Reinforced Agent: Inference-Time Feedback for Tool-Calling Agents

Anh Ta **Junjie Zhu** **Shahin Shayandeh**
Apple
{atta, jason.zhu, shn}@apple.com

Abstract

Tool-calling agents are evaluated on tool selection, parameter accuracy, and scope recognition, yet LLM trajectory assessments remain inherently *post-hoc*. Disconnected from the active execution loop, such assessments identify errors that are usually addressed through prompt-tuning or retraining, and fundamentally cannot course-correct the agent in real time. To close this gap, we move evaluation into the execution loop at *inference time*: a specialized reviewer agent evaluates provisional tool calls *prior to* execution, shifting the paradigm from post-hoc recovery to proactive evaluation and error mitigation.

In practice, this architecture establishes a clear separation of concerns between the primary execution agent and a secondary review agent. As with any multi-agent system, the reviewer can introduce new errors while correcting others, yet no prior work to our knowledge has systematically measured this tradeoff. To quantify this tradeoff, we introduce *Helpfulness-Harmfulness metrics*: helpfulness measures the percentage of base agent errors that feedback corrects; harmfulness measures the percentage of correct responses that feedback degrades. These metrics directly inform reviewer design by revealing whether a given model or prompt provides net positive value.

We evaluate our approach on BFCL (single-turn) and τ^2-Bench (multi-turn stateful scenarios), achieving +5.5% on irrelevance detection and +7.1% on multi-turn tasks. Our metrics reveal that reviewer model choice is critical: the reasoning model o3-mini achieves a 3:1 benefit-to-risk ratio versus 2.1:1 for GPT-4o. Automated prompt optimization via GEPA provides an additional +1.5–2.8%. Together, these results demonstrate a core advantage of separating execution and review: the reviewer can be systematically improved through model selection and prompt optimization, without retraining the base agent.

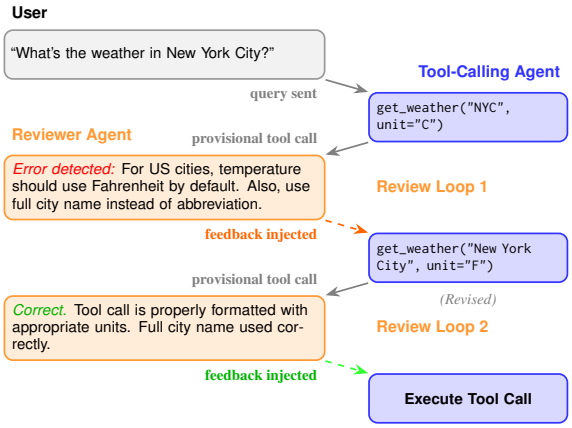

Figure 1: Example trajectory with inference-time feedback. The feedback agent (o3-mini) evaluates provisional tool calls from the tool-calling agent (GPT-4o) before execution. Loop 1: feedback provided. Loop 2: revised call approved.

1 Introduction

Large language models are increasingly deployed as agents that interact with external tools and APIs. These tool-calling agents face systematic challenges: selecting the correct tool, constructing calls with appropriate arguments, and recognizing when no tool can address a request (Patil et al., 2023, 2024; Kokane et al., 2025).

Two main classes of strategies address these challenges. **Training-based approaches** like GRPO (Tang et al., 2024) require substantial compute and slow deployment. **Inference-time approaches** like Self-Refine (Madaan et al., 2023) and Reflexion (Shinn et al., 2023) enable self-correction without training, but require complex infrastructure and context management when agents must simultaneously generate and reflect on tool calls.

Both strategies face a fundamental **state recovery problem**. When an agent executes an incorrect action (such as deleting an alarm instead of updating it), self-correction requires maintaining the previous state in context. This becomes prohibitively

Proceedings of the Fifth Workshop on Generation, Evaluation and Metrics (GEM), pages 136–147

expensive in complex execution environments and multi-turn scenarios, where the space of alternative trajectories grows exponentially. Without unreasonably large context windows (limited by model capacity and context budget), agents cannot reliably recover from destructive errors.

To address the native challenges of tool-calling agents and the state recovery problem, we propose **inference-time feedback** using a simple, configurable duo-agent architecture: a specialized reviewer agent evaluates provisional tool calls before execution and either provides feedback to the tool-calling agent or uses a selection strategy to choose among candidates. An example trajectory with the reviewer model is shown in Figure 1. The key proposal is simple separation of concerns, which yields strong benefits. First, it requires only one additional agent (configurable by model and review strategy) rather than complex infrastructure changes. Second, by reviewing calls before execution, it helps mitigate destructive errors rather than attempting recovery, reducing the state recovery problem. The tool-calling agent requires no retraining or rearchitecture and seamlessly adopts feedback from the reviewer.

However, introducing a reviewer brings tradeoffs: feedback can mitigate errors but can also break valid responses. To quantify this, we introduce **Helpfulness-Harmfulness metrics** quantifying how often feedback corrects errors versus introduces new ones. Reviewer quality can be improved through model capacity or prompt optimization. Latency overhead can be reduced through distillation.

To find the optimal feedback agent configuration, we explore multiple review strategies (progressive feedback, best-of-N selection, and best-of-N grading) and address reviewer failures through automated prompt optimization (APO). APO optimizes only the reviewer's prompt (the main agent's prompt remains unchanged), automatically refining it by observing cases where the reviewer made incorrect judgments.

For the reviewer agent, we compare non-reasoning (GPT-4o) and reasoning models to assess the impact of reasoning on review quality. We use o3-mini for initial experiments, then GPT-5 mini (adopted upon release) for APO experiments to leverage the latest reasoning capability. We chose mini variants to balance reasoning capability with cost efficiency. The main tool-calling agent remains GPT-4o throughout.

