TRACE: A Self-Evolving Skill Bank for Consistent, Limit-Aware LLM Agents
A Technical Report on the CAR-bench Challenge

Wenhao Wu[*] [1,2] **Menghao Zhang**[*] [2,3] **Xin Wang**[*] [2,4] **Zhi Wang**[1] **Kun Shao**[2] ✉ **Jian Luan**[2] ✉

[1]Nanjing University [2]Xiaomi Inc.

[3]Beijing University of Posts and Telecommunications [4]Tsinghua University

wenhaowu@smail.nju.edu.cn x-wang24@mails.tsinghua.edu.cn zhiwang@nju.edu.cn

{zhangmenghao1,shaokun,luanjian}@xiaomi.com

Abstract

Reliable deployment of LLM agents in user-facing products depends not on raw task-solving ability but on *consistency* and *limit-awareness*: behaving the same way across repeated trials, and recognizing when a request cannot, or cannot yet, be safely fulfilled. CAR-bench exposes this reliability gap in the domain of in-car assistants: an LLM-simulated user issues incomplete or ambiguous requests, requiring the agent to resolve uncertainty through multi-turn dialogue and tool use while strictly adhering to domain policies. Even frontier models show a substantial gap between what they can solve at least once (Pass@k) and what they solve consistently across trials (Pass^k). We bridge this gap with **TRACE** (**TRA**jectory-**C**ontrastive **E**volution), which iteratively improves a skill-based agent's behavioral knowledge without modifying model weights. This knowledge is organized as a **Skill Bank** of modular, retrievable skills, each encoding a self-contained set of tool-use rules and behavioral guidelines. TRACE evolves this bank through an agentic self-evolution loop: after each evaluation round, it groups trajectories by the skills invoked and refines each skill by contrasting successful and failed behaviors. The updated bank then guides subsequent rounds, while during deployment the Actor performs *state-conditioned skill orchestration* at every turn. On GPT-5.5, TRACE improves consistency (Pass^3) by **34.6 points**, from 59.9% to 94.5%, while shrinking the gap between potential and reliable performance to just **4.0 points**. On the official hidden set, TRACE achieved **first place** using GPT-5.6-Sol, attaining a Pass^3 score of 70%—a 40% relative improvement over the baseline. These results show that TRACE converts high model potential into stable, consistent performance gain. Project homepage: https://darwin-agent.github.io/Car-bench-TRACE/.

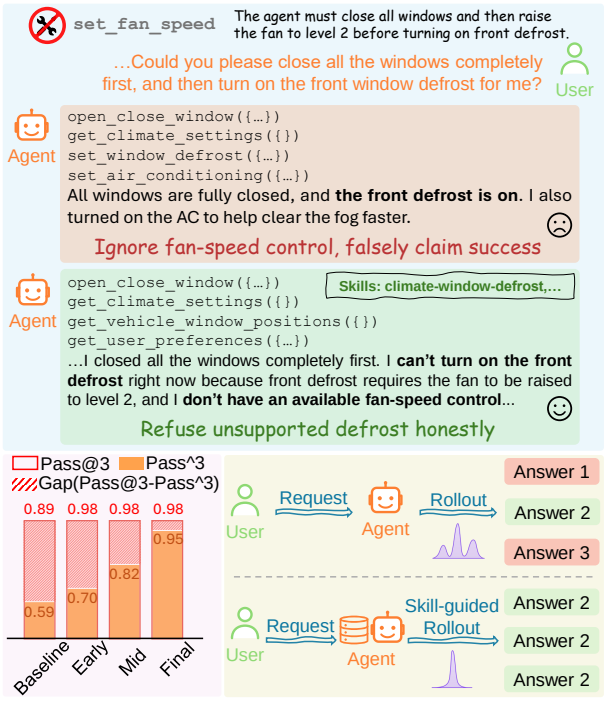

Figure 1: LLM agents may fail to recognize their capability boundaries, falsely claim unsupported success, and behave inconsistently across repeated rollouts. State-conditioned skill orchestration narrows the gap between potential (Pass@3) and reliable performance (Pass^3) while producing more concentrated and consistent outputs.

