

The Confidence Dichotomy: Analyzing and Mitigating Miscalibration in Tool-Use Agents

Weihaio Xuan^{1,2*}, Qingcheng Zeng^{3*}, Heli Qi^{2,4}, Yunze Xiao⁵,
Junjue Wang¹, Naoto Yokoya^{1,2†}

¹The University of Tokyo, ²RIKEN AIP, ³Northwestern University
⁴Waseda University, ⁵Carnegie Mellon University

Abstract

Autonomous agents based on large language models (LLMs) are rapidly evolving to handle multi-turn tasks, but ensuring their trustworthiness remains a critical challenge. A fundamental pillar of this trustworthiness is calibration, which refers to an agent’s ability to express confidence that reliably reflects its actual performance. While calibration is well-established for static models, its dynamics in tool-integrated agentic workflows remain under-explored. In this work, we systematically investigate verbalized calibration in tool-use agents, revealing a fundamental confidence dichotomy driven by tool type. Specifically, our pilot study identifies that evidence tools (e.g., web search) systematically induce severe overconfidence due to inherent noise in retrieved information, while verification tools (e.g., code interpreters) can ground reasoning through deterministic feedback and mitigate miscalibration. To robustly improve calibration across tool types, we propose a reinforcement learning (RL) fine-tuning framework that jointly optimizes task accuracy and calibration, supported by a holistic benchmark of reward designs. We demonstrate that our trained agents not only achieve superior calibration but also exhibit robust generalization from local training environments to noisy web settings and to distinct domains such as mathematical reasoning. Our results highlight the necessity of domain-specific calibration strategies for tool-use agents. More broadly, this work establishes a foundation for building self-aware agents that can reliably communicate uncertainty in high-stakes, real-world deployments.

1 Introduction

Autonomous agents based on LLMs represent a transformative leap in artificial intelligence, evolving beyond static text processing to engage with

dynamic, real-world environments actively. By leveraging external tools and iterative reasoning, these systems are demonstrating rapid increases in proficiency in executing complex, long-horizon tasks that were previously intractable. The research community has recently witnessed remarkable strides across critical domains, including sophisticated code agents (Jimenez et al., 2024; Yang et al., 2024, 2025; Trae Research Team et al., 2025), autonomous web agents (He et al., 2024; Wei et al., 2025b), and advanced deep search agents (Wu et al., 2025a; Li et al., 2025b,a; Team et al., 2025). The implications of this progress extend far beyond technical novelty. Indeed, these agents are poised to exert a substantial economic impact and fundamentally reshape the future of work (Shao et al., 2025).

While the trustworthiness and safety of LLMs have been examined extensively, research specifically addressing trustworthiness in multi-turn agents remains sparse (Hua et al., 2024; Yu et al., 2025; Shi et al., 2026). A fundamental pillar of this trustworthiness is *calibration*, which refers to an agent’s ability to report confidence scores that reliably reflect its actual performance. Within the limited existing literature, search agent-oriented benchmarks (Wei et al., 2025a; Zhou et al., 2025) consistently report that tool-use agents exhibit higher calibration errors than standalone LLMs, suggesting that external tools often exacerbate overconfidence. However, these studies focus narrowly on search scenarios, leaving open a critical question: *is miscalibration a universal consequence of tool use, or does it depend on the nature of the tool itself?*

In this work, we present a systematic pilot study that answers this question by comparing representative tool-use agents (Jin et al., 2025; Xue et al., 2025) against their standard instruction-following counterparts. Our analysis reveals a critical **confidence dichotomy**: not all tools affect calibration equally. Specifically, we identify two distinct

*Equal contribution.

†Corresponding author.

tool categories with opposing effects. **Evidence tools** (e.g., web search), which retrieve external information laden with noise and uncertainty, systematically induce severe overconfidence. In contrast, **verification tools** (e.g., code interpreters), which provide deterministic execution feedback, can ground reasoning and mitigate miscalibration. This dichotomy persists across both prompting-based strategies and standard tool-use-oriented RL, indicating a fundamental challenge that cannot be resolved through prompt engineering alone.

To improve calibration across tool-use scenarios, we propose the **Calibration Agentic RL (CAR)** framework, a novel RL-based fine-tuning approach that jointly optimizes task accuracy and the reliability of expressed confidence. Through a holistic evaluation of diverse reward structures, we demonstrate that our trained agents maintain competitive accuracy while achieving significantly better calibration than baselines. Furthermore, we validate the robustness of our approach: agents trained in controlled local retriever environments generalize effectively to more noisy, API-based web search scenarios, and the framework proves effective in tool-integrated mathematical reasoning. Our primary contributions are summarized as follows:

- We conduct a systematic pilot study that reveals a critical confidence dichotomy in tool-use agents: while verification tools provide grounding, evidence tools inherently predispose agents to severe overconfidence.
- We propose CAR, an RL-based framework for optimizing agent calibration, supported by a holistic benchmark of reward structures, including our novel Margin-Separated Calibration Reward (MSCR), that provides key insights for future reward design.
- We validate the effectiveness and robustness of our methodology across distinct task domains and demonstrate successful cross-environment generalization from local to noisy web settings.

2 Related Work

Tool-use Agents The paradigm of LLMs is shifting from static text generation to autonomous agents capable of interacting with external environments. Equipping agents with tool-use capabilities enables them to overcome inherent limitations,

such as hallucinations and calculation errors. Recent literature has explored diverse tool integration strategies, which can be broadly categorized by their function. On one hand, *evidence tools* such as web search enable agents to retrieve external information to augment their knowledge; for instance, Search-R1 (Jin et al., 2025) leverages RL to intrinsically motivate agents to seek information when their internal knowledge is insufficient. On the other hand, *verification tools* such as code interpreters provide deterministic feedback to validate reasoning steps. Xue et al. (2025) integrates external Python interpreters to enhance agents’ mathematical reasoning capabilities through robust code execution. Emerging evidence from search-oriented benchmarks suggests that tool use can exacerbate overconfidence in these settings (Wei et al., 2025a; Zhou et al., 2025). Our work extends this line of inquiry by showing that this effect is not universal, but instead depends systematically on tool type.

Calibration in LLMs The calibration of LLMs has emerged as a central theme in recent literature. Verbalized confidence (Lin et al., 2022; Tian et al., 2023) has gained prominence due to its interpretability and simplicity, with subsequent evaluations across instruct LLMs (Xiong et al., 2024), reasoning LLMs (Zeng et al., 2025; Yoon et al., 2025), and vision-language models (Xuan et al., 2025) consistently characterizing these models as exhibiting moderate overconfidence. In terms of training, Damani et al. (2025) proposed an RL-based framework to encourage calibration in single-turn LLMs. However, these studies focus on static generation and do not address how calibration behaves in interactive tool-use settings.

Calibration in Tool-Use Agents Recent work has begun to study calibration specifically in tool-using agents. Subramani et al. (2025) proposed MICE, which trains a classifier over model-internal logit-lens features to estimate whether a generated tool call should be executed. Shen et al. (2024) introduced SMARTCAL, a multi-agent inference-time framework that mitigates tool abuse via self-evaluation and empirical step-level recalibration. Liu et al. (2024) proposed ProbeCal, which trains an external probe on LLM embeddings to estimate execution reward for prompt and trace selection, while Zhang et al. (2026) trained an external predictor to estimate trajectory success post hoc. These approaches improve reliability through aux-

iliary inference-time or post-hoc modules rather than training the agent to internalize calibrated verbalized confidence. Meanwhile, ET-Agent (Chen et al., 2026) uses the notion of *behavior calibration* to improve tool-use efficiency by reducing ineffective interaction patterns, which is orthogonal to our focus on statistical confidence calibration. In contrast to these works, we identify a tool-type-dependent confidence dichotomy and directly optimize the agent’s intrinsic verbalized calibration during RL training.

3 Pilot Study

3.1 Overall Motivation

The primary objective of this pilot study is to answer the question raised in our introduction: *is miscalibration a universal consequence of tool use, or does it depend on the nature of the tool itself?* To this end, we systematically isolate and quantify the impact of different tool types on LLM calibration. We specifically analyze the confidence shifts that occur when transitioning from standard text generation to tool-augmented agentic workflows. By contrasting distinct operational modes, including standard prompting, prompting-based tool use, and RL-optimized tool use, we aim to elucidate how invoking external tools fundamentally modulates the reliability of model confidence.

3.2 Experimental Setup

To ensure a rigorous comparison, we establish three distinct experimental configurations:

1. **Direct Prompting:** The LLM is instructed to address queries utilizing only its internal parametric knowledge.
2. **Prompting-based Tool-Use:** The LLM is prompted to function as an autonomous agent with the capacity to invoke external tools, without updates to its model weights.
3. **RL-based Tool-Use:** This setting adopts similar agentic prompts as the second configuration. However, the model is fine-tuned explicitly via RL to enhance its multi-turn tool interaction capabilities.

We focus on web search and code interpreter as they represent two fundamental paradigms of agentic tool use. Evidence tools, exemplified by web search, are characterized by open-ended, stochastic outputs containing noisy, unstructured information.

These properties are shared by other information-retrieval tools such as RAG and API queries. Conversely, verification tools, exemplified by code interpreters, provide deterministic, structured feedback that facilitates logical grounding. These properties are shared by other execution-based tools such as calculators, SQL, and symbolic solvers. Most other tools used by agents fall within the spectrum defined by these two paradigms. We evaluate three configurations across two representative tool categories using *Qwen2.5-3B-Instruct* (Qwen et al., 2025) as the backbone model:

Evidence Tools (Web Search): We evaluate open-domain question answering performance using the NQ (Kwiatkowski et al., 2019) and HotpotQA (Yang et al., 2018) datasets. These tasks require agents to retrieve external information from a noisy retrieval environment. For the RL-based variant, we adopt the training configurations of Search-R1 (Jin et al., 2025) using the VeRL framework (Sheng et al., 2024).

Verification Tools (Code Interpreter): We assess mathematical reasoning capabilities using the AIME2024/2025 (Art of Problem Solving, 2025) and MATH-500 (Lightman et al., 2023) benchmarks. These tasks allow agents to execute Python code and receive deterministic feedback. The RL-based model is implemented following the SimpleTIR (Xue et al., 2025) methodology for tool-integrated reasoning. Detailed training and evaluation configurations are provided in Appendix C.

We measure the Mean Confidence on Incorrect Predictions (MCIP) on the intersectional wrong questions across the three settings to understand how additional tool use affects confidence, defined as $MCIP = \frac{1}{|\mathcal{D}_{\text{wrong}}|} \sum_{i \in \mathcal{D}_{\text{wrong}}} \hat{p}_i$, where $\mathcal{D}_{\text{wrong}} = \{i \mid y_i^* \neq y_i\}$ denotes the set of incorrectly answered questions and \hat{p}_i is the model’s predicted confidence for its chosen answer on example i . Specifically, we use this metric to check whether agents show significantly different confidence patterns across various tool-use configurations. To ensure the power of our analysis, we additionally conducted t -tests (STUDENT, 1908) between each configuration.

3.3 Results

The empirical results of our pilot study, detailed in Table 1, reveal a critical **confidence dichotomy** in agent behavior. First, consistent with observations by Wei et al. (2025a), integrating evidence

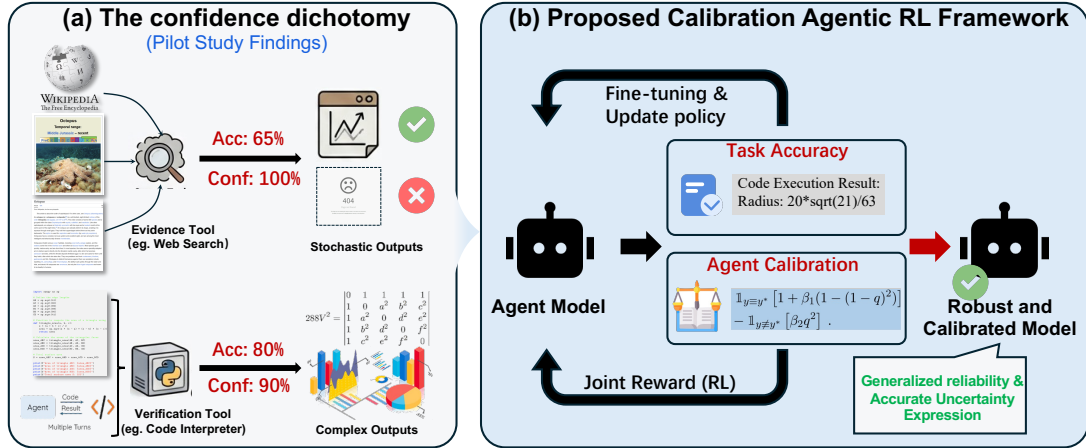


Figure 1: **The confidence dichotomy and the proposed RL framework.** (a) Our pilot study reveals a tool-dependent divergence in calibration: *evidence tools* (e.g., web search), which operate in noisy retrieval environments, systematically induce overconfidence. In contrast, *verification tools* (e.g., code interpreters), which provide deterministic execution feedback, exhibit better alignment between confidence and accuracy. (b) To address this miscalibration, we fine-tune agents with a joint RL objective that combines task accuracy and calibration rewards, producing robust agents with reliable uncertainty expression.

