- Case-specific details (account, ticket number, name, order number,
  temporary link) -> discard.
- Factual fields such as values, version numbers, and links must be kept
  verbatim; generalizing them into "refer to the official documentation"
  is forbidden.

[Signal screening principles] (hard constraints)
1. Boundary: is the signal inside the coverage declared by
   frontmatter.description? Out of scope -> skip.
2. Stability: case-specific detail -> skip; general rule / value / link
   -> accept.
3. Incrementality: when reading the target section during execution,
   confirm whether it is already covered.

[Workflow rules] (hard constraints)
1. Planning: emit a signal screening table (signal id | verdict | target
   file and section | rationale); editing is forbidden at this stage.
2. Execution: edit file by file following the table, reading the target
   section before modifying it.
3. Wrap-up: run the self-check list and emit a change summary.
```

Fix mode (dual-anchor diff constraints).

```
You are a skill-evolution agent. The previous evolution result failed the
consistency inspection. Repair the skill using the feedback plus two diffs.

===== baseline -> current: lost content (deleted lines only, for
      precise restoration) =====
{lost_diff}

===== previous round -> current: changes (legitimate edits; reverting all
      of them is forbidden) =====
{last_diff}

===== inspection feedback, MUST-RESTORE class (knowledge loss) =====
```

```
{loss_items}

===== inspection feedback, MUST-CONDENSE class (structural bloat) =====
{bloat_items}

Conflict resolution: case-specific detail -> condensing takes priority;
general stable knowledge -> restoration takes priority.
```

Refine mode (driven by the evaluation report).

```
You are a skill-evolution agent. The previous skill version was deployed to
the canary environment and evaluated with simulated users; the evaluation
report exposes unresolved tickets. Induce the common problems from the
report and revise the skill in a targeted manner.

[Report-driven revision rules] (hard constraints)
- Do not patch tickets one by one; induce common problems: aggregate the
  causes of multiple unresolved tickets into 2-4 root causes, then revise
  against those.
- Keep edits small and precise: prefer supplementing concrete facts /
  paths / rules in the references; modify the main structure of SKILL.md
  with caution.
- If avg_score already lies in an acceptable range (>=0.6), make only minor
  adjustments, avoiding over-revision that introduces regression.
- If an unresolved ticket falls outside the coverage declared by the skill,
  do not widen the boundary; flag it as "out of scope" in the summary.
- Factual fields (values / version numbers / links) must be kept verbatim;
  generalizing them is forbidden.

===== previous evaluation report (condensed) =====
{report_text}
```

B.1.7 Quality Checker prompt

The Quality Checker prompt directs an independent checker to compare the candidate against both the initial skill and the previous round, detecting knowledge loss, error introduction, format corruption, and structural bloat.

```
You are a skill quality checker. Two diffs are supplied for two classes of
check; the baselines are strictly distinct.

1) Knowledge loss (baseline = original version). Against the
[original -> current diff], check whether key information present in the
original is missing from the current version. Pay particular attention to
over-generalization: turning a concrete value in the original (price,
timeline, version number, capacity limit, official link, API name, error
code) into a vague statement such as "refer to the official docs", "depends
on the plan", "a recent version", or "per current product capability"
counts as severe knowledge loss; you MUST name the specific value lost in
reasons and set passed=false.
Criterion: does the information vary with the user or the ticket? Invariant
-> officially stable knowledge -> loss is a defect. Variant ->
case-specific detail -> deletion is legitimate.
Scope of case-specific detail: a particular account / ticket number / name /
order number / temporary link, a concrete operation path or folder name, a
multi-step ticket workflow, a troubleshooting conclusion that depends on a
specific operation history, and a multi-turn clarification flow. Deleting
case-specific detail is not knowledge loss.

2) Error introduction (baseline = previous round). Against the
```

```
[previous -> current diff], check whether this round introduced new factual
errors or content unrelated to the signal.

3) Format corruption (baseline = previous round). Incomplete YAML
frontmatter, disordered heading levels, broken Markdown structure, UTF-8
corruption, and similar problems newly introduced this round.