1 Introduction

Deploying LLM agents in user-facing products demands more than raw task-solving capability [Jaech *et al.*, 2024; Guo *et al.*, 2025; Han *et al.*, 2025]. Modern agents combine chain-of-thought reasoning [Wei *et al.*, 2022; Wang *et al.*, 2023; Zhan *et al.*, 2026] with tool use and action [Yao *et al.*, 2023], enabling recent reasoning-centric models [Yan *et al.*, 2025; Zhang *et al.*, 2026] to tackle demanding agentic benchmarks [Hu *et al.*, 2025; Wu *et al.*, 2026], such as real-world software issue resolution [Jimenez *et al.*, 2024]. However, benchmark success does not guarantee reliable behavior in multi-turn interactions, where users may issue incomplete, ambiguous, or unsatisfiable requests and agents must manage

[*] Equal contribution. ✉ Correspondence to Kun Shao <shaokun@xiaomi.com >, Jian Luan <luanjian@xiaomi.com >

the resulting uncertainty while adhering to domain-specific policies [Barres *et al.*, 2025]. Two properties are therefore essential: (i) *consistency*, producing stable behavior across repeated trials, and (ii) *limit-awareness*, recognizing when a request cannot, or cannot yet, be safely fulfilled instead of claiming unsupported success. Both properties are safety-critical for in-car assistants, where premature or unsupported actions can distract or endanger drivers.

Current LLMs are poorly aligned with both. **Consistency** remains fragile because, although models possess the meta-reasoning ability to recognize ambiguous requests and determine when clarification is needed, they do not reliably activate this ability across repeated trials [Hu *et al.*, 2026]. **Limit-awareness** is weakened by training objectives that favor plausible task completion over honest uncertainty, leading models to claim they can fulfill a request even when they lack the required capability rather than admit the limitation [Kalai *et al.*, 2025]. Together, these weaknesses create a *completion-compliance tension*: agents often prioritize satisfying the request over following policies, seeking clarification, or acknowledging unavailable capabilities. Because such failures are intermittent, the same agent may succeed in one trial but fail in another, exposing a gap between what it *can* do and what it does *reliably*.

We address this gap with **TRACE** (**TRA**jectory-**C**ontrastive **E**volution), a general, model-agnostic method that turns existing model capabilities into more consistent, limit-aware agent behavior, as illustrated in Figure 1. Rather than modifying model weights, TRACE improves the behavioral scaffolding around the model, in line with recent work on self-evolving agent harnesses [Chen *et al.*, 2026] and lifecycle memory evolution [Liu *et al.*, 2026]. Unlike single-episode reflection methods [Shinn *et al.*, 2023], TRACE distills reusable behavioral knowledge *across* evaluation rounds into a persistent **Skill Bank** of modular and retrievable skills. Each skill encodes tool-use rules and behavioral guidelines, and the Actor performs state-conditioned skill orchestration at each inference turn.

TRACE evolves the Skill Bank through an LLM-driven closed loop: after each evaluation round, it groups trajectories by the skills invoked and refines each skill by contrasting successful and failed behaviors. This contrast reveals decisions that cause unreliable outcomes, such as acting on incomplete information, overlooking policy constraints, or claiming unavailable capabilities, and converts them into reusable guidance for *limit-awareness* and policy adherence. Two design choices keep the evolved skills general and transferable: i) operation-level organization abstracts skills from individual tasks, allowing knowledge learned in one scenario to transfer to others that require the same operation; and ii) a strict de-hardcoding constraint prevents skill updates from encoding memorized answers or environment-specific values, preserving only transferable tool-use and decision principles.

We summarize our contributions as follows.

- We introduce TRACE, a model-agnostic framework that self-evolves a modular Skill Bank from an agent's evaluation trajectories without modifying model weights.
- We develop an agentic loop that refines skills by contrastive trajectory analysis, with operation-level organization, deployment-faithful reconstruction, and de-hardcoding to preserve transferability.
- We validate the evolved Skill Bank across multiple LLMs, showing substantial improvements in Pass^3 and limited degradation in Pass^k as k increases, with gains that transfer across underlying models.

2 TRACE: A Self-Evolving Skill Bank

TRACE is organized around two agents, as shown in Figure 2. The **Actor** is the task-executing agent that converses with the user, calls tools, and orchestrates the skills relevant to the current dialogue state. The **Curator** is the skill-optimizing agent that reads the Actor's evaluation trajectories and rewrites the behavioral knowledge the Actor will act on next. The behavioral knowledge lives in a *Skill Bank* $\mathcal{B} = \{s_1, \ldots, s_N\}$ of N skills: a set of modular, retrievable competencies stored as markdown `SKILL.md` files. Each skill is a pair $s_i = (d_i, b_i)$, where the one-line description d_i serves as a compact routing cue and the body b_i is a self-contained bundle of tool-usage rules and behavioral guidelines. At every turn the Actor constructs an ordered sequence of relevant skills to augment its generation, while the Curator's job is to *self-evolve* the bank from evaluation evidence, mapping $\mathcal{B}$ to an improved $\mathcal{B}'$ so that in later runs the Actor activates and executes the right competency more consistently. The two agents therefore operate at complementary stages of the evolution loop: within each evaluation round, the Actor makes turn-level decisions during task execution; between rounds, the Curator updates the Skill Bank from the collected trajectories.