Key Results We evaluate on two benchmarks: BFCL (Berkeley Function-Calling Leaderboard) (Patil et al., 2024) for single-turn function calling and τ^2-Bench (Barres et al., 2025) for multi-turn stateful scenarios. Our best configuration achieves +5.5% on irrelevance detection (84.9% $\rightarrow$ 90.4%), +1.6% on the relevance suite (90.9% $\rightarrow$ 92.5%) on BFCL (Table 5), and +7.1% on τ^2-Bench (48.7% $\rightarrow$ 55.8%; Table 8). Using our Helpfulness-Harmfulness metrics, we find that reasoning models (o3-mini) outperform standard language models as reviewers, achieving a 3:1 benefit-to-risk ratio (36.8% helpfulness, 11.7% harmfulness; Figure 4).

Reasoning model comparison and APO are evaluated on BFCL only; extending these to τ^2-Bench remains future work.

This approach provides practical benefits for tool-calling systems: it requires no retraining or infrastructure modifications, supports rapid iteration on reviewer strategies through automated optimization, and offers tunable accuracy-latency trade-offs for different application requirements. The modular architecture enables organizations to enhance agent reliability incrementally while leaving existing tool-calling pipelines unchanged.

Key Contributions In summary, our work makes the following contributions:

1. **Inference-time feedback mechanism** improving tool-calling performance without training, achieving +5.5% on irrelevance detection (BFCL) and +7.1% on multi-turn scenarios (τ^2-Bench).

2. **Helpfulness-Harmfulness metrics** quantifying benefit-risk tradeoffs of feedback interventions, showing reasoning models achieve 3:1 ratios versus standard language models (BFCL).

3. **Automated reviewer prompt optimization** systematically discovering effective review strategies via GEPA (Agrawal et al., 2025), achieving +1.5% (relevance) and +2.8% (irrelevance) on BFCL (Table 5).

2 Method

2.1 Multi-Agent Architecture

We evaluate three collaboration mechanisms between a tool-calling agent and a reviewer agent:

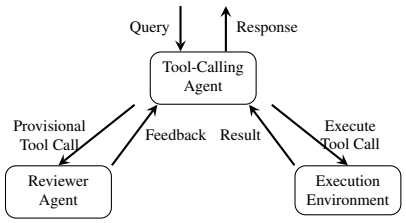

Figure 2: Feedback Architecture. The reviewer agent reviews provisional tool calls before execution. If errors are detected, feedback is provided for revision. This loop continues until approval or maximum iterations (N) is reached.

Progressive Feedback: The feedback agent iteratively reviews the tool-calling agent's responses (Figure 2). If errors are found, feedback is injected as a system message and the tool-calling agent generates a revised response. This continues for up to N review loops or until no errors are detected. We denote this as `rN` (e.g., `r2` for up to 2 review loops).

Best-of-N Selection (Selector): The tool-calling agent generates N candidate responses with varying temperatures (0.3 to 1.0). The selector agent evaluates all candidates and selects the best one. This single-shot selection is denoted as `sN` (e.g., `s5` for 5 candidates).

Best-of-N Grading (Grader): Similar to selection, but the grader agent assigns explicit numeric scores (0.0-1.0) to each candidate with rationales. The highest-scored candidate is selected. Denoted as `gN` (e.g., `g5`).

We use a systematic naming convention for all mechanisms when reporting the evaluation results. For example, `4o-r2-5-mini-v3-gepa` indicates GPT-4o base model, progressive feedback with up to 2 feedback loops (r2), GPT-5 mini feedback model (5-mini), GEPA prompt version 3. See Appendix A.3 for concrete examples of how each mechanism operates.

2.2 Reviewer Prompt Optimization

Manual prompt engineering for reviewer agents is labor-intensive and may miss subtle failure patterns. Inspired by GEPA (Agrawal et al., 2025), which iteratively improves prompts based on execution feedback, we automatically refine reviewer prompts by observing cases where the reviewer made incorrect judgments. Starting from manually-refined v2 prompts for BFCL, we iteratively improve them using a reasoning model for reflection. Applying this optimization to τ^2-Bench prompts remains future work. Details and results appear in Section 4.4.3.

3 Experiment Setup

3.1 Benchmarks

We evaluate on two benchmarks: BFCL (Berkeley Function Calling Leaderboard) for single-turn tool calling and τ^2-Bench for multi-turn stateful scenarios.

3.1.1 BFCL

Single-turn, stateless tool calling. We evaluate on Non-Live (BFCL V1, curated) and Live (BFCL V2, community-contributed) categories (Patil et al., 2024). Categories include `simple`, `multiple`, `parallel`, and `parallel_multiple` (combined parallel and sequential; hardest), which together form the `relevance suite`. The `irrelevance` category tests detecting when no tool is relevant.

3.1.2 τ^2-Bench

Multi-turn, stateful tool calling with domain-specific policies across three domains (airline, retail, telecom). Agents must maintain conversational context, verify state preconditions, and handle benchmark-specific constraints.

3.2 Models

All experiments use GPT-4o (gpt-4o-2024-11-20 snapshot) as the base tool-calling agent with temperature=0 and seed=42 for reproducibility. Initial experiments use o3-mini as the reasoning model reviewer; APO experiments use GPT-5 mini (adopted upon release to leverage the latest reasoning capability). Both reasoning models use reasoning_effort=medium.

4 Results & Analysis

We organize our evaluation around three research questions:

- **RQ1 (Effectiveness & Error):** What is the effectiveness of inference-time feedback for tool-calling agents, and what are the associated error correction trade-offs?
- **RQ2 (Design & Optimization):** How do feedback mechanism design, reviewer model selection, and automated optimization affect reviewer agent performance?
- **RQ3 (Latency & Deployment):** What are the latency overhead and deployment trade-offs

of inference-time feedback across different application scenarios?