4) Structural bloat (baseline = previous round). Applies only to the
SKILL.md routing entry and is a minor problem. Bloat of
frontmatter.description is decided by a word count in code (the verdict is
injected into the prompt): above the 400-word threshold -> bloat; below ->
not flagged. Listing verbatim user questions in the topic routing table is
bloat; listing sub-topic keywords is legitimate routing and is not bloat.
Stuffing concrete operation steps / file paths / commands / compensation
amounts into the global constraints is bloat. Supplementing concrete
values / paths / links inside the references is legitimate knowledge-base
behavior and is not bloat. Structural bloat is recorded in reasons for the
editor to repair later, but does not set passed=false.

5) Governance exemption (baseline = original version). If, following a
governance recommendation, the editor performed section merging, tail
consolidation, or file splitting this round, causing large content
migration or file addition/removal:
   - Content migration (a section moved from A to B with nothing lost) is
     not knowledge loss; note "content migrated to X" in reasons.
   - File splitting (one file split into several with nothing lost) is not
     knowledge loss.
   - When merging and deduplicating, content deleted but still semantically
     covered by the merged section (even if phrased more concisely) is not
     knowledge loss; only a knowledge point that disappears entirely
     counts.

6) Cross-round deduplication, for severe problems only (class 1 knowledge
loss / class 2 error introduction / class 3 format corruption). If the
prompt supplies a [problems reported by the previous checker] list and the
current diff still repeats those severe problems, you MUST list them in
reasons and set passed=false; if a problem has been resolved (restored,
corrected, or removed), reporting it again is forbidden. Governance
recommendations do not participate in cross-round deduplication---they are
recommendations, not problems.

7) passed criterion: passed=true if and only if no severe problem (class
1 / 2 / 3) exists. Class 4 structural bloat is minor: recorded but does not
set passed=false. Minor knowledge loss is exempt: a single file losing only
a few isolated knowledge points (<=2-3, not a whole section, not a core
rule / value / link) is recorded but does not set passed=false. Key
judgment: if what was lost is a whole section, a core rule, a concrete
value, an official link, an API name, or an error code, the problem is not
minor.

8) score (float in [0,1]):
   no severe and no minor problems -> score >= 0.9;
   no severe but minor problems -> score 0.6-0.8;
   any severe problem -> score <= 0.5.

9) reasons: record only "problems found", never a description of the
checking process. No problems -> return an empty array. Governance
recommendations are not written into reasons.
   Return JSON only; emit nothing else.
```

B.1.8 Skill Governor prompt

The Skill Governor prompt directs an independent inspector to detect structural degradation in a single reference file and to emit actionable recommendations (section merging, tail consolidation, file splitting).

```
You are a skill governance inspector. A skill is a RAG knowledge base: once
a customer question is routed to the skill, the model loads SKILL.md plus
the references and answers directly. The references form a topic-level
knowledge base, one topic per file. Your task is to inspect a single
reference file and emit actionable governance recommendations (section
merging / tail consolidation / file splitting).

Recommendation types:
1. merge_sections: section topics are orthogonal or duplicated and can be
   merged into one. Write the two section headings in target and the
   rationale in reason.
2. consolidate_tail: scattered tail entries (>=3) such as "additional
   notes / miscellaneous / remarks" can be folded into an existing section.
   Write the tail heading in target and which entries fold where in reason.
3. split_file: emit only when the prompt explicitly states "this file has N
   lines, above the 700-line threshold". Write how many files to split into
   in target and what topic each covers in reason.

Priority (every recommendation must be labeled):
- high: large-paragraph duplication or conflict, or the file has reached the
  splitting threshold (>=700 lines); the gain from merging or splitting is
  evident.
- medium: moderate duplication (section topics overlap but paragraphs are
  not long); merging yields some gain.
- low: scattered minor cleanups (folding short entries); the gain from
  merging is limited.