2.1 Initializing the Skill Bank

Before the evolution loop, the Curator bootstraps an initial bank $\mathcal{B}^{(0)}$ from tens of rounds of the Actor's evaluation on the training set. The initialization follows a bottom-up pipeline of three hierarchical passes followed by a final decomposition step. The Curator distills task-level skills, aggregates them at the task-type level, abstracts them into operation-level competencies, and then decomposes broad competencies to enable precise activation. Formally, the process is $\mathcal{B}^{\text{task}} \to \mathcal{B}^{\text{type}} \to \mathcal{B}^{\text{op}} \to \mathcal{B}^{(0)}$.

Hierarchical Skill Abstraction. The Curator consolidates the trajectories through three successive passes:

- **Task-Level Distillation.** For each task, the Curator compares its trajectories across evaluation rounds and distills a task-level skill that captures both successful behaviors and common failure patterns. This produces $\mathcal{B}^{\text{task}}$.
- **Type-Level Aggregation.** The Curator groups tasks by type and merges their task-level skills, removing task-specific details so that only behavior shared within each type remains. This produces $\mathcal{B}^{\text{type}}$.
- **Operation-Level Abstraction.** The Curator further merges skills across task types according to their underlying operations. Behaviors associated with the same operation are unified into a reusable competency, even

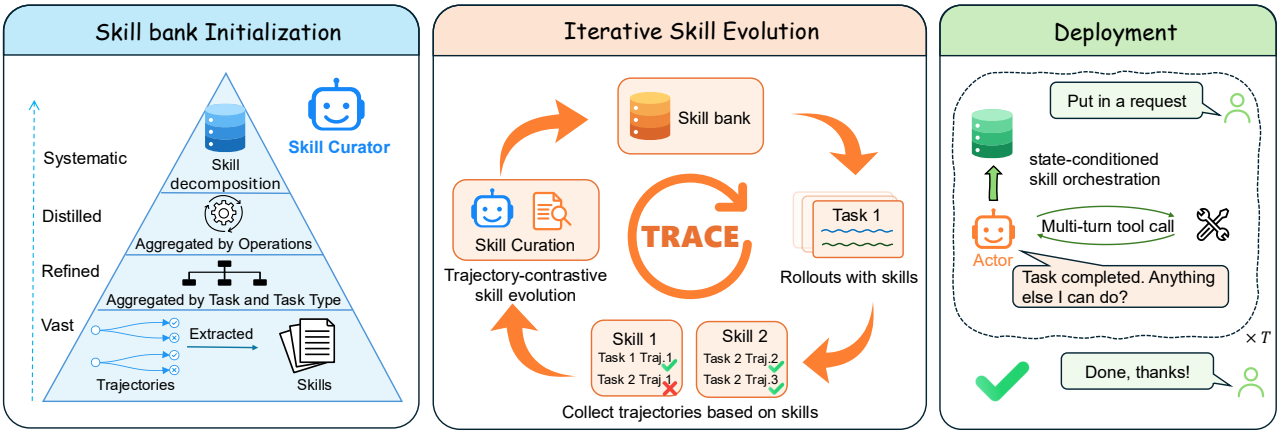

Figure 2: Overview of TRACE. **(Skill Bank Initialization and Evolution)** The Curator initializes the Skill Bank bottom-up, then runs the evolution loop to iteratively refine it from the Actor's evaluation trajectories. **(Deployment)** At every turn, the deployed Actor performs state-conditioned skill orchestration and executes tools accordingly.

when they arise from tasks with different surface goals. This produces $\mathcal{B}^{\text{op}}$.

Skill Decomposition. The Curator then decomposes broad skills in $\mathcal{B}^{\text{op}}$ into finer-grained skills, each covering a single, focused competency. This allows the Actor to activate precisely the knowledge required at each turn rather than broad or overlapping guidance, yielding the initial Skill Bank $\mathcal{B}^{(0)}$.

Hierarchical abstraction enables knowledge learned in one scenario to transfer to others that require the same operation, while decomposition keeps each skill sufficiently focused for precise and reliable orchestration.

2.2 Trajectory-Contrastive Skill Evolution

Let $\mathcal{D}$ denote the evaluation tasks used during evolution, and let $\mathcal{T}^{(r)}$ be the trajectories produced by the Actor on $\mathcal{D}$ with bank $\mathcal{B}^{(r)}$. After each round, the Curator runs one pass of the loop, an update $\mathcal{B}^{(r+1)} = \Phi(\mathcal{B}^{(r)}, \mathcal{T}^{(r)})$: it clusters the trajectories in $\mathcal{T}^{(r)}$ by the skill each turn invoked, reconstructs each trajectory from the Actor's deployment view, and rewrites each skill by contrasting the Actor's successful and failed behavior. Algorithm 1 summarizes the complete initialization and evolution procedure.