We answer these questions using both standard benchmark metrics and additional metrics that quantify error correction versus error introduction (Section 4.1). We then address RQ1 by evaluating effectiveness on BFCL and τ^2-Bench (Section 4.2), RQ2 by comparing reviewer models, feedback mechanisms, and automated prompt optimization (Section 4.4), and RQ3 by analyzing latency overhead and deployment trade-offs (Section 4.5).

4.1 Evaluation Metrics

We evaluate using each benchmark's default metrics: per-category accuracy for BFCL (simple, multiple, parallel, parallel_multiple, irrelevance) and relevance suite (unweighted average of the first four), and per-domain pass rate for τ^2-Bench (airline, retail, telecom). To complement these standard metrics, we introduce three metrics that quantify reviewer quality and error correction trade-offs:

- **Helpfulness:** Percentage of test cases where base agent is wrong AND reviewer agent corrects it.
- **Harmfulness:** Percentage of test cases where base agent is right AND reviewer introduces error.
- **Benefit-to-Risk Ratio:** Helpfulness ÷ Harmfulness.

These metrics reveal whether feedback provides net positive value, showing not just final accuracy but how the reviewer affects correct and incorrect responses.

4.2 Effectiveness & Error Trade-offs

RQ1: What is the effectiveness of inference-time feedback for tool-calling agents, and what are the associated error correction trade-offs?

4.2.1 BFCL Evaluation

We first evaluate inference-time feedback on BFCL to establish baseline effectiveness on single-turn, stateless tool calling. Results are averaged across experimental sessions.

Initial Results With minimal v1 prompts and GPT-4o as reviewer, we improved irrelevance detection but yielded only marginal gains on other categories.

Category	Baseline	+ Reviewer	Δ
Simple	92.4%	92.8%	+0.4%
Multiple	92.8%	93.0%	+0.2%
Irrelevance	**84.9%**	**89.6%**	**+4.7%**
Rel. Suite	90.9%	91.4%	+0.5%

Table 1: Initial Reviewer Impact (4o-r5-4o-v1).

Root Cause Analysis The reviewer incorrectly flagged valid tool calls as "incomplete," expecting execution results. However, as shown in Figure 2, the reviewer evaluates provisional tool calls before execution, so no results exist yet. Analysis showed 23% of cases had redundant iterations where the reviewer demanded elaboration it should not expect. Figure 3 illustrates a typical example of this mismatch.

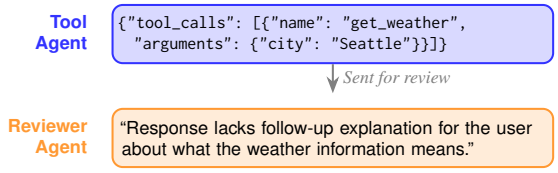

Figure 3: Reviewer misunderstanding example. The reviewer expects user-facing dialogue, but BFCL only evaluates tool call accuracy.

Addressing Over-Skepticism We added an explicit guideline addressing the over-skepticism failure mode:

> **[CRITICAL] Tool-only responses are complete.** Do not mark tool-only responses as incomplete for lacking user-facing answers, follow-up explanations, or final results. Tool calls are standalone steps. Focus on whether the actual tool calls are correct.

The explicit guidance reduced redundant review loops from 23% to 8%, improving efficiency. However, v2 prompts reduced harmfulness but did not fully solve the irrelevance detection challenge. This motivated comparing different review models.

Reviewer	Prompt	Rel.	Irrel.	Help.	Harm.	Ratio
GPT-4o	v1	89.5%	90.0%	30.2%	14.2%	2.1:1
GPT-4o	v2	91.0%	90.4%	34.9%	12.9%	2.7:1
o3-mini	v2	**91.8%**	**91.0%**	**36.8%**	**11.7%**	**3.1:1**

Table 2: Helpfulness vs. Harmfulness on BFCL Non-Live. Results from dedicated experiment measuring reviewer quality (see Figure 4). Rel. = relevance suite, Irrel. = irrelevance, Help. = helpfulness, Harm. = harmfulness.

Table 2 compares o3-mini and GPT-4o as reviewer models with manually-engineered prompts

(v1, v2; see Appendix A.4 for full prompts). The o3-mini configuration achieves the best performance: 91.8% on the relevance suite and 91.0% on irrelevance detection. Beyond accuracy, the helpfulness and harmfulness metrics reveal critical differences in reviewer quality. The o3-mini configuration corrects 36.8% of base agent errors while introducing errors in only 11.7% of previously correct responses, achieving a 3.1:1 benefit-to-risk ratio (Figure 4). A detailed analysis of model comparison is presented in Section 5.2.

4.3 τ^2-Bench Evaluation

We extend evaluation to τ^2-Bench to test whether feedback mechanisms generalize to multi-turn, stateful scenarios with domain-specific policies. Results are averaged over 3 benchmark runs.

Initial Results Table 3 shows performance across three domains. The best configuration (4o-r5-4o-v1) achieves +7.1% average improvement over baseline (48.7% → 55.8%).

Configuration	Airline	Retail	Telecom	Average
4o baseline	42.0%	62.9%	41.2%	48.7%
Progressive Feedback (rN)				
4o-r5-4o-v1	40.7%	62.6%	**64.0%**	**55.8%**
4o-r5-4o-v2	40.7%	58.5%	59.6%	52.9%
Best-of-N Selection (sN)				
4o-s5-4o-v1	42.7%	61.1%	38.0%	47.3%
4o-s5-4o-v2	48.0%	61.4%	48.2%	52.5%
Best-of-N Grading (gN)				
4o-g5-4o-v1	47.3%	60.5%	45.3%	51.0%
4o-g5-4o-v2	46.7%	**65.8%**	48.8%	53.8%

Table 3: Performance on τ^2-Bench across all mechanisms. Progressive Feedback achieves the highest average (55.8%). See Table 8 for additional details.