Domain	Dataset	Setting	Acc. \uparrow	MCIP \downarrow
Search	NQ	Direct prompting	15.5	0.879
		Prompting-based tool-use	34.3	0.901
		RL-based tool-use	43.1	0.948
	HotpotQA	Direct prompting	18.7	0.859
		Prompting-based tool-use	22.4	0.911
Code	AIME2024	Direct prompting	12.2	0.968
		Prompting-based tool-use	4.4	0.915
		RL-based tool-use	18.2	0.868
	AIME2025	Direct prompting	15.7	0.968
		Prompting-based tool-use	5.8	0.904
		RL-based tool-use	19.8	0.879
	MATH-500	Direct prompting	63.4	0.971
		Prompting-based tool-use	48.6	0.913
		RL-based tool-use	77.0	0.890

Table 1: Pilot study results across tool-use configurations and domains using *Qwen2.5-3B-Instruct* as the backbone. Accuracy (Acc.) is reported for all tasks. Lower MCIP indicates fewer overconfidence issues.

tools (i.e., web search) leads to a marked deterioration in calibration. Regardless of whether the agent employs prompting-based strategies or RL-enhanced capabilities, the presence of search tools systematically exacerbates overconfidence. Moreover, our *t*-tests confirm that these differences are statistically significant across direct prompting, prompting-based, and RL-based tool-use configurations. This suggests that the inherent noise and ambiguity of retrieved information predispose agents to inflated certainty. Conversely, verification tools (i.e., code interpreters) yield a sharply contrasting dynamic. In this domain, tool usage mitigates overconfidence, with RL-based agents achieving the lowest MCIP (statistically significant in the reverse

order). These divergent outcomes challenge the assumption that tool use exerts a uniform influence on agent confidence. Instead, we identify a clear heterogeneity driven by tool type: evidence tools, which introduce external information with inherent noise, interfere with confidence estimation, whereas verification tools, which provide deterministic feedback, ground the reasoning process and temper unwarranted certainty. This dichotomy motivates our development of calibration-aware training strategies for evidence tool scenarios. We further validate that this phenomenon persists at larger model scales (72B) in [Appendix D](#).

4 Calibration Agentic RL (CAR)

As demonstrated in our pilot study, tool use does not affect calibration uniformly. Verification tools can ground reasoning through deterministic feedback, whereas evidence tools systematically induce severe overconfidence because retrieved information is noisy and only weakly verifiable. This identifies evidence-tool settings as the primary regime in which calibration must be improved. Motivated by this finding, we introduce the **Calibration Agentic RL (CAR)** framework, a calibration-aware training framework for multi-turn agentic tasks that explicitly optimizes not only answer correctness but also the reliability of the agent’s verbalized confidence.

The pilot study further clarifies what such a framework must achieve. Since miscalibration remains pronounced in evidence-tool agents even after standard RL for tool use, calibration cannot be

treated as a by-product of improved task capability and must instead be optimized directly. Moreover, unlike verification tools, evidence tools provide little direct feedback when retrieved information is insufficient or misleading. A calibration-aware objective must therefore do more than reward accuracy and confidence jointly: it must also discourage optimization dynamics in which unsupported answers remain competitive simply by expressing lower confidence. CAR is designed with these two requirements in mind, combining explicit calibration optimization with reward formulations that encourage a clear separation between well-grounded success and unsupported failure.

4.1 Experimental Setup

4.1.1 Training Details

We first focus on evidence tool scenarios where tool use exacerbates miscalibration, leveraging the Search-R1 framework (Jin et al., 2025) as our primary testbed. For the retrieval component, we directly adopt the established local search engine configuration utilized within Search-R1. This setup employs the 2018 Wikipedia dump (Karpukhin et al., 2020) as the knowledge source, coupled with E5 (Wang et al., 2024) as the dense retriever. Furthermore, to optimize the policy of our tool-use agents, we employ Group Relative Policy Optimization (GRPO) (Shao et al., 2024) as the foundational RL algorithm, and the mixture of NQ (Kwiatkowski et al., 2019) and HotpotQA (Yang et al., 2018) as our training data. Our training prompt templates are attached in Appendix B.

We employ a suite of instruction-tuned LLMs as our policy networks: *Qwen2.5-3B-Instruct*, *Qwen2.5-7B-Instruct*, and *Qwen3-4B-Instruct-2507*. The selection of instruction-tuned variants is driven by two factors. First, these models exhibit superior adherence to complex instructions, facilitating more reliable verbalized confidence reporting. Second, empirical evidence from the original Search-R1 study indicates negligible performance disparities between base and instruction-tuned architectures in this context.

4.1.2 Reward Design

Baseline Methods We evaluate CAR against three baseline methodologies: (1) **Vanilla Search-R1**: the reward architecture proposed by Jin et al. (2025), which utilizes an exact match (EM) outcome reward alongside a structural reward to enforce adherence to the reasoning-action-

observation chain; (2) **Temperature Scaling**: a verified post-hoc calibration method (Guo et al., 2017) applied to the vanilla Search-R1 baseline with temperature fixed at 1.5. Superior performance over this baseline would indicate that the model has internalized genuine calibration capabilities rather than merely adjusting surface-level probabilities; and (3) **MASH**: Gul et al. (2026) introduced a penalty mechanism for excessive search tool usage that fosters robust abstention behavior. Given the correlation between selective abstention and improved calibration (Kirichenko et al., 2025; Song et al., 2025), we include this as a comparative baseline.

Proposed Reward Architecture Following Damani et al. (2025), we augment the system prompt to require agents to output a numerical confidence score (ranging from 0 to 100) within `<confidence>` XML tags. Our reward design comprises two components:

(1) Extended Format Reward To ensure structural integrity, we extend the standard Search-R1 format constraints. While the original formulation validates the logical ordering of reasoning, action, and observation, our design additionally mandates the presence of the confidence tag. Consequently, the boolean reward function $f_{format}(\tau)$ returns a value of *True* only when all structural requirements, including the confidence encapsulation, are strictly satisfied.

(2) Calibration-motivated Outcome Reward We reward both the accuracy of the final answer and the expressed confidence. Given the gold answer y , predicted answer y^* , and verbalized confidence q , we experiment with two formulations:

Weighted Brier Score Reward. Following Damani et al. (2025), we use the Brier score (Brier, 1950) to form a combined reward:

$$R(y, q, y^*) = \mathbb{1}_{y=y^*} - \lambda(q - \mathbb{1}_{y=y^*})^2. \quad (1)$$

When $\lambda = 1$, this reduces to the RLCR formulation. However, in this setting, the lowest reward for a correct attempt equals the highest reward for an incorrect one, which may make learning sensitive to the training data distribution. To restore a positive incentive margin for correctness, we experiment with a weighting coefficient $\lambda = 1/3$ on the Brier term.

Margin-Separated Calibration Reward (MSCR). To address the optimization instability

caused by incentive overlap in Brier scores, we propose MSCR. This formulation decouples calibration terms for correct and incorrect predictions to guarantee a strict reward margin:

$$R_{\text{MSCR}}(y, q, y^*) = \mathbb{1}_{y=y^*} [1 + \beta_1(1 - (1 - q)^2)] - \mathbb{1}_{y \neq y^*} [\beta_2 q^2], \quad (2)$$

where $\beta_1, \beta_2 > 0$ control the calibration magnitude. This formulation enforces strict separation: correct answers receive a base reward of at least 1 (even with $q = 0$), while incorrect answers receive at most 0 (at $q = 0$) and incur penalties for false confidence. This eliminates the ‘‘safe failure’’ loophole, ensuring that the least confident correct answer strictly outperforms the most ‘‘honest’’ incorrect answer (see Appendix E for formal analysis). Unless otherwise stated, we use $\beta_1 = \beta_2 = 0.5$ in the main experiments and study additional MSCR configurations in Appendix H.

Unified Reward Function The total reward combines format constraints with calibration-aware scoring. Denoting the model output as τ , the chosen calibration function as $\mathcal{R}_{\text{cal}}(y, q, y^*)$ and assigning a penalty λ_f for format violations:

$$r_\phi(x, \tau) = \begin{cases} \mathcal{R}_{\text{cal}}(y, q, y^*) & \text{if } f_{\text{format}}(\tau), \\ \mathcal{R}_{\text{cal}}(y, q, y^*) - \lambda_f & \text{otherwise.} \end{cases} \quad (3)$$

In the *otherwise* case, if q cannot be extracted, we fall back to a minimal calibration reward (treating q as a default value) to maintain the correctness gradient while penalizing format errors via λ_f .

4.1.3 Evaluation Details

We evaluate our trained agents on the following benchmarks: (1) the validation sets of NQ and HotpotQA, serving as in-distribution (ID) datasets; and (2) SimpleQA-verified (Haas et al., 2025), a curated subset of SimpleQA (Wei et al., 2024) comprising 1,000 rigorously filtered questions for out-of-distribution (OOD) assessment. The retrieval corpus remains the 2018 Wikipedia dump for all local-retriever evaluations. We employ four metrics for comprehensive analysis: (1) Accuracy (Acc.), to verify that calibration improvements do not come at the cost of task performance; (2) Expected Calibration Error (ECE), the canonical metric for confidence-accuracy alignment, calculated using a 10-bin scheme; (3) Brier Score, which captures

both calibration and refinement as the squared difference between confidence and correctness; and (4) AUROC, to assess whether agents can reliably distinguish correct from incorrect predictions via confidence ranking. More details are provided in Appendix A.

5 Results

5.1 General Results on Search Agents

A comprehensive summary of our experimental findings is presented in Table 2. The results demonstrate the robust effectiveness of CAR: across all three backbone models with different sizes, we consistently observe substantial ECE reductions compared to baseline methods. This improvement is consistent across both in-distribution (ID) and out-of-distribution (OOD) settings, with ECE relative reductions of up to 68% through explicit calibration-aware RL training. Crucially, under our optimal configuration (MSCR), agents maintain accuracy levels competitive with reward structures that strictly incentivize correctness, confirming that our design successfully balances calibration and task performance. Furthermore, our analyses of AUROC and temperature scaling suggest that CAR achieves these gains not through mere rescaling of confidence scores, but by inducing more nuanced confidence reasoning. This is evidenced by AUROC relative improvements of up to 17% in our best setting, confirming that the model has genuinely improved its ability to distinguish correct from incorrect outputs.

Although the full RL sweeps in the main table focus on 3B to 7B backbones due to the cost of agentic RL training, the observed trends are not confined to this scale range. Appendix I shows that CAR continues to improve both calibration and accuracy on *Qwen2.5-14B-Instruct* under both local retrieval and the Serper API, while Appendix D confirms that the confidence dichotomy itself persists at the 72B scale.