Constraints:
- At most 3 recommendations per file; prioritize evident, large-paragraph
  duplication or conflict, and avoid scattering the editor's attention over
  minor cleanups.
- Do not deduplicate across files (a skill uses progressive disclosure, and
  cross-file references accumulate model context).
- Emit only valuable recommendations; if there are none, return an empty
  suggestions array.
- Never emit a meaningless merge or split merely to fill the quota.
- Every recommendation must be labeled with priority (high/medium/low).
- Return JSON only; emit nothing else.
```

B.2 Scoring rubrics

B.2.1 Verifier prompt and scoring rubric

The consultation evaluation prompt concentrates on the consistency of knowledge content, scored continuously in $[0, 100]$: 90–100, knowledge accurate and complete; 70–89, largely accurate with minor omissions; 40–69, right direction but missing key knowledge; 10–39, direction deviates; 0–9, entirely irrelevant. Knowledge errors, such as wrong rules or opposite conclusions, are capped strictly at 59, and a case that misses a key condition of the task definition is treated as a failure even when its continuous score reaches 60. Non-knowledge differences (phrasing style, conciseness, supplementary information) are not primary grounds for deduction.

```
You are a professional reviewer of customer-support solutions. Compare the
solution given by the new bot against the one given by the human agent and
assess the consistency of their knowledge content. This evaluation serves
skill self-evolution; the goal is to identify the bot's knowledge gaps.
```

```
## Scoring focus: knowledge content
Judge only whether the learnable factual content the bot gives is accurate
and complete:
- Rules (refund rules, credit rules, validity, eligibility, billing)
- Paths/entries (operation pages, ticket entry, purchase, cancellation)
- Constraints (deadlines, quotas, version and account requirements)
- Product facts (feature boundaries, version differences, account systems)

Non-knowledge differences are not primary grounds for deduction:
- Phrasing style, conciseness -> no deduction
- Whether extra supplementary information is given -> no deduction
- Dialogue strategy -> no deduction

## Rubric (0-100)
- 90-100: accurate and complete; core rules/paths/conditions match
- 70-89: largely accurate, minor omissions
- 40-69: right direction but key knowledge missing or incomplete
- 10-39: direction deviates; only a small part matches
- 0-9: knowledge content entirely irrelevant or wrong

## Knowledge errors must be penalized strictly (cap at 59)
1. Wrong rule: the bot's rule contradicts the human's, or is
   self-contradictory.
2. Opposite conclusion: bot and human give opposite factual verdicts on
   the same question.

## Output format
Return JSON: {"score": 80, "ai_solution_summary": "...",
"human_solution_summary": "...", "reasoning": "rationale"}
```

B.2.2 Quality score rubric

The quality-score rubric rates a ticket's value as an evaluation sample on an integer scale of 1–10:

- **9–10**: the human gave a complete solution plus several factual knowledge points, the turn count is moderate, and the problem is clear; an excellent evaluation sample.
- **7–8**: the human gave partial knowledge or a partial solution and the dialogue is reasonably clear; a usable evaluation sample. A lightweight-troubleshooting ticket may also score 7 or above when its steps are clear and reusable.
- **5–6**: the human gave some information but incompletely, or the dialogue is too short or too long to judge; evaluation value is moderate.
- **3–4**: the human gave only procedural replies (escalation, requests to elaborate, ticket filing) with no substantive business knowledge; unsuitable for evaluation.
- **1–2**: no valid human reply, an extremely short dialogue, or pure complaint with no business request; unsuitable for evaluation.

Key principle. The quality score is independent of generalizability: `generalizable` judges whether the knowledge can be generalized, whereas `quality_score` rates whether the ticket is worth using as an evaluation sample. The governing criterion is whether the ticket can clearly reveal if the skill resolved the user's problem.

C Algorithm

```
Input: initial skill S0; development tickets T_dev;
       held-out evaluation tickets T_eval; max rounds R
C <- {S0}; S <- S0
for r = 1 ... R:
```

```
    traces   <- Simulate(Scenarios(T_dev), ServiceAgent(S))
    feedback <- Verify(traces, references(T_dev))
    signals  <- Merge({f in feedback |
                       f.result = failure and
                       f.root_cause = knowledge_gap})
    if signals is empty: break   # no repairable signal; terminate early