Error analysis We analyzed failures to understand where feedback helps and hurts. Table 4 shows distribution.

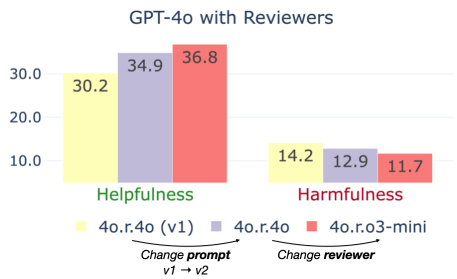

Figure 4: Helpfulness vs. Harmfulness trade-off on BFCL Non-Live. o3-mini achieves the best balance (3.1:1 ratio).

Error Type	Baseline	With Reviewer
Policy Constraint Violation	31%	18% (-13%)
Missing Context Awareness	24%	15% (-9%)
Incorrect Tool Selection	19%	22% (+3%)
Argument Errors	16%	18% (+2%)
Over-verbalization	10%	27% (+17%)

Table 4: Failure Mode Distribution on τ^2-Bench

For example, in an airline booking domain, the policy requires checking seat availability before booking. Base agent error: directly calls `book_flight()` without calling `check_availability()` first. Reviewer agent catches: "Response violates state precondition: seat availability must be checked."

The reviewer effectively catches policy violations (-13%) but introduces new "over-verbalization" errors (+17%). The over-skepticism problem observed in BFCL recurs here: the reviewer flags tool-only responses in contexts where τ^2-Bench expects mixed responses. Comparing mechanisms, Progressive Feedback substantially outperforms Best-of-N approaches (+3–8% on average), with selection actually underperforming the baseline on some domains (Table 3).

Cross-benchmark generalization: Applying BFCL-optimized prompts (v2-bfcl) directly to τ^2-Bench introduces errors, highlighting the domain mismatch between single-turn function calling and multi-turn stateful agents. The different evaluation criteria and interaction patterns require benchmark-specific prompt adaptation.

Addressing Domain Constraints τ^2-specific prompts (v2-tau; see Appendix A.4) emphasize context awareness (marked CRITICAL), state preconditions, and benchmark-specific constraints. However, baseline reviewer prompts (v1) sometimes outperform domain-tuned reviewer prompts on average across domains (55.8% vs. 52.9%; Table 3), suggesting that manually-engineered instructions may not generalize across all tasks within a domain. This opens an opportunity for automated prompt optimization on τ^2-Bench as future work.

4.4 Design & Optimization

RQ2: How do feedback mechanism design, reviewer model selection, and automated optimization affect reviewer agent performance?

4.4.1 Model Comparison

On the BFCL benchmark, we compare o3-mini and GPT-4o as reviewer models with manually-engineered prompts (v1, v2; see Appendix A.4 for full prompts) for initial experiments, then GPT-5 mini (its successor, as the more up-to-date reasoning model) for automated reviewer prompt optimization using GEPA (v3-gepa).

The reasoning model's advantage manifests in two key areas. First, o3-mini outperforms GPT-4o on irrelevance detection by +0.6% (91.0% vs. 90.4%), a category that requires determining whether any available tool can address the request. Second, o3-mini exhibits lower harmfulness (11.7% vs. 12.9%; Table 2), making fewer false positive errors. The reasoning model's systematic verification reduces over-correction, leading to more reliable feedback.

Irrelevance detection: o3-mini outperforms GPT-4o by +0.6% on irrelevance (91.0% vs. 90.4% for v2 prompts; Table 2). This category requires determining whether any available tool can address the request.

Error introduction rates: o3-mini exhibits lower harmfulness (11.7% vs. 12.9% for GPT-4o v2; Table 2), making fewer false positive errors. The reasoning model's systematic verification reduces over-correction.

Key insight: The goal is maximizing helpfulness while minimizing harmfulness (Figure 4). The o3-mini configuration achieves the best balance: 36.8% of base agent errors corrected with 11.7% new errors introduced, achieving a 3.1:1 benefit-to-risk ratio. This demonstrates that reasoning models provide better feedback quality than standard language models.

4.4.2 Mechanism Comparison

We evaluate three feedback mechanisms on BFCL to understand their strengths: Progressive Feedback (iterative refinement with feedback), Best-of-N Selection (choosing the best response from multiple candidates), and Best-of-N Grading (scoring and selecting from multiple candidates). Appendix Table 6 provides complete results across all mechanisms and categories.

Reasoning model advantage: As shown in Table 2, o3-mini achieves the best helpfulness-to-harmfulness ratio (3.1:1) compared to GPT-4o (2.7:1 for v2 prompts). This suggests that reasoning models provide more reliable feedback with fewer false corrections.

Mechanism effectiveness: Progressive Feedback outperforms Best-of-N approaches on irrelevance detection by +4–5%, while Best-of-N shows only marginal gains on relevance. Progressive Feedback's advantage lies in explicitly identifying and correcting errors through targeted feedback, particularly for irrelevance where the model must determine no tool can address the request.

4.4.3 Automatic Context Optimization

Having established effectiveness on BFCL and demonstrated generalization on τ^2-Bench, we now systematize improvements through automated reviewer prompt optimization. We apply GEPA to BFCL reviewer prompts; extending this to τ^2-Bench remains future work.

GEPA Algorithm and Results GEPA (Genetic-Pareto prompt evolution) addresses limitations of manual prompt engineering. The algorithm starts with manually-engineered reviewer prompts (v2), collects failure cases, uses an LLM to reflect and propose improvements, and iterates until convergence. This produces v3 prompts approximately 4.5× longer (1,599 vs. 358 tokens)[1] with detailed error criteria, edge case handling, and error checklists (Appendix A.4).