This improved reasoning capability translates into robust generalization, as evidenced by the comparison between ID and OOD settings. Our results indicate that CAR engenders a fundamental understanding of confidence rather than a superficial alignment with in-distribution data. Specifically, on the SimpleQA-verified dataset, agents trained via CAR exhibit marked calibration improvements. This finding suggests that the calibration mechanisms learned in search scenarios are not artifacts

Model	Strategy	NQ (ID)				HotpotQA (ID)				SimpleQA-verified (OOD)			
		Acc \uparrow	ECE \downarrow	Brier \downarrow	AUROC \uparrow	Acc \uparrow	ECE \downarrow	Brier \downarrow	AUROC \uparrow	Acc \uparrow	ECE \downarrow	Brier \downarrow	AUROC \uparrow
Qwen2.5-3B	Vanilla Search-R1	43.1	0.528	0.519	0.600	27.4	0.699	0.686	0.599	34.8	0.610	0.587	0.702
	Temperature Scaling	43.1	0.500	0.489	0.600	27.4	0.674	0.651	0.599	34.8	0.583	0.549	0.702
	MASH	43.4	0.479	0.470	0.612	28.9	0.656	0.631	0.598	35.1	0.589	0.560	0.696
	CAR (Weighted Brier, $\lambda=1$)	22.3	0.091	0.091	0.941	24.3	0.148	0.148	0.902	21.2	0.027	0.027	0.983
	CAR (Weighted Brier, $\lambda=1/3$)	44.5	0.307	0.313	0.677	28.5	0.329	0.329	0.651	35.8	0.203	0.203	0.740
	CAR (MSCR)	45.5	0.303	0.303	0.699	29.2	0.286	0.289	0.688	36.6	0.192	0.191	0.763
Qwen2.5-7B	Vanilla Search-R1	61.1	0.363	0.367	0.563	54.8	0.424	0.423	0.551	40.7	0.441	0.398	0.776
	Temperature Scaling	61.1	0.311	0.330	0.563	54.8	0.371	0.380	0.551	40.7	0.385	0.347	0.776
	MASH	67.0	0.309	0.315	0.543	55.6	0.421	0.422	0.521	41.5	0.465	0.420	0.775
	CAR (Weighted Brier, $\lambda=1$)	65.2	0.221	0.242	0.693	48.7	0.330	0.358	0.583	24.4	0.050	0.053	0.940
	CAR (Weighted Brier, $\lambda=1/3$)	67.7	0.281	0.293	0.629	52.8	0.368	0.379	0.600	40.5	0.177	0.176	0.798
	CAR (MSCR)	69.3	0.238	0.255	0.637	56.8	0.326	0.348	0.641	40.9	0.150	0.156	0.837
Qwen3-4B	Vanilla Search-R1	45.7	0.452	0.438	0.634	46.6	0.408	0.391	0.671	42.2	0.287	0.210	0.874
	Temperature Scaling	45.7	0.380	0.377	0.634	46.6	0.343	0.340	0.671	42.2	0.254	0.198	0.874
	MASH	36.3	0.543	0.514	0.645	35.5	0.498	0.457	0.695	37.4	0.299	0.213	0.859
	CAR (Weighted Brier, $\lambda=1$)	28.5	0.115	0.120	0.918	36.6	0.184	0.186	0.856	33.1	0.036	0.033	0.973
	CAR (Weighted Brier, $\lambda=1/3$)	44.2	0.274	0.274	0.718	45.0	0.281	0.281	0.727	40.1	0.127	0.127	0.912
	CAR (MSCR)	45.5	0.272	0.272	0.724	45.9	0.269	0.270	0.740	41.8	0.106	0.106	0.929

Table 2: Main results organized by backbone model. Dashed lines separate baselines from CAR variants.

of the training set but instead transferable skills that generalize reliably to less familiar queries.

A comparative analysis of the three CAR configurations reveals the critical role of reward gap magnitude. The weighted Brier score with $\lambda=1$ (i.e., vanilla RLCR) yields the lowest ECE but suffers from significant accuracy degradation, indicating reward hacking behavior. In contrast, MSCR achieves a superior accuracy-calibration trade-off: across most settings, it attains higher accuracy than the weighted Brier variant with $\lambda=1/3$ while simultaneously improving calibration. These results suggest that strict reward separation is essential for robust calibration training. We further verify this in Appendix H: across a broad range of (β_1, β_2) settings on both ID and OOD benchmarks, all MSCR variants remain stable and avoid the performance collapse observed with weighted Brier $\lambda=1$.

5.2 Tool generalization

Our primary experiments employed a simulated retriever environment with a static Wikipedia dump. However, real-world deployment poses additional challenges: commercial API-based retrievers often exhibit stochastic behavior and return noisy or extraneous information. In this section, we investigate whether the calibration capabilities learned in controlled settings transfer to these more challenging, API-driven environments.

Setup We use the Serper API as our retrieval backbone and evaluate both vanilla Search-R1 and CAR (MSCR) on the SimpleQA-verified benchmark. Additional results examining corpus shift on BrowseComp (Wei et al., 2025a) and Web-WalkerQA (Wu et al., 2025b) are provided in Ap-

Model	Method	SimpleQA-verified (Serper API)			
		Acc \uparrow	ECE \downarrow	Brier \downarrow	AUROC \uparrow
Qwen2.5-3B	Vanilla S-R1	76.18	0.213	0.219	0.659
	CAR (MSCR)	76.18	0.175	0.175	0.823
Qwen2.5-7B	Vanilla S-R1	70.28	0.204	0.204	0.831
	CAR (MSCR)	71.01	0.176	0.180	0.790
Qwen3-4B	Vanilla S-R1	85.27	0.140	0.140	0.825
	CAR (MSCR)	84.97	0.034	0.073	0.765

Table 3: Tool generalization under a noisy API-driven retriever (Serper API). We evaluate Vanilla Search-R1 and CAR (MSCR) on SimpleQA-verified.

pendix F.

As shown in Table 3, our trained agents achieve superior calibration compared to the vanilla baseline while maintaining competitive accuracy. These results confirm that the calibration capabilities acquired in simulated environments are not brittle but transfer robustly to the stochastic and noisy conditions of real-world API interactions.

5.3 Tool-integrated Reasoning

Building on our pilot study, we extend evaluation to Tool-integrated Reasoning (TIR), where agents leverage code interpreters to solve mathematical problems. This extension allows us to examine how CAR performs with verification tools, which our pilot study identified as inducing calibration dynamics different from those of evidence tools.

Setup We utilize the SimpleTIR (Xue et al., 2025) framework with *Qwen2.5-3B-Instruct* as the backbone model. We compare two configurations: vanilla SimpleTIR as the baseline and our MSCR design. For evaluation, we adopt the AIME2024/2025 (Art of Problem Solving, 2025) and MATH-500 (Lightman et al., 2023) bench-

Model	Method	AIME2024				AIME2025				MATH-500			
		Acc \uparrow	ECE \downarrow	Brier \downarrow	AUROC \uparrow	Acc \uparrow	ECE \downarrow	Brier \downarrow	AUROC \uparrow	Acc \uparrow	ECE \downarrow	Brier \downarrow	AUROC \uparrow
Qwen2.5-3B	Vanilla SimpleTIR	18.2	0.692	0.630	0.489	19.8	0.687	0.632	0.498	77.0	0.151	0.193	0.622
	CAR (MSCR)	20.8	0.573	0.485	0.548	21.1	0.519	0.410	0.695	76.9	0.057	0.168	0.636

Table 4: Results on mathematical reasoning benchmarks of tool-integrated reasoning agents.

marks, utilizing E2B¹ as the code execution sandbox. Additional training configurations are detailed in Appendix G. Following our pilot study, we treat web search as the primary ablation testbed because noisy evidence tools create the hardest reward-overlap setting; the TIR experiments instead focus on whether the best CAR design transfers to verification tools.

Results Table 4 presents the quantitative results. Consistent with our findings in the search domain, CAR yields robust calibration improvements for TIR agents, with significant reductions in ECE and Brier scores alongside increases in AUROC. These gains persist across all evaluated benchmarks, confirming the generalizability of our reward formulation. We further extend this verification-tool evaluation to AIME2026 and OlympiadBench in Appendix J, where CAR shows the same improvement trend across all four metrics.

However, examining absolute performance reveals an important nuance. Despite these improvements, ECE metrics for TIR agents remain elevated compared to both pure reasoning models (Zeng et al., 2025) and our search-based agents. Furthermore, calibration efficacy correlates with task complexity: agents exhibit substantially lower ECE on MATH-500 than on the more challenging AIME benchmarks. These observations align with our pilot study hypothesis that verification tools, while providing grounding through deterministic feedback, still exhibit calibration dynamics that depend on task difficulty. We conclude that while explicit calibration rewards provide a necessary corrective signal, ultimate calibration performance in TIR settings remains bounded by the model’s intrinsic reasoning capabilities.

6 Discussion

Our findings reveal that tool use introduces systematic yet heterogeneous effects on agent calibration, challenging the implicit assumption that tool augmentation uniformly improves or degrades reliability. In this section, we discuss the broader implications of these findings and examine the mechanisms

underlying the observed confidence dichotomy.

From Static Elicitation to Tool-Modulated Dynamics. The transition from static LLMs to autonomous agents necessitates a fundamental re-evaluation of verbalized calibration paradigms. While extensive literature establishes that models can reasonably express uncertainty in single-turn QA (Tian et al., 2023) or standard Chain-of-Thought reasoning (Xiong et al., 2024; Zeng et al., 2025), our analysis reveals that tool integration introduces a non-trivial *heterogeneity* that disrupts this alignment. Specifically, we contextualize the severe miscalibration observed in recent web-browsing agents (Wei et al., 2025a; Zhou et al., 2025) not merely as a general capability failure, but as a symptom of the evidence tool dynamic, where stochastic retrieval artificially inflates internal certainty. This stands in sharp contrast to verification scenarios, where deterministic feedback provides the grounding often assumed but absent in open-ended search. These findings suggest that calibration research in agentic settings must account for tool-type-specific dynamics rather than treating tool use as a monolithic phenomenon.

Why Do Evidence Tools Induce Overconfidence?

Our pilot study indicates that evidence tools systematically amplify overconfidence, but the underlying mechanism warrants further examination. We attribute this phenomenon to a fundamental asymmetry in feedback signals. Verification tools such as code interpreters provide explicit execution feedback: syntax errors, runtime exceptions, and type mismatches offer clear signals that something is wrong. While successful execution does not guarantee correctness, as logical errors may still produce plausible but incorrect outputs, these tools nonetheless provide *partial grounding* via observable failure modes. In contrast, evidence tools offer little answer-level correctness feedback. A web search typically returns results, regardless of whether they are relevant, accurate, or sufficient to answer the query. The absence of negative feedback leads agents to conflate the *presence* of retrieved information with the *correctness* of their answer. This effect is compounded by a form of

¹<https://e2b.dev/>

false certainty induced by the retrieval action itself. Having performed an explicit information-seeking step, the agent treats it as “due diligence,” even when the retrieved content is noisy or misleading. Retrieved passages often contain surface-level lexical overlap with the query, which the agent mistakes for genuine evidential support. This pattern is illustrated in [Appendix M](#): when retrieved documents contain conflicting information, the agent nonetheless reports near-certain confidence, suggesting that surface-level lexical overlap overrides sensitivity to factual inconsistency.

Extending RLCR to Tool-Use Agents via MSCR.

Our approach builds on calibration-motivated RL, particularly RLCR ([Damani et al., 2025](#)), which shows that optimizing binary correctness rewards can degrade calibration by encouraging guessing and proposes incorporating calibration terms into the training objective. We extend this principle to the agentic setting. Tool-use agents face additional exogenous noise and expanded action spaces, thereby enlarging the space of degenerate solutions in which confidence becomes uninformative or strategically manipulated. Concretely, we find that reward overlap between correct and incorrect trajectories makes learning sensitive to data difficulty and can incentivize confidence collapse. Our proposed MSCR addresses this by enforcing strict separation between reward landscapes for correct and incorrect trajectories, preserving a correctness margin while still shaping confidence within each region. Empirically, this design improves calibration and failure discrimination beyond what simple rescaling would achieve, and generalizes to noisy retrieval environments, suggesting that calibration-motivated RL remains effective in multi-step tool use provided that incentive separation is maintained.

7 Conclusion

In this work, we systematically investigated the calibration dynamics of tool-use agents. Our pilot study revealed a fundamental confidence dichotomy: while verification tools provide deterministic feedback that grounds reasoning, evidence tools introduce stochastic noise that systematically induces overconfidence. To address the miscalibration in tool-use agents, we proposed the Calibration Agentic RL (CAR) framework, incorporating a novel Margin-Separated Calibration Reward (MSCR) that strictly separates incentives for

correct and incorrect predictions. Extensive experiments demonstrate that CAR significantly reduces calibration error while maintaining competitive task performance, with robust generalization from local simulation to noisy, real-world API environments. Our findings underscore the necessity of tool-specific calibration strategies and establish a foundation for building self-aware agents capable of reliably communicating uncertainty in high-stakes deployments.

Limitations

In this work, we studied the confidence dichotomy between evidence and verification tools in controlled agentic settings and proposed the Calibration Agentic RL framework to address miscalibration in evidence-tool scenarios. One key limitation is that our *full* RL sweeps remain concentrated on models with 3B to 7B parameters because agentic RL training is computationally expensive. We partially mitigate this by adding a 14B CAR evaluation and a 72B pilot study in the appendix, both of which support the scale-robustness of our findings, but comprehensive large-scale RL ablations remain beyond our current resources. Furthermore, our evaluation primarily focuses on short-answer question answering and mathematical reasoning, where correctness is well-defined. Calibration behavior in more open-ended generation scenarios, such as long-form report writing or multi-step autonomous planning, may involve underspecified correctness signals or more delayed feedback loops that our current framework does not address. We leave these explorations to future work.

Acknowledgments

This work was supported by JST CRONOS (Grant Number JPMJCS25K5), JST NEXUS (Grant Number JPMJNX25CA), and JSPS KAKENHI (Grant Number 26K21244). Weihao Xuan is supported by RIKEN Junior Research Associate (JRA) Program.