    S_candidate <- BoundedEdit(S, signals, S0)  # bounded revision (S, S0)
    S_candidate <- Govern(S_candidate, S0, S)   # fact consistency (hard)
                                                # + structural governance
    results     <- Verify(Simulate(Scenarios(T_dev),
                                   ServiceAgent(S_candidate)))
    SaveCheckpoint(S_candidate, results)
    C <- C union {S_candidate}
    S <- S_candidate    # no scalar TSR gate once fact consistency holds
    if EarlyStop(results): break

S* <- argmax_{s in C} (TSR_dev(s), AvgScore_dev(s))   # selection on dev set
Freeze(C, S*)
final_results <- Verify(Simulate(Scenarios(T_eval), ServiceAgent(S*)))
return S*, final_results   # the evaluation set is used only for reporting
```

D Case study

We trace ticket 17413068 of `object-storage-consultation` (object storage service) through reconstruction, verification, attribution, and bounded revision. The agent did not merely omit knowledge: it stated a rule *exactly opposite* to the correct one, and no subsequent turn corrected it. We note that this initial error is itself visible in the first exchange; what the multi-turn dialogue adds is evidence that the conversation proceeds on the false premise without self-correction.

D.1 Scenario reconstruction

The scenario synthesizer extracts the intent agenda, behavior facts, and human reference solution from the real ticket and assembles a constrained evaluation task.

```
Opening message: When a traffic package is renewed, does it take effect
                 immediately or only after the current one expires?
Behavior facts:  Holds one outbound-traffic package, 200 GB for one
                 month, expiring July 27 at 23:59; 182 GB already used,
                 18 GB left.
Emotion:         Worried that the remaining traffic is insufficient;
                 repeatedly seeks confirmation of the rule.
Intent agenda:
  [key]   Does a renewed traffic package take effect immediately or only
          after the current one expires?
  [minor] Will service be suspended between exhausting the old package and
          the new one taking effect?
  [minor] Is the account balance deducted first once the old package is
          exhausted?
Human reference: Renewal extends the validity period. The current 200 GB
                 package expires July 27 at 23:59; renewing for one month
                 extends validity to August 27 at 23:59, with the reset
                 date at July 28 00:00, when a fresh 200 GB quota is
                 granted. Renewal extends duration rather than refreshing
                 quota. Once the old package is exhausted, usage switches
                 to pay-as-you-go without service suspension.
```

D.2 Failed interaction trajectory

Loaded with the initial skill, the agent states the opposite of the correct rule in its very first turn. The simulated user then asks about the risk of suspension; the agent answers that question correctly but never corrects its initial error.

```
User:  When a traffic package is renewed, does it take effect immediately
       or only after the current one expires?
Agent: Renewal takes effect immediately. Once you renew, the traffic quota
       is added to your current package right away; you need not wait for
       the original package to expire.
User:  After the remaining 18 GB of the old package runs out, will service
       be suspended if I have not renewed?
Agent: No. Once a traffic package is exhausted, the excess automatically
       switches to pay-as-you-go and is charged against your account
       balance; the service remains available.
User:  Pay-as-you-go is deducted from the balance, right? And what if the
       balance is insufficient too?
Agent: If the balance is insufficient, the account enters arrears; one day
       after arrears begin, the service is suspended and data is retained
       for 15 days.
User:  Understood, thanks.
```

The intent state machine records the key intent as expressed but *answered incorrectly* (the agent claims immediate accrual whereas the human reference specifies extension of validity), so $c_U = 1.0$ and the sample clears the simulator-side coverage gate before proceeding to agent-side evaluation and attribution. Notably, the user does not abandon the dialogue after the first turn, because the agent supplied a *plausibly self-consistent but wrong* answer; the user then builds on that false premise to ask about suspension. The dialogue continues on an incorrect premise without self-correction, a persistence that only multi-turn interaction reveals: incorrect knowledge is more deceptive than absent knowledge.

D.3 Verifier verdict

The Verifier compares the simulated dialogue with the human reference and records a severe failure.