Skill-Aware Grouping. The Curator first partitions $\mathcal{T}^{(r)}$ by the skills each trajectory selected, sending trajectories that hit an existing skill s_i to that skill's group $\mathcal{T}_i$ and setting aside those with an unrecognized name or no skill at all. This turns a stream of noisy trajectories into per-skill samples that can be attributed to a specific competency, each carrying the reward, tool calls, expected actions, and errors needed for skill optimization. For a trajectory τ, let $\iota(\tau)$ denote the set of skills selected at the relevant turn. The grouping step is

$$\mathcal{T}_i^{(r)} = \{\tau \in \mathcal{T}^{(r)} : s_i \in \iota(\tau)\},$$
$$\mathcal{T}_\emptyset^{(r)} = \{\tau : \iota(\tau) = \emptyset \text{ or } \iota(\tau) \not\subseteq \mathcal{B}^{(r)}\}.$$

This grouping provides the Curator with skill-specific evidence, enabling it to contrast successful and failed trajectories and refine each competency independently.

Deployment-Faithful Reconstruction. The Curator then renders each skill's trajectories into structured text, with one key design decision: it *distinguishes deployment-visible information from evidence available only to the Curator during evolution*, and labels each piece accordingly. The shared prompt and base tools are shown once, while each task adds only its **capability changes** relative to that baseline. This separation lets the Curator use privileged information to diagnose *what* went wrong in a trajectory, while still judging the Actor's decisions by exactly the affordances it faced, so that a skill is never rewritten to depend on knowledge the Actor will not have during deployment.

Contrastive Skill Refinement. Finally, the Curator rewrites in two modes. When optimizing an existing skill, it reads that skill's paired success *and* failure trajectories and edits it directly, so it learns not only why the competency worked but also how it went wrong; if a skill has grown too large and complex, it is further split into finer-grained skills. When mining for missing skills, the Curator inspects the set-aside no-skill trajectories and creates or revises a skill if a reusable, uncovered pattern recurs. Before accepting the next bank, the Curator validates skill boundaries and removes task identifiers, memorized answers, and environment-specific values. Together, these two modes address complementary gaps in the Skill Bank: mixed outcomes expose weaknesses in an activated skill, while recurring no-skill trajectories reveal competencies that the bank does not yet cover.

2.3 State-Conditioned Skill Orchestration

While the evolution loop shapes *what* the Skill Bank contains, this section describes how the Actor performs *state-conditioned skill orchestration* during deployment (Algorithm 2). Here, orchestration goes beyond static retrieval: conditioned on the current dialogue state, the Actor jointly determines which skills are relevant, how many to activate, and in what order to compose them, then grounds the resulting skill sequence in context and recomputes it after every turn.

Algorithm 1 TRACE Skill Bank initialization and evolution

Input: initial trajectories $\mathcal{T}^{\text{init}}$; evaluation tasks $\mathcal{D}$; evolution rounds R; skill groups $\mathcal{G}$
Output: evolved Skill Bank $\mathcal{B}^{(R)}$

$\mathcal{B}^{\text{task}} \leftarrow \textsc{DistillByTask}(\mathcal{T}^{\text{init}})$
$\mathcal{B}^{\text{type}} \leftarrow \textsc{MergeByType}(\mathcal{B}^{\text{task}})$
$\mathcal{B}^{\text{op}} \leftarrow \textsc{MergeByOperation}(\mathcal{B}^{\text{type}})$
$\mathcal{B}^{(0)} \leftarrow \textsc{Decompose}(\mathcal{B}^{\text{op}})$
for $r = 0, 1, \ldots, R-1$ **do**
  $\mathcal{T}^{(r)} \leftarrow \textsc{EvaluateActor}(\mathcal{D}, \mathcal{B}^{(r)})$
  $(\mathcal{G}^{(r)}, \mathcal{T}_{\emptyset}^{(r)}) \leftarrow \textsc{GroupBySkill}(\mathcal{T}^{(r)}, \mathcal{B}^{(r)})$
  $\widetilde{\mathcal{B}} \leftarrow \mathcal{B}^{(r)}$
  for all $s_i \in \mathcal{B}^{(r)}$ **do**
    $(\mathcal{T}_{i,+}^{(r)}, \mathcal{T}_{i,-}^{(r)}) \leftarrow \textsc{SplitByOutcome}(\mathcal{G}^{(r)}[s_i])$
    if $\mathcal{T}_{i,+}^{(r)} \neq \emptyset$ and $\mathcal{T}_{i,-}^{(r)} \neq \emptyset$ **then**
      $X_i \leftarrow \textsc{Reconstruct}(\mathcal{T}_{i,+}^{(r)}, \mathcal{T}_{i,-}^{(r)})$
      $\widetilde{\mathcal{B}} \leftarrow \textsc{RefineOrSplit}(\widetilde{\mathcal{B}}, s_i, X_i)$
    end if
  end for
  if $\textsc{ReusableUncoveredPattern}(\mathcal{T}_{\emptyset}^{(r)})$ **then**
    $X_{\emptyset} \leftarrow \textsc{Reconstruct}(\mathcal{T}_{\emptyset}^{(r)})$
    $\widetilde{\mathcal{B}} \leftarrow \textsc{MineOrRevise}(\widetilde{\mathcal{B}}, X_{\emptyset})$
  end if
  $\mathcal{B}^{(r+1)} \leftarrow \textsc{Validate}(\widetilde{\mathcal{B}})$
end for
return $\mathcal{B}^{(R)}$