Configuration	Rel.	Irrel.	Gain
4o-r2-5-mini-v2	91.0%	87.6%	baseline
4o-r2-5-mini-v3-gepa	92.5%	90.4%	+1.5% / +2.8%

Table 5: GEPA Optimization on BFCL Non-Live (Progressive Feedback). v2 = manual, v3-gepa = GEPA-optimized. Rel. = relevance suite, Irrel. = irrelevance.

The GEPA-optimized prompts show improvements across categories, with particularly strong gains on the parallel_multiple category (+2.1%; Table 6). This category involves orchestrating multiple parallel tool calls with correct argument passing, where APO-optimized feedback catches mismatches and missing calls, demonstrating automated optimization discovers strategies difficult to anticipate manually.

4.5 Latency & Deployment Trade-offs

RQ3: What are the latency overhead and deployment trade-offs of inference-time feedback across application scenarios?

[1] Measured using OpenAI's tokenizer (https://platform.openai.com/tokenizer) with the GPT-5.x & O1/3 option.

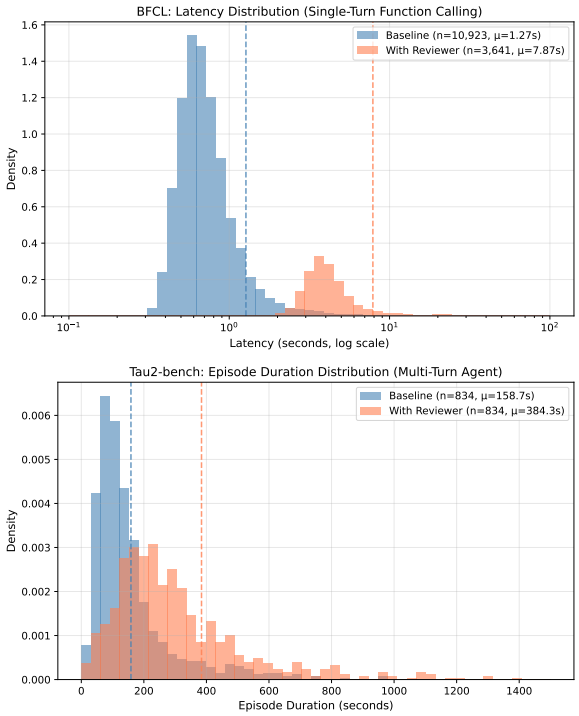

Figure 5: Latency distributions. Top: BFCL per-item latency (log scale). Bottom: τ^2-Bench per-episode duration. Blue = baseline, coral = with reviewer. Dashed lines = means.

Inference-time feedback introduces computational overhead that varies by task type. We analyze latency measurements from our experiments to characterize the cost-accuracy trade-off.

Latency Analysis: We measure latency impact on both BFCL (single-turn function calling) and τ^2-Bench (multi-turn agent episodes). On BFCL, the feedback mechanism (r5-4o) increases average latency from 1.27s to 7.87s, a **6.2× multiplier**. This increase occurs because the baseline is a single inference call, so reviewer overhead dominates. On τ^2-Bench, the same mechanism increases average episode duration from 158.7s to 384.3s, a **2.4× multiplier**. The lower relative impact reflects the multi-turn nature: reviewer overhead is amortized across approximately 40 turns per episode. Figure 5 shows the latency distributions.

Reviewer Call Patterns: On BFCL, feedback averages 1.33 reviewer calls per item. On τ^2-Bench, the mechanism averages 0.96 reviewer calls per turn. The per-turn call count is lower for τ^2-Bench likely due to stateful nature reducing ambiguity in later turns.

Application-Specific Deployment: The latency-accuracy trade-off allows flexible deployment strategies:

- **Single-turn, high-volume applications:** The 6.2× latency overhead may be prohibitive for real-time use cases. Consider baseline deployment or selective feedback on uncertain cases.
- **Multi-turn agents:** The 2.4× overhead is more acceptable when amortized across conversation turns. Feedback mechanisms are viable for complex, accuracy-critical workflows.
- **API-intensive workflows:** Feedback may achieve ROI-positive gains by preventing spurious API calls that incur downstream costs.

5 Related Work

Tool-Calling Benchmarks. BFCL (Patil et al., 2024, 2023) evaluates single-turn function calling across categories. ToolSandbox (Lu et al., 2025) introduces stateful, conversational evaluation. τ^2-Bench tests multi-turn tool calling with domain-specific policies. Our work is the first to systematically compare feedback mechanisms across both stateless (BFCL) and stateful (τ^2-Bench) benchmarks.

Agent Feedback and Self-Refinement. Self-Refine (Madaan et al., 2023) and Reflexion (Shinn et al., 2023) use self-feedback for iterative refinement. Our work differs by using a specialized reviewer agent instead of self-feedback, demonstrating that external feedback from reasoning models outperforms self-correction.

Prompt Optimization. MIPROv2 (Opsahl-Ong et al., 2024) uses Bayesian optimization for instructions and few-shot examples. GEPA (Agrawal et al., 2025) uses genetic-Pareto evolution with LLM reflection. We apply GEPA to reviewer agent prompts, showing improvements over manual engineering.

Training-Based Approaches. GRPO (Tang et al., 2024) fine-tunes via weight-space RL but requires thousands of rollouts. Recent reasoning models (o1, o3) (OpenAI, 2024) demonstrate strong verification capabilities. Knowledge distillation (Hinton et al., 2015; Ho et al., 2023; Fu et al., 2023) provides a foundation for future work on distilling reviewer agents. Our inference-time approach provides immediate gains without training.