```
{"score": 10,
 "ai_solution_summary": "claims renewal takes effect immediately and quota
   is added to the current package at once",
 "human_solution_summary": "renewal extends validity; the fresh quota
   arrives on the reset date (00:00 of the day after expiry)",
 "reasoning": "the renewal rule given by the agent is the exact opposite of
   the human agent's: the agent claims the quota accrues immediately upon
   renewal, whereas the human states explicitly that renewal extends
   validity and the new quota is usable only from the reset date. This is
   a core rule error that directly misleads the user about when a renewed
   package takes effect and may cause the user to misjudge traffic
   availability."}
```


D.4 Attribution output

Comparing the human handling process, the simulated dialogue, and the Verifier's evidence, the Attributor classifies the failure as a Knowledge Gap: the root cause is the absence of the renewal-extension rule in the skill, which skill revision can supply.


```
{"root_cause": "knowledge_gap",
 "knowledge_facts": [
   "The core renewal rule is extension of validity, with a fresh quota for
    the new cycle; the reset date is 00:00 of the day after expiry",
   "Renewal does not refresh traffic directly: after renewal the remaining
    traffic of the original package is still consumed on the original
```

```
    cycle, and the new quota becomes usable only from the reset date",
   "When the original package is exhausted before the new one takes effect,
    usage switches to pay-as-you-go rather than being suspended"],
 "suggested_change": "add the renewal-extension rule to
   resource-pack-deduction.md and correct the erroneous statement that
   renewal takes effect immediately with quota added at once",
 "target_file": "references/resource-pack-deduction.md",
 "evidence": [
   "human: the core renewal rule is extension of validity, with a fresh
    quota for the new cycle",
   "human: renewal renews the duration of the package; it does not refresh
    the quota directly",
   "bot: renewal takes effect immediately; the traffic quota is added to
    your current package right away"],
 "needs_human_review": false}
```

A further failed ticket in the same round points to the same renewal-rule gap, and collective attribution merges the two into a single learning signal.

D.5 Bounded revision

On receiving the signal, the Skill Editor makes a minimal edit to the package-validity section of `references/resource-pack-deduction.md`, appending three renewal rules without deleting any existing content.

```
 - Newly purchased outbound-traffic packages accrue additively.
 - Multiple newly purchased packages each count validity from their own
   effective time, independently.
 - Renewed outbound-traffic packages do not accrue additively.
 - Renewal extends validity only and does not add traffic quota; to add
   quota, purchase a new package.
+- Renewal renews the duration of the package rather than refreshing the
+  quota directly: after renewal, the new cycle grants a fresh quota on the
+  reset date (00:00 of the day after expiry). If the original package is
+  exhausted before the reset date, the new package's traffic is unavailable
+  and the excess is charged pay-as-you-go against the account balance.
 - The renewal price equals the original purchase price; the console price
   is authoritative.
```

The revision is anchored to the initial skill S_0; the additions are confined to the three facts covered by the signal and leave the remaining thirteen sections of the file untouched.

D.6 Interaction after the update

Re-simulating the same scenario with the revised Skill, the agent states the correct rule in its first turn and addresses the user's central concern directly.