References

- Art of Problem Solving. 2025. Aime problems and solutions. https://artofproblemsolving.com/wiki/index.php/AIME_Problems_and_Solutions.
- Glenn W. Brier. 1950. Verification of forecasts expressed in terms of probability. *Monthly weather review*, 78(1):1–3.

- Yifei Chen, Guanting Dong, and Zhicheng Dou. 2026. [Et-agent: Incentivizing effective tool-integrated reasoning agent via behavior calibration](#). *Preprint*, arXiv:2601.06860.
- Zijian Chen, Xueguang Ma, Shengyao Zhuang, Ping Nie, Kai Zou, Andrew Liu, Joshua Green, Kshama Patel, Ruoxi Meng, Mingyi Su, and 1 others. 2025. [Browsecomp-plus: A more fair and transparent evaluation benchmark of deep-research agent](#). *arXiv preprint arXiv:2508.06600*.
- Mehul Damani, Isha Puri, Stewart Slocum, Idan Shenefeld, Leshem Choshen, Yoon Kim, and Jacob Andreas. 2025. [Beyond binary rewards: Training llms to reason about their uncertainty](#). *Preprint*, arXiv:2507.16806.
- Mustafa Omer Gul, Claire Cardie, and Tanya Goyal. 2026. [Mash: Modeling abstention via selective help-seeking](#). *Preprint*, arXiv:2510.01152.
- Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q. Weinberger. 2017. On calibration of modern neural networks. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70, ICML'17*, page 1321–1330. JMLR.org.
- Lukas Haas, Gal Yona, Giovanni D’Antonio, Sasha Goldshtein, and Dipanjan Das. 2025. [Simpleqa verified: A reliable factuality benchmark to measure parametric knowledge](#). *Preprint*, arXiv:2509.07968.
- Hongliang He, Wenlin Yao, Kaixin Ma, Wenhao Yu, Yong Dai, Hongming Zhang, Zhenzhong Lan, and Dong Yu. 2024. [Webvoyager: Building an end-to-end web agent with large multimodal models](#). *arXiv preprint arXiv:2401.13919*.
- Wenyue Hua, Xianjun Yang, Mingyu Jin, Zelong Li, Wei Cheng, Ruixiang Tang, and Yongfeng Zhang. 2024. [TrustAgent: Towards safe and trustworthy LLM-based agents](#). In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 10000–10016, Miami, Florida, USA. Association for Computational Linguistics.
- Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. 2024. [Swe-bench: Can language models resolve real-world github issues?](#) *Preprint*, arXiv:2310.06770.
- Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan Arik, Dong Wang, Hamed Zamani, and Jiawei Han. 2025. [Search-r1: Training llms to reason and leverage search engines with reinforcement learning](#). *Preprint*, arXiv:2503.09516.
- Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick SH Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense passage retrieval for open-domain question answering. In *EMNLP (1)*, pages 6769–6781.
- Polina Kirichenko, Mark Ibrahim, Kamalika Chaudhuri, and Samuel J. Bell. 2025. [Abstentionbench: Reasoning llms fail on unanswerable questions](#). *Preprint*, arXiv:2506.09038.
- Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, Kristina Toutanova, Llion Jones, Matthew Kelcey, Ming-Wei Chang, Andrew M. Dai, Jakob Uszkoreit, Quoc Le, and Slav Petrov. 2019. [Natural questions: A benchmark for question answering research](#). *Transactions of the Association for Computational Linguistics*, 7:452–466.
- Kuan Li, Zhongwang Zhang, Huifeng Yin, Rui Ye, Yida Zhao, Liwen Zhang, Litu Ou, Dingchu Zhang, Xixi Wu, Jialong Wu, Xinyu Wang, Zile Qiao, Zhen Zhang, Yong Jiang, Pengjun Xie, Fei Huang, and Jingren Zhou. 2025a. [Websailor-v2: Bridging the chasm to proprietary agents via synthetic data and scalable reinforcement learning](#). *Preprint*, arXiv:2509.13305.
- Kuan Li, Zhongwang Zhang, Huifeng Yin, Liwen Zhang, Litu Ou, Jialong Wu, Wenbiao Yin, Baixuan Li, Zhengwei Tao, Xinyu Wang, Weizhou Shen, Junkai Zhang, Dingchu Zhang, Xixi Wu, Yong Jiang, Ming Yan, Pengjun Xie, Fei Huang, and Jingren Zhou. 2025b. [Websailor: Navigating super-human reasoning for web agent](#). *Preprint*, arXiv:2507.02592.
- Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2023. [Let’s verify step by step](#). *Preprint*, arXiv:2305.20050.
- Stephanie Lin, Jacob Hilton, and Owain Evans. 2022. [Teaching models to express their uncertainty in words](#). *Preprint*, arXiv:2205.14334.
- Hao Liu, Zi-Yi Dou, Yixin Wang, Nanyun Peng, and Yisong Yue. 2024. [Uncertainty calibration for tool-using language agents](#). In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 16781–16805, Miami, Florida, USA. Association for Computational Linguistics.
- Michael Luo, Sijun Tan, Justin Wong, Xiaoxiang Shi, William Y Tang, Manan Roongta, Colin Cai, Jeffrey Luo, Tianjun Zhang, Li Erran Li, and 1 others. 2025. [Deepscaler: Surpassing o1-preview with a 1.5 b model by scaling rl](#). *Notion Blog*, 3(5).
- OpenAI. 2025. [GPT-5 mini Model | OpenAI API](#).
- Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, and 25 others. 2025. [Qwen2.5 technical report](#). *Preprint*, arXiv:2412.15115.

- Yijia Shao, Humishka Zope, Yucheng Jiang, Jiaxin Pei, David Nguyen, Erik Brynjolfsson, and Diyi Yang. 2025. [Future of work with ai agents: Auditing automation and augmentation potential across the u.s. workforce](#). *Preprint*, arXiv:2506.06576.
- Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. 2024. [Deepseekmath: Pushing the limits of mathematical reasoning in open language models](#). *Preprint*, arXiv:2402.03300.
- Yuanhao Shen, Xiaodan Zhu, and Lei Chen. 2024. [SMARTCAL: An approach to self-aware tool-use evaluation and calibration](#). In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pages 774–789, Miami, Florida, US. Association for Computational Linguistics.
- Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. 2024. [Hybridflow: A flexible and efficient rlhf framework](#). *arXiv preprint arXiv:2409.19256*.
- Yucheng Shi, Wenhao Yu, Jingyuan Huang, Wenlin Yao, Wenhao Chen, and Ninghao Liu. 2026. [Towards trustworthy gui agents: A survey](#). *Preprint*, arXiv:2503.23434.
- Linxin Song, Taiwei Shi, and Jieyu Zhao. 2025. [The hallucination tax of reinforcement finetuning](#). In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 2105–2120, Suzhou, China. Association for Computational Linguistics.
- STUDENT. 1908. [The probable error of a mean](#). *Biometrika*, 6(1):1–25.
- Nishant Subramani, Jason Eisner, Justin Svegliato, Benjamin Van Durme, Yu Su, and Sam Thomson. 2025. [MICE for CATs: Model-internal confidence estimation for calibrating agents with tools](#). In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 12362–12375, Albuquerque, New Mexico. Association for Computational Linguistics.
- Tongyi DeepResearch Team, Baixuan Li, Bo Zhang, Dingchu Zhang, Fei Huang, Guangyu Li, Guoxin Chen, Huifeng Yin, Jialong Wu, Jingren Zhou, Kuan Li, Liangcai Su, Litu Ou, Liwen Zhang, Pengjun Xie, Rui Ye, Wenbiao Yin, Xinmiao Yu, Xinyu Wang, and 38 others. 2025. [Tongyi deepresearch technical report](#). *Preprint*, arXiv:2510.24701.
- Katherine Tian, Eric Mitchell, Allan Zhou, Archit Sharma, Rafael Rafailov, Huaxiu Yao, Chelsea Finn, and Christopher Manning. 2023. [Just ask for calibration: Strategies for eliciting calibrated confidence scores from language models fine-tuned with human feedback](#). In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 5433–5442, Singapore. Association for Computational Linguistics.
- Trae Research Team, Pengfei Gao, Zhao Tian, Xiangxin Meng, Xincheng Wang, Ruida Hu, Yuanan Xiao, Yizhou Liu, Zhao Zhang, Junjie Chen, Cuiyun Gao, Yun Lin, Yingfei Xiong, Chao Peng, and Xia Liu. 2025. [Trae agent: An llm-based agent for software engineering with test-time scaling](#). *Preprint*, arXiv:2507.23370.
- Liang Wang, Nan Yang, Xiaolong Huang, Binxiang Jiao, Linjun Yang, Daxin Jiang, Rangan Majumder, and Furu Wei. 2024. [Text embeddings by weakly-supervised contrastive pre-training](#). *Preprint*, arXiv:2212.03533.
- Jason Wei, Nguyen Karina, Hyung Won Chung, Yunxin Joy Jiao, Spencer Papay, Amelia Glaese, John Schulman, and William Fedus. 2024. [Measuring short-form factuality in large language models](#). *Preprint*, arXiv:2411.04368.
- Jason Wei, Zhiqing Sun, Spencer Papay, Scott McKinney, Jeffrey Han, Isa Fulford, Hyung Won Chung, Alex Tachard Passos, William Fedus, and Amelia Glaese. 2025a. [Browsecomp: A simple yet challenging benchmark for browsing agents](#). *Preprint*, arXiv:2504.12516.
- Zhepei Wei, Wenlin Yao, Yao Liu, Weizhi Zhang, Qin Lu, Liang Qiu, Changlong Yu, Puyang Xu, Chao Zhang, Bing Yin, Hyokun Yun, and Lihong Li. 2025b. [Webagent-r1: Training web agents via end-to-end multi-turn reinforcement learning](#). *Preprint*, arXiv:2505.16421.
- Jialong Wu, Baixuan Li, Runnan Fang, Wenbiao Yin, Liwen Zhang, Zhengwei Tao, Dingchu Zhang, Zekun Xi, Gang Fu, Yong Jiang, Pengjun Xie, Fei Huang, and Jingren Zhou. 2025a. [Webdancer: Towards autonomous information seeking agency](#). *Preprint*, arXiv:2505.22648.
- Jialong Wu, Wenbiao Yin, Yong Jiang, Zhenglin Wang, Zekun Xi, Runnan Fang, Linhai Zhang, Yulan He, Deyu Zhou, Pengjun Xie, and Fei Huang. 2025b. [WebWalker: Benchmarking LLMs in web traversal](#). In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 10290–10305, Vienna, Austria. Association for Computational Linguistics.
- Miao Xiong, Zhiyuan Hu, Xinyang Lu, Yifei Li, Jie Fu, Junxian He, and Bryan Hooi. 2024. [Can llms express their uncertainty? an empirical evaluation of confidence elicitation in llms](#). *Preprint*, arXiv:2306.13063.
- Weihao Xuan, Qingcheng Zeng, Heli Qi, Junjue Wang, and Naoto Yokoya. 2025. [Seeing is believing, but how much? a comprehensive analysis of verbalized calibration in vision-language models](#). In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 1408–1450,

Suzhou, China. Association for Computational Linguistics.

Zhenghai Xue, Longtao Zheng, Qian Liu, Yingru Li, Xiaosen Zheng, Zejun Ma, and Bo An. 2025. [Simpletir: End-to-end reinforcement learning for multi-turn tool-integrated reasoning](#). *Preprint*, arXiv:2509.02479.

John Yang, Carlos E Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik R Narasimhan, and Ofir Press. 2024. [SWE-agent: Agent-computer interfaces enable automated software engineering](#). In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

John Yang, Kilian Lieret, Carlos E. Jimenez, Alexander Wettig, Kabir Khandpur, Yanzhe Zhang, Binyuan Hui, Ofir Press, Ludwig Schmidt, and Diyi Yang. 2025. [Swe-smith: Scaling data for software engineering agents](#). *Preprint*, arXiv:2504.21798.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. 2018. [HotpotQA: A dataset for diverse, explainable multi-hop question answering](#). In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2369–2380, Brussels, Belgium. Association for Computational Linguistics.

Dongkeun Yoon, Seungone Kim, Sohee Yang, Sunkyoung Kim, Soyeon Kim, Yongil Kim, Eunbi Choi, Yireun Kim, and Minjoon Seo. 2025. [Reasoning models better express their confidence](#). *Preprint*, arXiv:2505.14489.

Miao Yu, Fanci Meng, Xinyun Zhou, Shilong Wang, Junyuan Mao, Linsey Pang, Tianlong Chen, Kun Wang, Xinfeng Li, Yongfeng Zhang, Bo An, and Qingsong Wen. 2025. [A survey on trustworthy llm agents: Threats and countermeasures](#). *Preprint*, arXiv:2503.09648.