6 Conclusion

We introduce Reinforced Agent, an inference-time feedback mechanism for tool-calling agents. Evaluation on BFCL establishes baseline effectiveness and identifies over-skepticism as the primary reviewer error mode. Evaluation on τ^2-Bench demonstrates generalization to multi-turn scenarios. Our Helpfulness-Harmfulness metrics show reasoning models achieve favorable benefit-to-risk ratios as reviewers. Automated prompt optimization via GEPA systematizes reviewer improvement over manual engineering. Latency overhead is most viable for complex, accuracy-critical workflows where it amortizes across turns, and can be further mitigated by distilling the reviewer into a smaller, faster model (e.g., a lightweight reward model or classifier) suitable for local or on-device deployment. This work establishes a practical path from benchmarking to optimization to deployment, with potential for distillation into reward models for reinforcement learning.

7 Limitations

Our work has several limitations.

Base model scope. We evaluate only GPT-4o as the base tool-calling agent and compare two reviewer models (GPT-4o and o3-mini). While the modular architecture imposes no constraints on the base model (the reviewer operates on tool-call outputs regardless of which model generates them), generalization to open-source models (e.g., Llama, Mistral) and smaller proprietary models remains empirically unvalidated.

Analysis scope. GEPA-based prompt optimization and Helpfulness-Harmfulness metrics were applied only to BFCL; extending these to multi-turn benchmarks like τ^2-Bench requires adapting for partial-credit scoring and multi-turn error propagation. Additionally, generic prompts (v1) sometimes outperform domain-specific prompts (v2-tau) on τ^2-Bench, suggesting manual tuning may not generalize without automated optimization.

8 Ethics Statement

To the best of our knowledge, all results published in this paper are accurate. All data sources are publicly available benchmarks and are cited accordingly. No human subjects, private user data, or personally identifiable information were used in this work.

Acknowledgments

We thank Jaechan Lee for contributions to the exploratory experiments in this study during his internship at Apple. We are grateful to Sylvia Xu and Nishant Kanakia for detailed feedback on multiple drafts of this paper. We also thank Anatoly Adamov, Alex Braunstein, and Amar Subramanya for their continued support of this research.

References

Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, Christopher Potts, Koushik Sen, Alexandros G. Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab. 2025. Gepa: Reflective prompt evolution can outperform reinforcement learning. *Preprint*, arXiv:2507.19457.

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

Yao Fu, Hao Peng, Litu Ou, Ashish Sabharwal, and Tushar Khot. 2023. Specializing smaller language models towards multi-step reasoning. *Preprint*, arXiv:2301.12726.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network. *Preprint*, arXiv:1503.02531.

Namgyu Ho, Laura Schmid, and Se-Young Yun. 2023. Large language models are reasoning teachers. *Preprint*, arXiv:2212.10071.

Shirley Kokane, Ming Zhu, Tulika Awalgaonkar, Jianguo Zhang, Thai Hoang, Akshara Prabhakar, Zuxin Liu, Tian Lan, Liangwei Yang, Juntao Tan, Rithesh Murthy, Weiran Yao, Zhiwei Liu, Juan Carlos Niebles, Huan Wang, Shelby Heinecke, Caiming Xiong, and Silivo Savarese. 2025. Toolscan: A benchmark for characterizing errors in tool-use llms. *Preprint*, arXiv:2411.13547.

Jiarui Lu, Thomas Holleis, Yizhe Zhang, Bernhard Aumayer, Feng Nan, Felix Bai, Shuang Ma, Shen Ma, Mengyu Li, Guoli Yin, Zirui Wang, and Ruoming Pang. 2025. Toolsandbox: A stateful, conversational, interactive evaluation benchmark for llm tool use capabilities. *Preprint*, arXiv:2408.04682.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. 2023. Self-refine: Iterative refinement with self-feedback. *Preprint*, arXiv:2303.17651.

OpenAI. 2024. Learning to reason with llms. Accessed: 2025-02-01.

Krista Opsahl-Ong, Michael J Ryan, Josh Purtell, David Broman, Christopher Potts, Matei Zaharia, and Omar Khattab. 2024. Optimizing instructions and demonstrations for multi-stage language model programs. *Preprint*, arXiv:2406.11695.

Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. 2023. Gorilla: Large language model connected with massive apis. *Preprint*, arXiv:2305.15334.

Shishir G Patil, Tianjun Zhang, Xin Wang, and Joseph E Gonzalez. 2024. Berkeley function calling leaderboard. https://gorilla.cs.berkeley.edu/leaderboard.html.

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

Yunhao Tang, Zhaohan Daniel Guo, Zeyu Zheng, Daniele Calandriello, Rémi Munos, Mark Rowland, Pierre Harvey Richemond, Michal Valko, Bernardo Ávila Pires, and Bilal Piot. 2024. Generalized preference optimization: A unified approach to offline alignment. *Preprint*, arXiv:2402.05749.

A Complete Experimental Results

A.1 BFCL Results

Table 6 presents complete results on the Berkeley Function Calling Leaderboard (BFCL) across all configurations tested. We report both Non-Live (BFCL V1, curated) and Live (BFCL V2, community-contributed) categories.[2]

A.2 τ^2-Bench Results

Table 8 presents complete results on τ^2-Bench across all configurations tested.

A.3 Feedback Mechanism Examples

This section illustrates how each feedback mechanism operates with concrete examples using a weather query task.

A.3.1 Progressive Feedback (r2): Iterative Refinement

Progressive feedback allows up to 2 review loops. The process continues until the feedback agent accepts the response or the maximum loops are reached.

[2]BFCL V1 contains curated benchmark tasks; V2 adds community-contributed real-world data. See https://huggingface.co/datasets/gorilla-llm/Berkeley-Function-Calling-Leaderboard for dataset details.

Example execution:

User Query: "What's the weather in New York City?"

Loop 1:

- *GPT-4o generates:* `get_weather(location="NYC", temp_unit="celsius")`
- *o3-mini reviews:* "Error: For US cities, temperature should use Fahrenheit by default."