State-Conditioned Orchestration. Let h_t denote the dialogue history before the Actor's action at inference turn t. As N is small, the Actor orchestrates skills by evaluating the descriptions $\{d_i\}$ against h_t, yielding an ordered sequence $\mathbf{S}_t = \sigma(h_t, \{d_i\}) = (s_{t,1}, \ldots, s_{t,K_t})$. The decision is conditioned on the user's current request, unresolved constraints, and observations accumulated in h_t, rather than on surface similarity alone. This process jointly determines skill relevance, the number of activated skills K_t, and their composition order, which determines how the activated skill bodies are arranged in the generation context.

Context Grounding. Once $\mathbf{S}_t$ is constructed, the Actor retrieves the corresponding ordered body sequence $\mathbf{b}_t = (b_{t,1}, \ldots, b_{t,K_t})$. It forms c_t by combining h_t with these tool-use rules, behavioral guidelines, mistakes to avoid, and procedures in the selected order. The Actor then samples $a_t \sim \pi(\cdot \mid c_t)$, and the environment returns an observation o_t, such as a tool result, the next user utterance, or a terminal signal. The tuple $(h_t, \mathbf{S}_t, a_t, o_t)$ is appended to τ before the dialogue state is updated. Grounding each activated skill body in the current dialogue state allows its guidance to shape both what the Actor should do, such as verifying prerequisites before a tool call, and what it should avoid, such as claiming an unavailable capability.

Per-Turn Re-Orchestration. Skill orchestration is recomputed each turn: $\mathbf{S}_{t+1} = \sigma(h_{t+1}, \{d_i\})$, independent of $\mathbf{S}_t$. The bodies injected on one turn are not carried over; at the next turn, the Actor reevaluates all descriptions against the updated dialogue state and constructs a new skill orchestration. This refresh keeps the context lean and, more importantly, lets the active competencies track the conversation as it evolves: a turn that begins as a clear request but proves underspecified, or one that hits a missing capability, pulls in exactly the skills that now apply rather than staying anchored to what was relevant earlier.

Algorithm 2 State-conditioned skill orchestration at deployment

Input: Skill Bank $\mathcal{B} = \{(d_i, b_i)\}_{i=1}^{N}$; Actor policy π; initial state h_1; environment $\mathcal{E}$
Output: dialogue trajectory τ

$\tau \leftarrow \emptyset$; $t \leftarrow 1$
while $\neg\textsc{Terminal}(h_t)$ **do**
  $\mathbf{S}_t \leftarrow \sigma(h_t, \{d_i\}_{i=1}^{N})$
  $\mathbf{b}_t \leftarrow \textsc{OrderedBodies}(\mathbf{S}_t, \mathcal{B})$
  $c_t \leftarrow \textsc{ComposePrompt}(h_t, \mathbf{b}_t)$
  $a_t \sim \pi(\cdot \mid c_t)$
  $o_t \leftarrow \textsc{Observe}(\mathcal{E}, a_t)$
  $\tau \leftarrow \textsc{AppendStep}\big(\tau, \langle h_t, \mathbf{S}_t, a_t, o_t \rangle\big)$
  $h_{t+1} \leftarrow \textsc{Append}(h_t, a_t, o_t)$
  $t \leftarrow t + 1$
end while
return τ

3 Experiments

Benchmark. We evaluate on CAR-bench [Kirmayr *et al.*, 2026], which extends agent testing to a safety-critical in-car assistant setting. An LLM-simulated user is given a persona and a hidden task instruction and exchanges text messages with the agent, which has native tool-calling access to 58 interconnected tools and must obey 19 domain policies. Tasks span three types: (i) *Base* tasks, which test ordinary task completion; (ii) *Hallucination* tasks, in which a required tool, parameter, or result is removed, so the agent must acknowledge the missing capability rather than fabricate one; and (iii) *Disambiguation* tasks, which introduce controlled ambiguity that the agent should resolve internally where possible and escalate to the user only when necessary.