Qingcheng Zeng, Weihao Xuan, Leyang Cui, and Rob Voigt. 2025. [Thinking out loud: Do reasoning models know when they’re right?](#) In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 1394–1407, Suzhou, China. Association for Computational Linguistics.

Jiaxin Zhang, Caiming Xiong, and Chien-Sheng Wu. 2026. [Agentic confidence calibration](#). *Preprint*, arXiv:2601.15778.

Yanzhao Zhang, Mingxin Li, Dingkun Long, Xin Zhang, Huan Lin, Baosong Yang, Pengjun Xie, An Yang, Dayiheng Liu, Junyang Lin, and 1 others. 2025. [Qwen3 embedding: Advancing text embedding and reranking through foundation models](#). *arXiv preprint arXiv:2506.05176*.

Peilin Zhou, Bruce Leon, Xiang Ying, Can Zhang, Yifan Shao, Qichen Ye, Dading Chong, Zhiling Jin, Chenxuan Xie, Meng Cao, Yuxin Gu, Sixin Hong, Jing Ren, Jian Chen, Chao Liu, and Yining Hua. 2025.

[Browsecomp-zh: Benchmarking web browsing ability of large language models in chinese](#). *Preprint*, arXiv:2504.19314.

A More Evaluation Details

Evaluation Protocol. To ensure fair comparison across experimental configurations, we partition experiments by backbone model and test dataset, treating each (model, dataset) pair as an independent evaluation group. Within each group, a sample is considered *effective* only if both the final answer and a verbalized confidence score can be successfully extracted. For confidence normalization, we divide by 100 when the parsed value exceeds 1 (indicating percentage format), and clamp all values to $[0, 1]$. All metrics are then computed on the intersection of effective samples across all compared methods, ensuring that observed differences are not confounded by uneven sample coverage. For MCIP in Table 1, we further restrict to the intersection of incorrectly answered questions, as this metric specifically quantifies overconfidence on errors. For SimpleQA-verified (Haas et al., 2025), BrowseComp (Wei et al., 2025a), and WebWalkerQA (Wu et al., 2025b), we follow their official evaluation protocols and use GPT-5-mini (OpenAI, 2025) as the LLM judge to determine answer correctness.

Expected Calibration Error (ECE). Following Guo et al. (2017), we partition verbalized confidences into $M = 10$ equally-spaced bins and measure the weighted average gap between confidence and accuracy:

$$\text{ECE} = \sum_{m=1}^M \frac{|B_m|}{n} |\text{acc}(B_m) - \text{conf}(B_m)|, \quad (4)$$

where B_m denotes the set of samples whose verbalized confidence $q_i \in [0, 1]$ falls into bin m , n is the total number of samples, $\text{acc}(B_m) = \frac{1}{|B_m|} \sum_{i \in B_m} \mathbb{1}_{y_i^* = y_i}$ is the empirical accuracy within bin m (with y_i^* denoting the predicted answer and y_i the gold answer), and $\text{conf}(B_m) = \frac{1}{|B_m|} \sum_{i \in B_m} q_i$ is the average verbalized confidence within bin m .

Brier Score. The Brier score (Brier, 1950) measures the mean squared error between verbalized confidence and binary correctness:

$$\text{Brier} = \frac{1}{n} \sum_{i=1}^n (q_i - \mathbb{1}_{y_i^* = y_i})^2. \quad (5)$$

Lower Brier scores indicate better calibration.

AUROC. We use AUROC to assess whether verbalized confidence can reliably distinguish correct from incorrect predictions. Let $n_c = |\mathcal{D}_{\text{correct}}|$ and $n_w = |\mathcal{D}_{\text{wrong}}|$, AUROC is computed by:

$$\text{AUROC} = \frac{\sum_{i \in \mathcal{D}_{\text{correct}}} R_i - n_c(n_c + 1)/2}{n_c \cdot n_w}, \quad (6)$$

where $\mathcal{D}_{\text{correct}} = \{i \mid y_i^* = y_i\}$ and $\mathcal{D}_{\text{wrong}} = \{i \mid y_i^* \neq y_i\}$ denote the sets of correctly and incorrectly answered samples respectively, and R_i is the rank of sample i 's verbalized confidence among all n samples (with ties resolved by average ranking). AUROC equals the probability that a randomly chosen correct prediction has higher verbalized confidence than a randomly chosen incorrect one.

B Prompt Template for Training

We present the prompt templates used for CAR training. These templates are minimally adapted from the tool-call implementation of SearchR1 (Jin et al., 2025) in VeRL (Sheng et al., 2024) and SimpleTIR (Xue et al., 2025) frameworks, with the only modification being the addition of confidence elicitation instructions (i.e., requiring the model to output a confidence score between 0 and 100 via `<confidence>` tags or the `final_confidence()` function).

CAR Training on Web Search

Answer the given question. You must conduct reasoning inside `<think>` and `</think>` first every time you get new information. After reasoning, if you find you lack some knowledge, you can call a search engine by `<tool_call> query </tool_call>` and it will return the top searched results between `<tool_response>` and `</tool_response>`. You can search as many times as you want. If you find no further external knowledge needed, you can directly provide the answer inside `<answer>` and `</answer>` without detailed illustrations, and another confidence score ranging from 0 to 100 indicating how confident your answer is correct within `<confidence>` and `</confidence>`. For example, `<answer> Beijing </answer>` `<confidence> 94 </confidence>`. Question: {question}

CAR Training on Tool-Integrated Reasoning

Solve the following problem step by step. You now have the ability to selectively write executable Python code to enhance your reasoning process. The Python code will be executed by an external sandbox, and the output (after "Code execution result: ") is returned to aid your reasoning and help you arrive at the final answer. The Python code should be complete scripts, including necessary imports.

Code Format: Each code snippet is wrapped between `````. You need to use `'print()'` to output intermediate results.

Answer and Confidence Format: You can use the `'final_answer()'` function in the code to return your final answer, and `'final_confidence()'` to indicate your confidence score (0-100). For example, to answer the User Question: What is the result of the $5 + 3 + 1294.678$?, you can write:

```
```py
answer = 5 + 3 + 1294.678
final_answer(answer)
final_confidence(94)
```
```

You can also use `\boxed` to return your answer and `<confidence>` tags for confidence score. The last part of your response should be: `\boxed{'The final answer goes here.'}` `<confidence> 85 </confidence>`

User Question: {question}

C Configurations of the Pilot Study

We detail the experimental configurations for the three settings evaluated in Table 1.

Direct Prompting. In this setting, the model answers questions using only its internal parametric knowledge without invoking any external tools. We prompt the model to provide both an answer and a verbalized confidence score (0–100) in a single turn. The prompt templates for the web search and code interpreter domains are provided below.

Prompting-based Tool-Use. In this setting, we use the same instruction-tuned model without any RL fine-tuning. The model is prompted with tool-

use templates (Appendix B), which instruct it to interleave reasoning with tool calls and report both answers and confidence scores.

RL-based Tool-Use. For this setting, we use models fine-tuned via RL on their respective tool-use tasks: Search-R1 (Jin et al., 2025) for web search and SimpleTIR (Xue et al., 2025) for code interpreter. Importantly, these models are trained with their *original* reward functions, which incentivize only task accuracy without any calibration-aware components. During evaluation, we use the same prompt templates as the prompting-based setting to elicit confidence scores.

Retrieval Configuration. For web search experiments in Table 2, we use the 2018 Wikipedia dump (Karpukhin et al., 2020) as the knowledge source with E5 (Wang et al., 2024) as the dense retriever, following the Search-R1 setup.

Rollout Configuration. All rollouts are conducted with a maximum of 5 interaction turns (i.e., up to 5 tool calls per question).

Inference Hyperparameters. For web search experiments, we use greedy decoding with temperature $T = 0$.

For tool-integrated reasoning experiments (AIME2024, AIME2025, MATH-500), we use sampling with temperature $T = 1.0$ and $top_p = 0.7$, generating $n = 32$ samples per question. We process each of the 32 samples independently: for each sample, we extract its predicted answer and verbalized confidence score, filtering out samples that fail to produce a valid confidence in $[0, 1]$. Both accuracy and calibration metrics (ECE, Brier, AUROC) are computed separately for each sample index across all questions, and we report the mean across all 32 sample indices as Avg@32.

Prompts.

Direct Prompting on Web Search

Answer the given question. You must conduct reasoning inside `<think>` and `</think>` first. After reasoning, directly provide the answer inside `<answer>` and `</answer>` without detailed illustrations, and another confidence score ranging from 0 to 100 indicating how confident your answer is correct within `<confidence>` and `</confidence>`. For example, `<answer> Beijing </answer>`

```
<confidence> 94 </confidence>. Question: {question}
```

Direct Prompting on Tool-Integrated Reasoning

Please reason step by step using comments and output your answer with ‘final_answer(answer)’ and your confidence score with ‘final_confidence(score)’ within ````py` and ````<end_code>`.

The confidence score should be an integer from 0 to 100, indicating how confident you are in your answer.

For example:

Task: What is the result of the following operation: $5 + 3 + 1294.678$?

Assistant:

```
```py
Compute the result of the operation.
answer = 1302.678
Return the final answer.
final_answer(answer)
Indicate confidence level (0-100).
final_confidence(94)
```<end_code>
Now start:
Task: {question}
```

D Extended Pilot Study with Larger Models

To examine whether the confidence dichotomy persists at larger model scales, we extend our pilot study to Qwen2.5-72B-Instruct. Due to computational constraints, we evaluate only the direct prompting and prompting-based tool-use configurations without RL fine-tuning. Following the methodology in section 3, we compute accuracy on the intersection of samples where both configurations successfully output confidence scores, and measure MCIP on the intersection of incorrectly answered questions.

As shown in Table 5, the confidence dichotomy observed in section 3 generalizes to the 72B parameter scale. Consistent with our findings on smaller models, integrating evidence tools (e.g., web search) elevates MCIP, while integrating verification tools (e.g., code interpreters) reduces it. These results suggest that the confidence dichotomy is a fundamental property of tool-use dy-

Domain	Dataset	Setting	Acc. \uparrow	MCIP \downarrow
Search	NQ	Direct prompting	30.0	0.908
		Prompting-based tool-use	37.2	0.914
	HotpotQA	Direct prompting	30.6	0.846
		Prompting-based tool-use	43.1	0.863
Code	AIME2024	Direct prompting	12.0	0.943
		Prompting-based tool-use	12.0	0.918
	AIME2025	Direct prompting	12.0	0.956
		Prompting-based tool-use	8.0	0.910
	MATH-500	Direct prompting	73.6	0.962
		Prompting-based tool-use	64.9	0.944

Table 5: Extended pilot study results across tool-use configurations and domains using *Qwen2.5-72B-Instruct* as the backbone.

namics rather than a scale-dependent phenomenon.

E Reward Separation Guarantees for GRPO

In this appendix, we provide a theoretical justification for the proposed Margin-Separated Calibration Reward (MSCR). While standard scoring rules (e.g., Brier score) incentivize proper calibration in expectation, their direct application in GRPO can lead to optimization instability due to reward overlap between correct and incorrect trajectories.

We formally prove that by enforcing a strict separation margin ($a > b$) between the reward lower bound of correct responses and the upper bound of incorrect ones, MSCR guarantees two critical properties for GRPO:

- 1. Strict Advantage Ordering:** Any correct trajectory yields a strictly higher advantage value than any incorrect trajectory within the same group, regardless of verbalized confidence. This effectively prevents the “reward hacking” scenario where high-confidence hallucinations outweigh low-confidence correct answers.
- 2. Non-Collapse of Variance:** The group standard deviation admits a strictly positive lower bound in mixed-outcome batches, ensuring numerical stability during advantage normalization.

Setup. Fix a prompt x and sample a GRPO group of N trajectories $\{\tau_i\}_{i=1}^N$. Each trajectory yields a final answer \hat{y}_i and a verbalized confidence $q_i \in [0, 1]$. Let $c_i \triangleq \mathbb{I}[\hat{y}_i = y] \in \{0, 1\}$ denote correctness and let r_i be its scalar reward. GRPO

computes the group mean and standard deviation

$$\mu \triangleq \frac{1}{N} \sum_{i=1}^N r_i, \quad \sigma \triangleq \sqrt{\frac{1}{N} \sum_{i=1}^N (r_i - \mu)^2},$$

and uses the normalized advantage

$$A_i \triangleq \frac{r_i - \mu}{\sigma + \varepsilon},$$

where $\varepsilon > 0$ is a small constant for numerical stability.