Loop 2:

- *GPT-4o revises with feedback:* `get_weather(location="New York", temp_unit="fahrenheit")`
- *o3-mini reviews:* "Correct. Tool call is properly formatted with appropriate units."
- *Result:* Accepted. Process terminates after 2 loops.

A.3.2 Best-of-N Grading (g5): Scored Selection

The grader mechanism generates 5 candidate responses at different temperatures and assigns explicit scores to each.

Example execution:

User Query: "What's the weather in New York City?"

Generated Candidates (temperatures 0.3–1.0):

1. `get_weather(location="New York")`
2. `get_weather(location="NYC", temp_unit="C")`
3. `get_weather(location="New York", temp_unit="fahrenheit")`
4. `get_weather("New York")`
5. `get_weather(location="NY")`

o3-mini scores:

- Candidate 1: 0.8 (missing temperature unit but acceptable)
- Candidate 2: 0.3 (uses abbreviation "C" instead of full unit name)
- Candidate 3: **0.9** (complete, proper formatting, correct unit)

Configuration	Simple	Multiple	Parallel	Par_Mult	Irrel.	Rel. Suite
4o (baseline)	92.4	92.8	93.0	85.2	84.9	90.9
Progressive Feedback (Non-Live)						
4o-r5-4o-v1	92.8	93.0	92.5	87.5	89.6	91.4
4o-r5-4o-v2-bfcl	93.0	94.5	92.5	86.0	84.6	91.5
Best-of-N Selection (Selector)						
4o-s5-4o-v1	92.5	93.5	92.5	87.0	85.8	91.4
4o-s5-4o-v2-bfcl	93.2	93.5	93.0	87.5	85.4	91.8
Best-of-N Grading (Grader)						
4o-g5-4o-v1	92.8	93.5	91.5	87.5	85.8	91.3
4o-g5-4o-v2-bfcl	94.0	92.5	92.5	86.5	85.8	91.4
Automated Prompt Optimization (Non-Live)						
4o-r2-5-mini-v1-1	92.2	93.2	91.7	85.2	88.8	90.5
4o-r2-5-mini-v2-bfcl	92.4	93.8	92.2	85.5	87.6	91.0
4o-r2-5-mini-v3-gepa	**95.3**	**94.3**	**93.0**	**87.3**	**90.4**	**92.5**

Table 6: Complete BFCL Non-Live Results. All scores are percentages. *APO*: evaluates GEPA-optimized prompts using GPT-5 mini. Agent naming: `{base}-{mechanism}{N}-{feedback_model}-{prompt_version}`.

Configuration	Simple	Mult.	Irrel.	Rel.
4o (baseline)	79.8	78.9	77.6	79.2
o3-mini Reviewer				
4o-r-o3-mini	82.2	79.0	80.8	79.6
Automated Prompt Optimization (APO)				
4o-r2-5-mini-v1-1	80.6	77.6	83.2	78.1
4o-r2-5-mini-v2-bfcl	80.9	78.4	80.8	78.9
4o-r2-5-mini-v3-gepa	**83.1**	77.4	**83.6**	78.5
Progressive Feedback (GPT-4o feedback)				
4o-r5-4o-v1	78.7	76.8	82.2	77.5
4o-r5-4o-v2-bfcl	80.6	78.8	76.4	79.3
Best-of-N Selection (Selector)				
4o-s5-4o-v1	81.0	79.1	79.5	79.6
4o-s5-4o-v2-bfcl	81.8	**79.2**	78.8	**79.8**
Best-of-N Grading (Grader)				
4o-g5-4o-v1	81.4	78.7	79.5	79.5
4o-g5-4o-v2-bfcl	79.8	78.9	78.7	79.2

Table 7: BFCL Live Results. Live categories are community-contributed and generally more challenging than Non-Live. Parallel categories omitted due to limited Live data.

- Candidate 4: 0.6 (missing keyword argument)
- Candidate 5: 0.7 (abbreviation "NY" less precise than full name)

Result: Selects Candidate 3 with highest score 0.9.

A.3.3 Best-of-N Selection (s5): Direct Selection

The selector mechanism operates similarly to grading but picks the best candidate without explicit scoring.

Example execution:

Configuration	Airline	Retail	Telecom	Avg
4o (baseline)	42.0	62.9	41.2	48.7
Progressive Feedback				
4o-r5-4o-v1	40.7	62.6	**64.0**	**55.8**
4o-r5-4o-v2-tau	40.7	58.5	59.6	52.9
Best-of-N Selection				
4o-s5-4o-v1	42.7	61.1	38.0	47.3
4o-s5-4o-v2-tau	48.0	61.4	48.2	52.5
Best-of-N Grading				
4o-g5-4o-v1	47.3	60.5	45.3	51.0
4o-g5-4o-v2-tau	**46.7**	**65.8**	48.8	53.8

Table 8: Complete τ^2-Bench Results. All scores are percentages. Best results per column in bold.

User Query: "What's the weather in New York City?"

The tool-calling agent generates 5 candidates (same as grading example above).

o3-mini evaluates and selects:

"Candidate 3 is best: it uses the full city name, includes proper keyword arguments, and specifies the appropriate temperature unit for US locations."

Result: Selects Candidate 3 directly based on qualitative evaluation.

A.4 Reviewer Prompt Examples

We present key versions of reviewer prompts showing the evolution from simple instructions to GEPA-optimized policies. Selector and grader prompts minimally adapt these with their own output sections.

A.4.1 v1: Baseline Prompt

> You are evaluating an assistant's response for correctness, considering the full conversational

context.