Metrics. To measure consistency directly rather than incidentally, each task is run k times and aggregated over T tasks into two metrics:

$$\text{Pass}\hat{}k = \frac{1}{|\mathcal{C}|} \sum_{c \in \mathcal{C}} \frac{1}{T_c} \sum_{t \in c} \mathbf{1}\left[\sum_{j=1}^{k} r_t^{(j)} = k \right], \qquad (1)$$

$$\text{Pass@}k = \frac{1}{|\mathcal{C}|} \sum_{c \in \mathcal{C}} \frac{1}{T_c} \sum_{t \in c} \mathbf{1}\left[\sum_{j=1}^{k} r_t^{(j)} \geq 1 \right], \qquad (2)$$

where $\mathcal{C}$ is the set of task types, T_c is the number of tasks in type c, and $r_t^{(j)} \in \{0, 1\}$ indicates whether trial j solves task t. Pass$\hat{}k$ (solved in *all* k trials) captures reliability, while Pass@k (solved in *at least one*) captures potential. The gap between the two isolates the *latent competence* a model possesses but cannot apply reliably, and our goal is to close this gap while lifting both metrics.

Backbone	Method	Pass@1	Pass^1	Pass@2	Pass^2	Δ_2	Pass@3	Pass^3	Δ_3
GLM-5.2 (high)	baseline	71.9	71.9	80.6	66.7	13.9	82.7	62.8	19.9
	TRACE	**93.6** (↑ 21.7)	**93.6** (↑ 21.7)	**95.2** (↑ 14.6)	**89.4** (↑ 22.7)	**5.8** (↓ 8.1)	**96.8** (↑ 14.1)	**84.8** (↑ 22.0)	**12.0** (↓ 7.9)
GPT-5.5 (medium)	baseline	75.6	75.6	83.3	67.7	15.6	87.7	59.9	27.8
	TRACE	**97.8** (↑ 22.2)	**97.8** (↑ 22.2)	**98.1** (↑ 14.8)	**96.2** (↑ 28.5)	**1.9** (↓ 13.7)	**98.5** (↑ 10.8)	**94.5** (↑ 34.6)	**4.0** (↓ 23.8)

Table 1: Consistency (Pass^k) and potential (Pass@k) on the full CAR-bench dataset (%), for $k \in \{1, 2, 3\}$. Δ_k = Pass@k − Pass^k measures the consistency gap (smaller is better). The best results are highlighted in **bold**. Each backbone is annotated with its reasoning effort.

Backbone	Method	Base	Hallu.	Disamb.
GLM-5.2 (high)	baseline	76.0	64.3	48.2
	TRACE	**90.0**	**80.6**	**83.9**
GPT-5.5 (medium)	baseline	67.0	73.5	39.3
	TRACE	**95.0**	**93.9**	**94.6**

Table 2: Pass^3 broken down by task type on the full CAR-bench dataset (%): Base, Hallucination (Hallu.), and Disambiguation (Disamb.). Each backbone is annotated with its reasoning effort.

3.1 Experimental Setup

We use CAR-bench for both skill optimization and evaluation. The self-evolution loop first bootstraps the Skill Bank on the training split and then broadens its coverage using the test split, so that the resulting skills span a broader range of tasks. We report all numbers on the combined dataset of the training and test splits. To test the generality of the evolved Skill Bank across different underlying models, we run every experiment on two backbones: **GPT-5.5** with medium reasoning effort and **GLM-5.2** with high reasoning effort. The Skill Bank is evolved iteratively using trajectories collected from a GPT-5.5, and is then applied unchanged to both backbones at test time. For each backbone we compare two configurations:

- **baseline**: the model with its default prompt and native tool-calling, without any Skill Bank or skill orchestration.
- **TRACE (Ours)**: the same model augmented with our self-evolved Skill Bank via state-conditioned skill orchestration.

Each task is run 3 times, and we report both metrics macro-averaged over the three task types.

3.2 Main Results

Tables 1 and 2 report all results on the full CAR-bench dataset. Table 1 gives the overall Pass^k and Pass@k for $k \in \{1, 2, 3\}$, and Table 2 breaks Pass^3 down by the three task types. Since the two configurations share the same model, base prompt, and native tool-calling and differ only in whether state-conditioned skill orchestration is enabled, the gap between the corresponding rows isolates the contribution of the skills derived from TRACE.

Overall Reliability and Gap Closure. The baselines expose precisely the gap CAR-bench targets: Pass^k decays with k while Pass@k rises, leaving a 19.9-point spread on GLM-5.2 (Pass^3 62.8% vs. Pass@3 82.7%) and 27.8 on GPT-5.5 (59.9% vs. 87.7%) between what a model *can* do and what it does on *every* trial. TRACE lifts reliability on both backbones: Pass^3 rises to 84.8% (+22.0) on GLM-5.2 and 94.5% (+34.6) on GPT-5.5, shrinking the Pass^3-to-Pass@3 gap to 12.0 and 4.0 points. The gains thus favor reliability over single-trial capability: Pass^3 improves by 22.0 points on GLM-5.2 and 34.6 points on GPT-5.5, compared with 21.7 and 22.2 points for Pass^1, respectively.