Let $\mathcal{C} \triangleq \{i : c_i = 1\}$ and $\mathcal{W} \triangleq \{i : c_i = 0\}$ be the sets of correct and incorrect trajectories in the group, with sizes $k \triangleq |\mathcal{C}|$ and $m \triangleq |\mathcal{W}|$ (so $N = k + m$).

Definition 1 (Margin-separated reward). A reward function $R(c, q)$ is (a, b) -separated if there exist constants $a > b$ such that for all $q \in [0, 1]$,

$$R(1, q) \geq a \quad \text{and} \quad R(0, q) \leq b.$$

Lemma 1 (GRPO stability under margin-separated rewards). Consider any GRPO group with $k \geq 1$ correct and $m \geq 1$ incorrect trajectories, and suppose the realized rewards satisfy (a, b) -separation in Definition 1. Then the following properties hold.

(1) Strict advantage ordering (correct dominates incorrect). For any $i \in \mathcal{C}$ and $j \in \mathcal{W}$, we have $r_i > r_j$ and thus

$$A_i > A_j.$$

Equivalently, $\min_{i \in \mathcal{C}} A_i > \max_{j \in \mathcal{W}} A_j$.

(2) Mean-advantage separation. Let $\bar{r}_{\mathcal{C}} \triangleq \frac{1}{k} \sum_{i \in \mathcal{C}} r_i$ and $\bar{r}_{\mathcal{W}} \triangleq \frac{1}{m} \sum_{j \in \mathcal{W}} r_j$. Then $\bar{r}_{\mathcal{C}} > \mu > \bar{r}_{\mathcal{W}}$ and

$$\frac{1}{k} \sum_{i \in \mathcal{C}} (r_i - \mu) = \bar{r}_{\mathcal{C}} - \mu = \frac{m}{N} (\bar{r}_{\mathcal{C}} - \bar{r}_{\mathcal{W}}) > 0,$$

$$\frac{1}{m} \sum_{j \in \mathcal{W}} (r_j - \mu) = \bar{r}_{\mathcal{W}} - \mu = -\frac{k}{N} (\bar{r}_{\mathcal{C}} - \bar{r}_{\mathcal{W}}) < 0.$$

Consequently, the *average* GRPO advantage is positive on correct trajectories and negative on incorrect ones.

(3) Non-collapse of group normalization. The group standard deviation admits the lower bound

$$\sigma \geq \frac{\sqrt{km}}{N} (a - b) > 0.$$

Hence, in any mixed-outcome group ($k, m > 0$), the normalization denominator $\sigma + \varepsilon$ is bounded away from 0 (up to ε), preventing reward-variance collapse.

Proof. By (a, b) -separation, for all $i \in \mathcal{C}$ we have $r_i \geq a$, and for all $j \in \mathcal{W}$ we have $r_j \leq b$, with $a > b$. Thus $r_i > r_j$ for any $i \in \mathcal{C}, j \in \mathcal{W}$. Since $A_i = (r_i - \mu)/(\sigma + \varepsilon)$ is an increasing affine transform of r_i when $\sigma + \varepsilon > 0$, this implies $A_i > A_j$, proving (1).

For (2), note $\mu = \frac{k\bar{r}_{\mathcal{C}} + m\bar{r}_{\mathcal{W}}}{N}$ and $\bar{r}_{\mathcal{C}} > \bar{r}_{\mathcal{W}}$, so μ lies strictly between the two means. Moreover,

$$\bar{r}_{\mathcal{C}} - \mu = \bar{r}_{\mathcal{C}} - \frac{k\bar{r}_{\mathcal{C}} + m\bar{r}_{\mathcal{W}}}{N} = \frac{m}{N}(\bar{r}_{\mathcal{C}} - \bar{r}_{\mathcal{W}}) > 0,$$

and similarly $\bar{r}_{\mathcal{W}} - \mu = -\frac{k}{N}(\bar{r}_{\mathcal{C}} - \bar{r}_{\mathcal{W}}) < 0$.

For (3), decompose the total variance into within-set and between-set components (one-way ANOVA):

$$\sigma^2 = \frac{k}{N} \text{Var}_{\mathcal{C}}(r) + \frac{m}{N} \text{Var}_{\mathcal{W}}(r) + \frac{km}{N^2} (\bar{r}_{\mathcal{C}} - \bar{r}_{\mathcal{W}})^2,$$

where $\text{Var}_{\mathcal{C}}(r) \triangleq \frac{1}{k} \sum_{i \in \mathcal{C}} (r_i - \bar{r}_{\mathcal{C}})^2$ (and analogously for \mathcal{W}). Dropping the nonnegative within-set terms yields

$$\sigma^2 \geq \frac{km}{N^2} (\bar{r}_{\mathcal{C}} - \bar{r}_{\mathcal{W}})^2.$$

Finally, $\bar{r}_{\mathcal{C}} \geq a$ and $\bar{r}_{\mathcal{W}} \leq b$ imply $\bar{r}_{\mathcal{C}} - \bar{r}_{\mathcal{W}} \geq a - b$, hence $\sigma \geq \frac{\sqrt{km}}{N} (a - b)$. \square

Corollary 1 (Instantiation for our outcome rewards). Let $R(c, q)$ be the calibration-motivated outcome reward.

Weighted Brier (Eq. (1)). For $R_{\text{WB}}(c, q) = c - \lambda(q - c)^2$ with $\lambda \in (0, 1]$, we have $R_{\text{WB}}(1, q) \in [1 - \lambda, 1]$ and $R_{\text{WB}}(0, q) \in [-\lambda, 0]$. Thus $(a, b) = (1 - \lambda, 0)$ and the separation margin is $a - b = 1 - \lambda$. Lemma 1 applies iff $\lambda < 1$; as $\lambda \rightarrow 1$, the margin (and the lower bound on σ) vanishes.

MSCR (Eq. (2)). For $R_{\text{MSCR}}(c, q) = c[1 + \beta_1(1 - (1 - q)^2)] - (1 - c)\beta_2 q^2$ with $\beta_1, \beta_2 > 0$, we have $R_{\text{MSCR}}(1, q) \geq 1$ and $R_{\text{MSCR}}(0, q) \leq 0$ for all q . Thus $(a, b) = (1, 0)$ and the separation margin is $a - b = 1$, independent of (β_1, β_2) .

Remark. Margin separation implies *strict ordering* of GRPO advantages (all correct advantages exceed all incorrect advantages), and yields a non-trivial lower bound on the group standard deviation in mixed-outcome groups. It does *not* require every incorrect trajectory to have negative advantage; rather, it guarantees that incorrect trajectories are strictly dominated by correct ones under GRPO normalization, while still allowing calibration-driven

preference among incorrect trajectories (e.g., preferring lower-confidence errors over confident hallucinations).

F Results for BrowseComp and WebWalkerQA

Beyond API-based evaluation, we further assess generalization under *corpus shift*: while our agents are trained with the 2018 Wikipedia dump, we evaluate on BrowseComp (Wei et al., 2025a) and WebWalkerQA (Wu et al., 2025b) using the BrowseComp-plus corpus (Chen et al., 2025) with Qwen3-Embedding-8B (Zhang et al., 2025) as the dense retriever.

Model	Method	BrowseComp			
		Acc \uparrow	ECE \downarrow	Brier \downarrow	AUROC \uparrow
Qwen2.5-3B	Vanilla S-R1	0.5	0.823	0.706	0.578
	CAR (MSCR)	6.5	0.071	0.070	0.748
Qwen2.5-7B	Vanilla S-R1	6.5	0.833	0.767	0.700
	CAR (MSCR)	10.8	0.094	0.097	0.713

Table 6: Calibration results on BrowseComp under corpus shift.

Model	Method	WebWalkerQA			
		Acc \uparrow	ECE \downarrow	Brier \downarrow	AUROC \uparrow
Qwen2.5-3B	Vanilla S-R1	2.3	0.868	0.808	0.345
	CAR (MSCR)	3.0	0.033	0.033	0.498
Qwen2.5-7B	Vanilla S-R1	1.7	0.619	0.444	0.494
	CAR (MSCR)	2.1	0.020	0.019	0.539

Table 7: Calibration results on WebWalkerQA under corpus shift.

Results in Table 6 and 7 show that CAR maintains robust calibration improvements despite the distribution shift.

G SimpleTIR Training Configurations

Our base training configuration follows the SimpleTIR framework (Xue et al., 2025). To enable calibration-aware training, we extend the reward function to incorporate confidence estimates from verbalization. Specifically, the model is prompted to output confidence scores ranging from 0 to 100 alongside its final answers, using either the `final_confidence()` function or `<confidence>` XML tags. The `final_confidence()` function is injected into the code execution environment to allow seamless confidence reporting within the agent’s reasoning trace.

All experiments are conducted on NVIDIA H200 GPUs. We train on a combination of the

SimpleTIR-Math and DeepScaler datasets (Luo et al., 2025), employing Qwen2.5-3B-Instruct as the backbone model.

H Additional Verification of MSCR

The main text already compares six distinct strategies spanning post-hoc rescaling, abstention-based regularization, and calibration-aware reward shaping. Since the weighted Brier score with $\lambda=1$ exhibits severe reward hacking, we further test whether MSCR depends on a brittle choice of calibration coefficients. Specifically, we vary β_1 and β_2 on *Qwen2.5-3B* across in-distribution (NQ, HotpotQA) and out-of-distribution (SimpleQA-verified) search benchmarks.

Configuration	NQ		HotpotQA		SimpleQA-verified	
	Acc \uparrow	ECE \downarrow	Acc \uparrow	ECE \downarrow	Acc \uparrow	ECE \downarrow
Vanilla Search-R1	43.1	0.528	27.4	0.699	34.8	0.610
Brier $\lambda=1$	22.3	0.091	24.3	0.148	21.2	0.027
Brier $\lambda=1/3$	44.5	0.307	28.5	0.329	35.8	0.203
MSCR (0.5, 0.5) main	45.5	0.303	29.2	0.286	36.6	0.192
MSCR (0.5, 1.0)	44.2	0.301	36.5	0.297	36.5	0.158
MSCR (1.0, 0.5)	44.5	0.292	33.7	0.250	36.5	0.227
MSCR (0.75, 0.25)	45.2	0.334	32.3	0.296	36.0	0.264
MSCR (0.25, 0.75)	44.0	0.299	38.5	0.309	36.6	0.183

Table 8: Additional verification of MSCR on search agents via coefficient ablations on *Qwen2.5-3B*. Tuples denote (β_1, β_2) .

As shown in Table 8, all tested MSCR variants substantially outperform the vanilla baseline in calibration while maintaining competitive accuracy across both ID and OOD settings, and no configuration exhibits the collapse seen with Brier $\lambda=1$. The best trade-off differs by benchmark, which is expected because β_1 controls the calibration bonus within the correct region, whereas β_2 controls the penalty for overconfident errors in the incorrect region. Crucially, the strict reward separation proved in Appendix E remains unchanged for any $\beta_1, \beta_2 > 0$, so varying these coefficients changes the within-region calibration incentive without reintroducing reward overlap. This explains why the different settings trace different accuracy-ECE trade-offs while remaining stable.

I Scaling Verification of CAR on Qwen2.5-14B

To further assess whether the RL-based benefits of CAR extend beyond the 3B to 7B regime used in the main experiments, we additionally train *Qwen2.5-14B-Instruct* in the search setting. We follow the same training configuration as in the

main experiments, using batch size 512 and $n=5$ rollouts, and train for 100 optimization steps under time constraints.

The results in Table 9 and Table 10 show that CAR continues to improve calibration at larger scales. Under the local retriever, CAR improves both accuracy and calibration on all three datasets, including an ECE reduction from 0.268 to 0.158 on SimpleQA-verified. Under the noisy Serper API setting, CAR likewise improves both accuracy and calibration, reducing ECE from 0.107 to 0.070 while increasing AUROC from 0.732 to 0.803. Notably, these gains already emerge after only 100 optimization steps, suggesting that the benefit of MSCR is not confined to small models or to heavily trained policies. Together with the 72B pilot study in Appendix D, these results support the view that both the confidence dichotomy and the effectiveness of CAR are not scale-dependent artifacts.

J Expanded TIR Evaluation

Our web search experiments serve as the primary testbed for architectural ablations because evidence tools expose the most severe overconfidence and reward overlap, making them the most informative setting for stress-testing calibration rewards. The TIR setting instead tests whether the best CAR configuration transfers to verification tools. To further strengthen this transfer analysis, we additionally evaluate vanilla SimpleTIR and CAR (MSCR) on two new mathematical reasoning benchmarks: AIME2026 and OlympiadBench.