```
User:  When a traffic package is renewed, does it take effect immediately
       or only after the current one expires?
Agent: Renewal extends the validity period rather than refreshing the quota
       directly. Your current 200 GB package expires July 27 at 23:59;
       renewing for one month extends validity to August 27 at 23:59, and
       the reset date is July 28 at 00:00, when a fresh 200 GB quota is
       granted. If the old package runs out before the reset date, usage
       switches to pay-as-you-go and the service is not suspended.
User:  So if the 18 GB runs out before 24:00 on July 27, can I use the new
       package's traffic?
Agent: No. Before the reset date the new package's traffic is unavailable,
       and the excess is charged pay-as-you-go against your balance.
User:  Understood---I just need to keep the balance topped up. Thanks.
```

The intent state machine records the key intent and both minor intents as expressed and correctly addressed (c_U = 1.0), and the Verifier scores the dialogue 92: knowledge accurate, complete, and correct in rule.

E Implementation details

Model assignment. The Skill Editor performs bounded edits with `deepseek-v4-pro`, while the Verifier, User Agent, Attributor, and Skill Governor run verification, simulation, attribution, and governance detection on `minimax-m3`. Editing and evaluation therefore use different model families, which precludes the circular dependency of a model reviewing its own edits.

Evaluation environment. The service agent runs in-process headless under a tool allow-list exposing only skill loading and read-only retrieval; write tools are deregistered so that the agent cannot modify code or skills of its own accord.

F Hyperparameters and configuration

Table 9: Model assignment and evolution parameter configuration.

Component	Parameter	Value
Skill Editor	Model family	A (`deepseek-v4-pro`)
Verifier / Attributor / User Agent / Skill Governor	Model family	B (`minimax-m3`)
Evolution loop	max_cycles (full loop rounds)	4
Evolution loop	max_iterations (edit iterations per round)	3
Evolution loop	max_turns (interaction turns per ticket)	10
Evolution loop	early_stop_avg_score	70.0
Evolution loop	early_stop_solved_ratio	0.7
Evolution loop	pass_threshold	60.0
Evaluation	Intent weight α	0.7
Signal merging	keyword_jaccard_threshold	0.3

The editing and evaluation components belong to different model families. Threshold parameters (α, `pass_threshold`, and the early-stopping criteria) follow the settings already in use in the deployed evaluation pipeline; we did not perform a sensitivity sweep.

Simulation and evaluation protocol. The simulator is capped at ten interaction turns per conversation, with an average of five turns actually used, and produces ten conversations per query. Scenarios are regenerated at every round, so the agent faces freshly sampled tickets rather than a fixed set replayed across rounds; each ticket is simulated once. The evaluation set is re-run in full at every round, and the reported TSR at round r is measured on that round's full pass. The exposed-intent accuracy s_C is averaged over all rounds rather than read off a single round.

G System architecture and design principles

G.1 End-to-end pipeline

```
Ticket labeling (domain classification, daily)
  -> Escalated-ticket retrieval + quality assessment + de-identification
     (indexed by time bucket on the data side, T+1 day lag)
  -> Signal extraction (ticket_to_signal)
  -> Iterative editing (evolve fix -> governance fix -> consolidation fix)
  -> Two-level loop (inner: evaluate -> attribute -> edit -> check;
                     outer: cross-round select -> publish)
  -> Full evaluation each round
  -> Cross-round selection
  -> Human confirmation before production rollout
```

G.2 Two-level loop

The evolution loop is organized as two nested levels.

Inner loop (edit iterations within a round): edit → check → fix, up to three iterations. Each iteration first runs the governance check over the reference structure, then edits (*fix*, or *evolve* in the first round), then inspects; a failed inspection routes back to *fix*. On success the iteration emits the evolved skill together with a change summary.

Outer loop (multi-round evolution): evaluate → attribute → edit → evaluate, up to four rounds. Each round begins with the first-round edit (carrying the *evolve* summary), then runs a full evaluation to produce the report, and subsequently revises via *refine* on the multi-turn failure signals until early stopping. The governing principle is that every checkpoint must pass either an inspection or a full evaluation before being persisted.

G.3 Key design principles

1. **Feedback closure.** Real ticket → simulated user → agent dialogue → attribution → signal → edit. No hand-crafted scenarios are needed; failures are converted into learning signals automatically.
2. **Bounded editing.** Only gaps that a human agent can handle and the bot cannot are patched. The editor may inspect real trajectories but may not depart from the evidence, so evolution remains evidence-driven rather than a free exercise of model priors.
3. **Two-tier governance.** Every round of editing is followed by a mandatory inspection against degradation and bloat, and the governor must be a different model from the editor, since self-review induces circular dependency.
4. **Auditability.** The intent, evidence, and outcome of every round of editing are persisted, keeping the whole trajectory traceable.