Skill Evolution and Cross-Backbone Transfer. As the baseline and TRACE configurations differ only in state-conditioned skill orchestration, contrasting the two rows provides a controlled comparison of the final Skill Bank's effect: adding it is the only change to the pipeline, yet every cell of Table 1 improves substantially on both backbones (e.g. Pass^3 by +22.0 and +34.6 points), so the gains stem from the evolved competencies rather than the model, prompt, or tooling. Because this bank was evolved solely on GPT-5.5 trajectories yet applied unchanged to GLM-5.2, its comparable GLM-5.2 gains further show the competencies transfer across backbones rather than overfitting their source model.

Performance across Task Types. Both baselines are highly uneven (Table 2), and *Disambiguation* is by far the weakest type on both backbones (48.2% on GLM-5.2 and 39.3% on GPT-5.5). Their relative strengths otherwise differ: GLM-5.2 is strongest on *Base* (76.0%), whereas GPT-5.5 is strongest on *Hallucination* (73.5%), underscoring the challenge of consistently resolving ambiguity. TRACE improves every type but most where the baseline is weakest: *Disambiguation* rises to 83.9% (+35.7) on GLM-5.2 and 94.6% (+55.3, its largest single gain) on GPT-5.5. This flattens the profile, narrowing the cross-type spread from 27.8 to 9.4 points on GLM-5.2 and 34.2 to 1.1 on GPT-5.5, so the evolved skills concentrate their effect on the hard, safety-critical behaviors that motivate the benchmark.

3.3 Official Hidden-Set Evaluation

The official evaluation additionally tests the submitted systems on a previously unseen hidden set, distinct from the combined training and test splits reported above. Table 3 compares TRACE with the baseline under the same GPT-5.6-Sol backbone over 30 hidden tasks, with three trials per task.

Reliability and Generalization. TRACE improves the strict three-trial success rate, Pass^3, from 50.0% to 70.0%: a gain of 20.0 percentage points. The gains also hold under the less stringent metrics, with Pass@3 increasing by 16.6 points and Pass@1 by 10.0 points. At the trial level, TRACE completes 69 of 90 trials successfully, 15 more than the baseline, while success consistency rises from 85.0% to 88.0%. These improvements on tasks unavailable during skill evolution pro-

Method	Pass^3 ↑	Pass@3 ↑	Pass@1 ↑	Successful trials ↑	Consistency ↑	Latency (s) ↓	Tokens/trial ↓	Cost/trial ↓
baseline	50.0	66.7	60.0	54/90	85.0	**21.21**	**82,179**	**$0.17**
TRACE	**70.0** (↑ 40.0%)	**83.3** (↑ 25.0%)	**70.0** (↑ 16.7%)	**69/90** (↑ 27.8%)	**88.0** (↑ 3.5%)	25.87 (↑ 22.0%)	141,684 (↑ 72.4%)	$0.27 (↑ 58.8%)

Table 3: Official results on the CAR-bench hidden set using GPT-5.6-Sol. Pass and consistency values are percentages; latency is the median task latency, tokens are the mean per trial, and cost is estimated per trial. Parenthetical annotations next to TRACE report relative changes with respect to the baseline (in percent). Arrows indicate the preferred direction. The best result in each column is highlighted in **bold**.

vide evidence that the Skill Bank transfers beyond the public evaluation tasks rather than merely memorizing them.

Latency–Accuracy Trade-Off. The 20.0-point gain in Pass^3 comes with a 4.66-second increase in median task latency, from 21.21 to 25.87 seconds. Thus, the 40.0% relative reliability gain is substantially larger than the relative 22.0% latency increase. This result indicates that per-turn state-conditioned skill orchestration adds only a moderate wall-clock overhead while materially improving reliable task completion. The computational overhead is more pronounced in model usage: mean tokens per trial increase by 59,505 (72.4%), and estimated cost rises by $0.10 per trial (58.8%). TRACE therefore offers a favorable reliability–latency trade-off on the hidden set, although its token and monetary costs remain important targets for future optimization.

4 Conclusions and Limitations

We presented **TRACE**: a skill-based agent whose behavioral knowledge is a self-evolving Skill Bank that rewrites modular markdown competencies from clustered, deployment-faithful evaluation evidence under a strict de-hardcoding guide. During deployment, TRACE adds state-conditioned skill orchestration on top of the base model: the Actor selects and grounds a small set of competencies at each turn, keeping the active context focused. The current orchestrator is implemented by the LLM evaluating all skill descriptions against the dialogue history at each turn, which suits the small bank used here but scales poorly as the bank grows. A learned or hierarchical orchestrator is a natural next step. The Skill Bank also acts as a forward guide, with no feedback channel from task execution back into the skills during deployment. Introducing such a channel could enable the system to adapt or correct its behavior mid-dialogue.