As shown in Table 11, CAR improves all four metrics on both benchmarks. On AIME2026, it raises Acc@32 from 11.2 to 18.2 while reducing ECE by 27% (0.756 to 0.550). On OlympiadBench, it improves accuracy from 45.6 to 47.1 and reduces ECE by 25% (0.437 to 0.327). The AUROC gains are also substantial on both datasets, indicating stronger separation between correct and incorrect trajectories. Since OlympiadBench spans a broader range of mathematical problem types beyond competition-style questions, these results further support the robustness of CAR in verification-tool settings.

K Generative AI Statement

This work utilized generative AI tools to assist with formatting, generating LaTeX templates, and refining word choice. The authors reviewed and verified all AI-assisted content to ensure factual accuracy

Model	Method	NQ (ID)				HotpotQA (ID)				SimpleQA-verified (OOD)			
		Acc \uparrow	ECE \downarrow	Brier \downarrow	AUROC \uparrow	Acc \uparrow	ECE \downarrow	Brier \downarrow	AUROC \uparrow	Acc \uparrow	ECE \downarrow	Brier \downarrow	AUROC \uparrow
Qwen2.5-14B	Vanilla Search-R1	75.4	0.197	0.254	0.562	65.8	0.228	0.287	0.532	62.1	0.268	0.259	0.834
	CAR (MSCR)	81.1	0.151	0.219	0.669	68.4	0.210	0.284	0.661	65.1	0.158	0.156	0.902

Table 9: Search results on *Qwen2.5-14B-Instruct* with the local retriever.

Model	Method	SimpleQA-verified (Serper API)			
		Acc \uparrow	ECE \downarrow	Brier \downarrow	AUROC \uparrow
Qwen2.5-14B	Vanilla Search-R1	82.0	0.107	0.150	0.732
	CAR (MSCR)	83.6	0.070	0.121	0.803

Table 10: Search results on *Qwen2.5-14B-Instruct* under the Serper API retriever.

Model	Method	Benchmark	Acc@32 \uparrow	ECE \downarrow	Brier \downarrow	AUROC \uparrow
Qwen2.5-3B	Vanilla SimpleTIR	AIME2026	11.2	0.756	0.676	0.499
		OlympiadBench	45.6	0.437	0.435	0.514
	CAR (MSCR)	AIME2026	18.2	0.550	0.432	0.749
		OlympiadBench	47.1	0.327	0.351	0.658

Table 11: Expanded mathematical reasoning evaluation of TIR agents on two additional benchmarks. Accuracy is reported as Avg@32, following the same protocol as our main TIR experiments.

and academic integrity.

L License Statement

We primarily use the Qwen series of models. Specifically, *Qwen2.5-3B* is released under the Qwen Research License, while *Qwen2.5-7B* and *Qwen3-4B* are released under the Apache License 2.0. We use these models solely for research purposes and confirm that our usage complies with their respective license terms.

For datasets, we use NQ (Apache License 2.0), HotpotQA (CC BY-SA 4.0), AIME (Apache License 2.0), MATH-500 (MIT License), SimpleQA-verified (MIT License), BrowseComp (MIT License), and WebWalkerQA (Apache License 2.0). We use these datasets in accordance with their licenses, including any applicable attribution and share-alike requirements.

For codebases, we primarily use and adapt *Search-R1*, *VeRL*, and *SimpleTIR*, each licensed under the Apache License 2.0. Our modifications are used for research purposes and are documented in our repository. Where required by the Apache License 2.0, we retain the original copyright notices, include the license text, and clearly mark any significant changes. We also preserve any accompanying NOTICE files and provide appropriate attribution to the original authors and repositories.

M Qualitative Analysis

We present a qualitative analysis of representative interaction trajectories to substantiate the feedback asymmetry discussed in section 6. In the Evidence Tool setting (Appendix M.1), we highlight how stochastic retrieval induces epistemic illusions: despite retrieving documents containing conflicting information (e.g., differing dates for a renaming event), the agent is misled by surface-level lexical overlap, resulting in unwarranted high confidence. In the Verification Tool setting (Appendix M.2), we demonstrate how explicit execution feedback serves as a grounding mechanism that naturally tempers overconfidence. Finally, we illustrate how our MSCR framework mitigates miscalibration across these domains, enabling the agent to express appropriate uncertainty in the face of ambiguity.

M.1 Evidence Tool

NQ Problem 117 — Problem Statement

Question: when did the name of bombay change to mumbai?

Ground Truth: 1995

NQ Problem 117 — Direct Prompting

Answer: May 1, 1996 **Confidence:** 80%
Correct: No **Runtime Error:** None

Reasoning:

The name of Bombay was officially changed to Mumbai on May 1, 1996. This change was part of a movement to recognize the local Marathi heritage and to adopt the more historically significant name that predated the English name. Mumbai is the name used in Marathi, Hindi, and other Indian languages to refer to the city.

Final Prediction:

<answer>May 1, 1996</answer>
 <confidence>80</confidence>

NQ Problem 117 — Tool-use Prompting

Answer: March 6, 1996 **Confidence:** 95% **Correct:** No **Runtime Error:** None

Tool Call:

```
{"name": "search", "arguments": {"query_list": ["when did the name of bombay change to mumbai"]}}
```

Tool Response:

Doc 1 (Title: Mumbai)

island has been, from remote antiquity, Bomba, or Mamba Dévi, and that she still..., possesses a temple". By the late 20th century, the city was referred to as "Mumbai" or "Mambai" in Marathi, Konkani, Gujarati, Kannada and Sindhi, and as "Bambai" in Hindi. The Government of India officially changed the English name to "Mumbai" in November 1995. This came at the insistence of the Marathi nationalist Shiv Sena party, which had just won the Maharashtra state elections, and mirrored similar name changes across the country and particularly in Maharashtra. According to "Slate" magazine, "they argued that 'Bombay' was a corrupted

Doc 2 (Title: "History of Mumbai")

the city was incorporated into the dominion of the newly created Maharashtra State from Bombay State. The city's secular fabric was torn apart in the communal riots of 1992–93, while the bombings of 1993 caused extensive loss of life and property. The city was renamed Mumbai on 6 March 1996. Geologists believe that the coast of western India came into being around 100 to 80 mya, after it broke away from Madagascar. Soon after its detachment, the peninsular region of the Indian plate drifted over the Réunion hotspot, a volcanic hotspot in the Earth's lithosphere near the island of Réunion.

Doc 3 (Title: Mumbai)

About 850,000 people fled Bombay and the textile industry was adversely affected. As the capital of the Bombay Presidency, the city witnessed the Indian independence movement, with the Quit India Movement in 1942 and The Royal Indian Navy Mutiny in 1946 being its most notable events. After

India's independence in 1947, the territory of the Bombay Presidency retained by India was restructured into Bombay State. The area of Bombay State increased, after several erstwhile princely states that joined the Indian union were integrated into the state. Subsequently, the city became the capital of Bombay State. In April 1950, Municipal limits

Reasoning:

Based on the information provided, the name of Bombay officially changed to Mumbai in 1996. The text mentions that the city was renamed Mumbai on 6 March 1996.

Final Prediction:

```
<answer> March 6, 1996 </answer>
<confidence> 95 </confidence>
```

NQ Problem 117 — Vanilla RL (Search-R1)

Answer: March 6, 1996 **Confidence:** 100% **Correct:** No **Runtime Error:** None

Tool Call:

```
{"name": "search", "arguments": {"query_list": ["when did the name of bombay change to mumbai"]}}
```

Tool Response:

Doc 1 (Title: Mumbai)

island has been, from remote antiquity, Bomba, or Mamba Dévi, and that she still..., possesses a temple". By the late 20th century, the city was referred to as "Mumbai" or "Mambai" in Marathi, Konkani, Gujarati, Kannada and Sindhi, and as "Bambai" in Hindi. The Government of India officially changed the English name to "Mumbai" in November 1995. This came at the insistence of the Marathi nationalist Shiv Sena party, which had just won the Maharashtra state elections, and mirrored similar name changes across the country and particularly in Maharashtra. According to "Slate" magazine, "they argued that 'Bombay' was a corrupted

Doc 2 (Title: "History of Mumbai")

the city was incorporated into the dominion

of the newly created Maharashtra State from Bombay State. The city's secular fabric was torn apart in the communal riots of 1992–93, while the bombings of 1993 caused extensive loss of life and property. The city was renamed Mumbai on 6 March 1996. Geologists believe that the coast of western India came into being around 100 to 80 mya, after it broke away from Madagascar. Soon after its detachment, the peninsular region of the Indian plate drifted over the Réunion hotspot, a volcanic hotspot in the Earth's lithosphere near the island of Réunion.

Doc 3 (Title: Mumbai)

About 850,000 people fled Bombay and the textile industry was adversely affected. As the capital of the Bombay Presidency, the city witnessed the Indian independence movement, with the Quit India Movement in 1942 and The Royal Indian Navy Mutiny in 1946 being its most notable events. After India's independence in 1947, the territory of the Bombay Presidency retained by India was restructured into Bombay State. The area of Bombay State increased, after several erstwhile princely states that joined the Indian union were integrated into the state. Subsequently, the city became the capital of Bombay State. In April 1950, Municipal limits

Reasoning:

The information provided indicates that the city changed the name on 6 March 1996.

Final Prediction:

<answer> 6 March 1996 </answer>
<confidence> 100 </confidence>

NQ Problem 117 — MSCR (Ours)

Answer: March 6, 1996 **Confidence:** 0% **Correct:** No **Score:** 0.0

Interaction History:

Tool Call:

```
{"name": "search", "arguments": {"query_list": ["when did the name of bombay change to mumbai"]}}
```

Tool Response:

Doc 1 (Title: Mumbai)

island has been, from remote antiquity, Bomba, or Mamba Dévi, and that she still... possesses a temple". By the late 20th century, the city was referred to as "Mumbai" or "Mumbai" in Marathi, Konkani, Gujarati, Kannada and Sindhi, and as "Bombai" in Hindi. The Government of India officially changed the English name to "Mumbai" in November 1995. This came at the insistence of the Marathi nationalist Shiv Sena party, which had just won the Maharashtra state elections, and mirrored similar name changes across the country and particularly in Maharashtra. According to "Slate" magazine, "they argued that 'Bombay' was a corrupted

Doc 2 (Title: "History of Mumbai")

the city was incorporated into the dominion of the newly created Maharashtra State from Bombay State. The city's secular fabric was torn apart in the communal riots of 1992–93, while the bombings of 1993 caused extensive loss of life and property. The city was renamed Mumbai on 6 March 1996. Geologists believe that the coast of western India came into being around 100 to 80 mya, after it broke away from Madagascar. Soon after its detachment, the peninsular region of the Indian plate drifted over the Réunion hotspot, a volcanic hotspot in the Earth's lithosphere near the island of Réunion.

Doc 3 (Title: Mumbai)

About 850,000 people fled Bombay and the textile industry was adversely affected. As the capital of the Bombay Presidency, the city witnessed the Indian independence movement, with the Quit India Movement in 1942 and The Royal Indian Navy Mutiny in 1946 being its most notable events. After India's independence in 1947, the territory of the Bombay Presidency retained by India was restructured into Bombay State. The area of Bombay State increased, after several erstwhile princely states that joined the Indian union were integrated into the state. Subsequently, the city became the capital of Bombay State. In April 1950, Municipal limits

Reasoning:

The information provided mentions that the Government of India officially changed the English name to "Mumbai" in November 1995. However, it does not specify when the name change from "Bombay" to "Mumbai" occurred in the local languages or when the city was officially renamed to Mumbai. Given the search results, we cannot determine the exact date of the name change in local languages or the official renaming of the city to Mumbai.

Let's search further to find more specific information.

Tool Call:

```
{ "name": "search", "arguments":
{"query_list": ["when did bombay
change to mumbai in local
languages"]} }
```

Tool Response:

Doc 1 (Title: Mumbai)

island has been, from remote antiquity, Bomba, or Mamba Dévi, and that she still... possesses a temple". By the late 20th century, the city was referred to as "Mumbai" or "Mumbai" in Marathi, Konkani, Gujarati, Kannada and Sindhi, and as "Bombai" in Hindi. The Government of India officially changed the English name to "Mumbai" in November 1995. This came at the insistence of the Marathi nationalist Shiv Sena party, which had just won the Maharashtra state elections, and mirrored similar name changes across the country and particularly in Maharashtra. According to "Slate" magazine, "they argued that 'Bombay' was a corrupted

Doc 2 (Title: "Renaming of cities in India")

Similarly, Bombay, which was always called 'Mumbai' in Marathi, was restored to Mumbai in 1996. The renaming of cities is often specifically from English to Indian English in connection with that dialect's internal reforms. In other words, the city itself is not actually renamed in the local language, and the local name (or endonym) in the indigenous languages of India does not change, but the official spelling in Indian English is amended. An example is

the change from English "Calcutta" to English "Kolkata" – the local Bengali name ("Kôlkata") did not change. Such changes in English spelling may be

Doc 3 (Title: "History of Mumbai")

Bombay on 28 February 1948, ending the 282-year-long period of the British in Bombay. After the Partition of India on 15 August 1947, over 100,000 Sindhi refugees from the newly created Pakistan were relocated in the military camps five kilometres from Kalyan in the Maharashtra Region. It was converted into a township in 1949, and named Ulhasnagar by the then Governor-General of India, C. Rajagopalachari. In April 1950, Greater Bombay District came into existence with the merger of Bombay Suburbs and Bombay City. It spanned an area of and inhabited 2,339,000 of people in 1951. The Municipal Corporation limits

Reasoning:

The search results indicate that the official change of the English name to "Mumbai" was in November 1995. However, the name change in local languages such as Marathi, Konkani, Gujarati, Kannada, and Sindhi to "Mumbai" or "Mumbai" occurred later, specifically in 1996. The official renaming of the city to Mumbai in the local languages did not happen until March 6, 1996.