Output

Evaluate the assistant's response candidate. Output your evaluation in the following format:

{output_start_tag}
{
  reasoning: string, (Detailed evaluation reasoning)
  message: string, (Brief explanation of why the response is correct or incorrect)
  error: boolean (Whether response is erroneous or contextually inappropriate)
}
{output_end_tag}

Your response must be a valid JSON object and must be wrapped between {output_start_tag} and {output_end_tag} tags.

A.4.2 v1-1: With Critical Guideline

Adds the key insight discovered through error analysis:

You are evaluating an assistant's response for correctness, considering the full conversational context.

[CRITICAL] Tool-only responses are complete. DO NOT MARK TOOL-ONLY RESPONSES AS INCOMPLETE ON THE BASIS OF LACKING *USER-FACING ANSWER*, *FOLLOW-UP EXPLANATION*, OR *FINAL RESULTS PRESENTATION* IN THE SAME RESPONSE. Tool call is a standalone step. Marking correct tool-calling responses as incomplete for these reasons is wrong. Focus instead on whether their actual tool calls are correct.

[Output section same as v1]

A.4.3 v2-bfcl: Manually Optimized for BFCL

The baseline manually-engineered prompt (358 tokens).

You are evaluating an assistant's response for correctness, considering the full conversational context.

Criteria

Request Fulfillment
- Is the response necessary and reasonably sufficient given the conversational context and the user request?

Tool Call Correctness *(if response invokes tool calls)*
- Are the selected tools appropriate?
- Any syntax errors relative to the tool doc? Any type errors (e.g., float vs. integer)?
- Are argument assignments correct?
- **Accept sensible defaults**: If the parameter values are not explicitly specified in the user request, using default arguments is acceptable.

Other Guidelines
- **Don't be pedantic**: Take a charitable interpretation of the assistant's response. Critique logical failures, NOT surface-level features like tone, style, or minor inefficiencies.
- **No external facts**: Base judgments solely on the conversation, tool doc, and other available information.
- **Binary solvability**: Tasks are designed to be either reasonably solvable or unsolvable.
- **Unsolvability**: If there is no way to reasonably address the request, recognize this as an unsolvable task rather than an assistant error.

[Output section same as v1]

A.4.4 v2-tau: Manually Optimized for τ^2-Bench

You are evaluating an assistant's response for correctness, considering the full conversational context.

Evaluation Criteria

Context Awareness (CRITICAL)
- Does the response logically follow given the entire conversational context?
- If the task explicitly states constraints, policies, or rules for assistant actions, does the response apply them correctly?
- If the task explicitly states preconditions for assistant actions, does the response verify them before executing actions?

Request Fulfillment (CRITICAL)
- Does the response make a sensible progress towards the complete fulfillment of the user's last request?
- Warning: The response might represent one step in an ongoing multi-step request handling. Partial completion is not erroneous if it represents reasonable logical progress.

Logical Consistency
- If the assistant communicates tool execution outcomes: Do they match what the tools actually returned?
- If the assistant quotes its policy: Does it match what the policy actually states?

Tool Call Correctness *(if response includes tool calls)*
- Are the selected tools appropriate?
- Any syntax errors relative to the tool documentation?
- Are argument assignments correct?

[Additional Guidelines and Output section follow]

A.4.5 v3-2025-09-16-bfcl-gepa: GEPA-Optimized

The most comprehensive version (1,599 tokens). Key sections:

You are evaluating an assistant's response for correctness, considering the full conversational context and the available tool documentation. Judge technical and logical correctness of tool use and task coverage; do not critique tone or style.

[CRITICAL] Tool-only responses are complete. DO NOT MARK TOOL-ONLY RESPONSES AS INCOMPLETE ON THE BASIS OF LACKING *USER-FACING ANSWER*, *FOLLOW-UP EXPLANATION*, OR *FINAL

RESULTS PRESENTATION* IN THE SAME RESPONSE.

Review policy (apply all):
- Evidence scope: Base judgments solely on the conversation and the provided tool documentation. Do not fact-check with outside knowledge.
- Tool selection, necessity, and relevance: Approve a tool call only if the tool directly matches the user's intent and can produce the requested target output.
- Directness and parsimony: Prefer the minimal number of tool calls that directly yield the requested outputs.
- Completeness across multi-step or multi-item requests: If the user asks for multiple actions, all such calls must be present with appropriate arguments.
- Argument fidelity: Values must faithfully reflect the user's constraints. Prefer canonical/minimal forms.
- Thresholds and inequalities: Unless the tool documentation explicitly specifies strict inequality behavior, treat threshold-like parameters as inclusive bounds.
- Units and scales: For rates/percentages, prefer normalized proportions (e.g., 0.03 for 3%) unless the tool explicitly requires 0–100.
- Error criteria (mark error=true when any apply): Wrong or unnecessary tool used; missing required parameters; values that contradict user constraints; mis-scaled units; fabricating missing required values.

[Full prompt available in supplementary materials]

A.5 Latency Analysis

We analyze latency overhead of the feedback mechanism across both benchmarks. Table 9 presents summary statistics.

Benchmark	Config	Count	Mean	Median	P95
BFCL	Baseline	10,923	1.27s	0.79s	3.53s
	+Reviewer	3,641	7.87s	4.44s	22.1s
τ^2	Baseline	834	158.7s	116.2s	469.1s
	+Reviewer	834	384.3s	259.0s	988.0s

Table 9: Latency Statistics. BFCL measures per-item latency (seconds). τ^2-Bench measures per-episode duration (seconds).

Key Observations. On BFCL (single-turn function calling), the feedback mechanism increases average latency by 6.2×. This substantial increase occurs because the baseline is a single inference call, so reviewer overhead dominates. On τ^2-Bench (multi-turn agent episodes), the mechanism increases average duration by 2.4×. The lower relative impact reflects the multi-turn nature: reviewer overhead is amortized across approximately 40 turns per episode. Figure 5 shows the latency distributions.