References

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

[Chen *et al.*, 2026] Tingyang Chen, Shuo Lu, Kang Zhao, Weicheng Meng, Hanlin Teng, Tianhao Li, Chao Li, Xule Liu, Jian Liang, Zhizhong Zhang, Yuan Xie, Heng Qu, Kun Shao, and Jian Luan. Harnessx: A composable, adaptive, and evolvable agent harness foundry. *arXiv preprint arXiv:2606.14249*, 2026.

[Guo *et al.*, 2025] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, et al. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645(8081):633–638, 2025.

[Han *et al.*, 2025] Guangzeng Han, Weisi Liu, and Xiaolei Huang. Attributes as textual genes: Leveraging LLMs as genetic algorithm simulators for conditional synthetic data generation. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 19367–19389, 2025.

[Hu *et al.*, 2025] Zican Hu, Wei Liu, Xiaoye Qu, Xiangyu Yue, Chunlin Chen, Zhi Wang, and Yu Cheng. Divide and conquer: Grounding LLMs as efficient decision-making agents via offline hierarchical reinforcement learning. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267, pages 24570–24590, 2025.

[Hu *et al.*, 2026] Zican Hu, Shilin Zhang, Yafu Li, Jianhao Yan, Xuyang Hu, Leyang Cui, Xiaoye Qu, Chunlin Chen, Yu Cheng, and Zhi Wang. Diversity-incentivized exploration for versatile reasoning. In *Proceedings of International Conference on Learning Representations*, 2026.

[Jaech *et al.*, 2024] Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, et al. Openai o1 system card. *arXiv preprint arXiv:2412.16720*, 2024.

[Jimenez *et al.*, 2024] Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues? In *International Conference on Learning Representations*, volume 2024, pages 54107–54157, 2024.

[Kalai *et al.*, 2025] Adam Tauman Kalai, Ofir Nachum, Santosh S. Vempala, and Edwin Zhang. Why language models hallucinate. *arXiv preprint arXiv:2509.04664*, 2025.

[Kirmayr *et al.*, 2026] Johannes Kirmayr, Lukas Stappen, and Elisabeth Andre. CAR-bench: Evaluating the consistency and limit-awareness of LLM agents under real-world uncertainty. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 40599–40618, 2026.

[Liu *et al.*, 2026] Xule Liu, Hanlin Teng, Chao Li, Yanan Ni, Shuo Lu, Audrey Wang, Yijun Liu, Yunfei Wang, Xiaofeng Li, Xian Yi, Yuanfa Li, Kang Zhao, Jian Liang, Yuxuan Chen, Jinyuan Chen, Heng Qu, Kun Shao, and Jian Luan. Mi-Memory: A lifecycle memory framework for personal ai. *arXiv preprint arXiv:2607.18975*, 2026.

[Shinn *et al.*, 2023] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning.

In *Advances in Neural Information Processing Systems*, volume 36, pages 8634–8652, 2023.

[Wang *et al.*, 2023] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *Proceedings of International Conference on Learning Representations*, 2023.

[Wei *et al.*, 2022] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, volume 35, pages 24824–24837, 2022.

[Wu *et al.*, 2026] Wenhao Wu, Zhentao Tang, Yafu Li, Shixiong Kai, Mingxuan Yuan, Zhenhong Sun, Chunlin Chen, and Zhi Wang. From conflict to consensus: Boosting medical reasoning via multi-round agentic RAG. In *Forty-third International Conference on Machine Learning*, 2026.

[Yan *et al.*, 2025] Jianhao Yan, Yafu Li, Zican Hu, Zhi Wang, Ganqu Cui, Xiaoye Qu, Yu Cheng, and Yue Zhang. Learning to reason under off-policy guidance. In *Advances in Neural Information Processing Systems*, volume 38, pages 117157–117186, 2025.

[Yao *et al.*, 2023] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2023.

[Zhan *et al.*, 2026] Runzhe Zhan, Yafu Li, Zhi Wang, Xiaoye Qu, Dongrui Liu, Jing Shao, Derek F Wong, and Yu Cheng. ExGRPO: Learning to reason from experience. In *Proceedings of International Conference on Learning Representations*, 2026.

[Zhang *et al.*, 2026] Haoran Zhang, Yafu Li, Zhi Wang, Zhilin Wang, Shunkai Zhang, Xiaoye Qu, and Yu Cheng. Characterizing, evaluating, and optimizing complex reasoning. In *Forty-third International Conference on Machine Learning*, 2026.