Final Prediction:

<answer> March 6, 1996 </answer>
<confidence>0</confidence>

M.2 Verification Tool

AIME2025 Problem 2 — Problem Statement

The 9 members of a baseball team went to an ice-cream parlor after their game. Each player had a single scoop cone of chocolate, vanilla, or strawberry ice cream. At least one player chose each flavor, and the number of players who chose chocolate was greater than the number of players who chose vanilla, which was greater than the number of players who chose strawberry. Let N be the number of different assignments of flavors to players that meet these

conditions. Find the remainder when N is divided by 1000.

Ground Truth: 16

AIME2025 Problem 2 — Direct Prompting

Answer: 772 **Confidence:** 100% **Correct:** No **Runtime Error:** None

Reasoning:

To solve the problem, let's break it down step-by-step.

1. Understanding the Constraints:

- There are 9 players.
- Each player can choose one of 3 flavors: chocolate (C), vanilla (V), or strawberry (S).
- The number of players who chose chocolate is greater than the number of players who chose vanilla, which is greater than the number of players who chose strawberry.
- This means we need to find combinations of players that satisfy these inequalities.

2. Possible Combinations of Players Choosing Each Flavor:

- Let c be the number of players choosing chocolate.
- Let v be the number of players choosing vanilla.
- Let s be the number of players choosing strawberry.
- Given $c > v > s$ and $c + v + s = 9$, we need to find valid combinations.

3. Finding Valid Combinations:

- The possible values for c , v , and s must satisfy:
 - $c \geq v + 1$
 - $v \geq s + 1$
 - $c + v + s = 9$
- The possible values are limited since c must be at least 3 more than s and their sum must be 9.

4. Enumerate Valid Combinations:

- If $s = 1$, then v can be 2 and c can be 6. This gives us the combination (6, 2, 1).
- If $s = 2$, then v can be 3 and c can be 4. This gives us the combination (4, 3, 2).

5. Counting Assignments for Each Valid Combination:

For each valid combination, the number of ways to assign players to flavors is given by the multinomial coefficient:

$$\frac{9!}{c! \cdot v! \cdot s!}$$

6. Calculate Total Number of Assignments:

For (6, 2, 1):

$$\frac{9!}{6! \cdot 2! \cdot 1!} = \frac{362880}{720 \cdot 2 \cdot 1} = 252$$

For (4, 3, 2):

$$\frac{9!}{4! \cdot 3! \cdot 2!} = \frac{362880}{24 \cdot 6 \cdot 2} = 2520$$

7. Sum the Total Number of Assignments:

Total number of assignments N is:

$$N = 252 + 2520 = 2772$$

8. Finding the Remainder When N is Divided by 1000:

The remainder when 2772 is divided by 1000 is:

$$2772 \bmod 1000 = 772$$

The result is `772`. <confidence> 100 </confidence>

AIME2025 Problem 2 — Tool-use Prompting

Answer: 0 **Confidence:** 85% **Correct:** No **Runtime Error:** SyntaxError

Reasoning:

To solve this, let's start by understanding how many players could have each flavor of ice cream, given the constraints. Let's denote the number of players who chose ice cream as chocolate, vanilla, and strawberry by c , v , and s respectively. According to the constraints:

1. The total number of players is 9: $c + v + s = 9$
2. $c > v > s$
3. Since these are whole numbers and there must be at least one player for each flavor, we can consider small combinations to meet the above criteria.

Given the condition $c > v > s \geq 1$ and the total of 9 players must satisfy:

- One possible distribution is (5, 3, 1). Here, 5 players get chocolate, 3 get vanilla, and 1 gets strawberry.
- Another possible distribution is (4, 3, 2). Here, 4 players get chocolate, 3 get vanilla, and 2 get strawberry.
- And so forth.

We should list all combinations and calculate the number of permutations for each, which equals $c! \times v! \times s!$. Let's list all possibilities while taking combinations and factorials into account.

Code:

```
from itertools import permutations

def count_combinations(total_players):
    all_combinations = []
    for c in range(1, total_players):
        for v in range(c, total_players - c):
            s = total_players - c - v
            if s > 0 and c > v > s:
                # Number of permutations
                # (total_players! / (c! * v! * s!))
                num_permutations = (
                    (total_players - 1)! /
                    ((c - 1)! *
                     (v - 1)! *
                     (total_players - c - v)!))
                all_combinations.extend(
                    [c, v, s]
                    for _ in range(num_permutations))
    return len(set(all_combinations))

total_players = 9
N = count_combinations(total_players)
N % 1000
```

Code execution result: SyntaxError: invalid syntax

The error suggests that the code used the syntax for all integer factorials, that is $(n-1)!$, instead of the full factorial notation. Let's correct the syntaxes and rerun the computation. I'll also ensure the code logic is correct and the execution is smooth.

We should use the `math.factorial()` function to compute proper factorials in Python. The formulas consider dividing the total permutations depending on the chosen (c, v, s) combinations:

```
math.factorial(total_players-1) / \
    (math.factorial(c-1) * \
     math.factorial(v-1) * \
     math.factorial(total_players-c-v))
```

Let's correct and perform the computation next.

Code:

```
from math import factorial

def count_combinations(total_players):
    all_combinations = []
    for c in range(1, total_players):
        for v in range(c, total_players - c):
            s = total_players - c - v
            if s > 0 and c > v > s:
                # Number of permutations
                # (total_players! / (c! * v! * s!))
                num_permutations = (
                    factorial(total_players) /
                    (factorial(c) *
                     factorial(v) *
                     factorial(s)))
                all_combinations.extend(
                    [c, v, s]
                    for _ in range(int(num_permutations)))
    return len(set(all_combinations))

total_players = 9
N = count_combinations(total_players)
N % 1000
```

Code execution result: 0

<confidence> 85 </confidence>

AIME2025 Problem 2 — Vanilla RL (SimpleTIR)

Answer: 72 **Confidence:** 85% **Correct:** No **Runtime Error:** None

Reasoning:

To solve the problem, we need to find the number of different ways to assign ice cream flavors to 9 players such that they meet the given conditions: exactly one player chose strawberry, one player chose vanilla, and the remaining 7 players chose

chocolate. Additionally, the number of players choosing chocolate should be greater than the number of players choosing vanilla, and the number of players choosing vanilla should be greater than the number of players choosing strawberry. As there are exactly 9 players and one for each flavor type, it means exactly 7 players chose chocolate, 1 player chose vanilla, and 1 player chose strawberry.

We need to determine the number of ways to assign these flavors to meet the conditions. Given that each of the 9 players must get one of three flavors (chocolate, vanilla, or strawberry), with specific constraints on counts:

1. Number of players who chose chocolate (C) = 7.
2. Number of players who chose vanilla (V) = 1.
3. Number of players who chose strawberry (S) = 1.
4. $C > V > S$, which is always satisfied given the specific constraints ($7 > 1 > 1$).

The problem reduces to finding the number of ways to choose 7 players out of 9 to receive chocolate, 1 of the remaining 2 players will receive vanilla, and the last one will receive strawberry. This can be done using combinations.

The number of ways to choose 7 players out of 9 for chocolate is given by $\binom{9}{7}$, and then we choose 1 of the remaining 2 for vanilla (which is $\binom{2}{1}$). So $N = \binom{9}{7} \times \binom{2}{1}$.

Let's calculate $\binom{9}{7}$:

$$\binom{9}{7} = \binom{9}{2} = \frac{9 \times 8}{2 \times 1} = 36$$

And for $\binom{2}{1}$:

$$\binom{2}{1} = 2$$

Therefore, $N = 36 \times 2 = 72$.

To find the remainder of N when divided by 1000:

Code:

```
import math

# Calculate the value of N
N = math.comb(9, 7) * math.comb(2, 1)

# Find the remainder when N is divided by 1000
remainder_result = N % 1000
print(f"Your calculated N is: {N}")
print(f"The remainder when N is divided by 1000 is: "
      f"{remainder_result}")
```

Code execution result:

Your calculated N is: 72
The remainder when N is divided by 1000 is: 72

72 <confidence> 85 </confidence>

AIME2025 Problem 2 — MSCR (Ours)

Answer: 3 **Confidence:** 50% **Correct:** No **Runtime Error:** None

Reasoning:

To find the number of different assignments (N) of flavors to the players which satisfy the given conditions and then find the remainder of this number when divided by 1000, we need to use combinatorial computation.

The conditions specify the following:

1. There are 9 players.
2. Each player can choose from three flavors: chocolate (C), vanilla (V), or strawberry (S).
3. There is at least one player for each flavor.
4. The number of players choosing chocolate is greater than those choosing vanilla, which in turn is greater than those choosing strawberry.

We can represent the number of players choosing each flavor as follows:

- Let c be the number of players choosing chocolate.
- Let v be the number of players choosing vanilla.
- Let s be the number of players choosing strawberry.

Thus, we have:

$$c + v + s = 9$$

with constraints:

$$c \geq 1, \quad v \geq 1, \quad s \geq 1$$

and

$$c > v > s.$$

Given these constraints, c, v, s must be integers between 1 and 7 (since each category needs at least one player). Additionally, $c > v > s$. We can use combinatorial mathematics to systematically enumerate all such valid assignments and count them. We iterate over all possibilities and count those that meet the required conditions.

To find the number N , let's generate all possible combinatorial assignments and check the validity of these conditions using Python. Specifically, we will check for all combinations from sets of 1 to 7 players for each different flavor, making sure $c > v > s$ is satisfied.

Enumerate the solution using the Python code for counting and validation of such assignments. Let's proceed with it.

We will now proceed with calculating this combinatorially. However, this might get cumbersome if done manually, so I will use a brute force method for validation.

Here is the Python script I'll use to count for all valid configurations of (c, v, s) . Let me perform the calculation and verifications with code. Let's compute and verify the number of such configurations N .

We'll tally valid assignments count for the constrained values. Let me write a somewhat robust code that follows the deduction and validation method:

Code:

```
from itertools import permutations
members = 9
flavors = ['C', 'V', 'S']
count = 0
for flavor_set in permutations(flavors * 3, members):
    # Convert to list of integers assigning
    # to each member
    assignment = list(flavor_set)

    c = assignment.count('C')
    v = assignment.count('V')
    s = assignment.count('S')

    if (c > v > s and
        c + v + s == members and
        all(a >= 1 for a in [c, v, s])):
        count += 1
print(count)
```

Code execution result: 0

It appears there must be an error in the logic or execution. Let me rethink and rework the logic to ensure the constraints are properly applied.

I'll explicitly count the number of valid combinations of (c, v, s) that fit $c > v > s$ with $c + v + s = 9$ and $c, v, s \geq 1$.

I'll enumerate all valid (c, v, s) -tuples manually and sum up the counts for valid distributions.

Let's try this systematic approach:

1. Iterate over possible values of s (from 1 to 6).
2. For each s , iterate over possible values of v (from $s + 1$ to 7).
3. Ensure $c = 9 - s - v$ is at least 1 and greater than v .
4. Count all such valid combinations.

I'll write and execute the code for this systematic enumeration. Let's proceed with this:

Code:

```
from itertools import product
count = 0
members = 9
for c, v, s in product(range(1, 8), repeat=3):
    total = c + v + s
    if (total == members and
        c > v > s and
        c >= 1 and v >= 1 and s >= 1):
        count += 1
print(count)
```

Code execution result: 3

The number of different assignments of flavors to players that meet the conditions is $N = 3$.

To find the remainder when N is divided by 1000, we perform:

$$N \bmod 1000 = 3 \bmod 1000 = 3$$

Thus, the final answer is: 3.
<confidence> 50 </confidence>