Agentic Reasoning and Tool Integration for LLMs via Reinforcement Learning

Joykirat Singh, Yash Pandya, Pranav Vajreshwari, Raghav Magazine, Akshay Nambi
Microsoft Research
`corresponding author: akshayn@microsoft.com`

Abstract

Large language models (LLMs) have achieved remarkable progress in complex reasoning tasks, yet they remain fundamentally limited by their reliance on static internal knowledge and text-only reasoning. Real-world problem solving often demands dynamic, multi-step reasoning, adaptive decision making, and the ability to interact with external tools and environments. In this work, we introduce `ARTIST` (**A**gentic **R**easoning and **T**ool **I**ntegration in **S**elf-improving **T**ransformers), a unified framework that tightly couples agentic reasoning, reinforcement learning, and tool integration for LLMs. `ARTIST` enables models to autonomously decide *when*, *how*, and *which* tools to invoke within multi-turn reasoning chains, leveraging outcome-based RL to learn robust strategies for tool use and environment interaction without requiring step-level supervision. Extensive experiments on mathematical reasoning and multi-turn function calling benchmarks show that `ARTIST` consistently outperforms state-of-the-art baselines, with up to 20% absolute improvement over base models and strong gains on the most challenging tasks. Detailed studies and metric analyses reveal that agentic RL training leads to deeper reasoning, more effective tool use, and higher-quality solutions. Our results establish agentic RL with tool integration as a powerful new frontier for robust, interpretable, and generalizable problem-solving in LLMs.

1 Introduction

Large language models (LLMs) have achieved strong results in complex reasoning tasks [43, 17] due to advances in architecture, scale, and training with reinforcement learning (RL) [36], further improving reasoning by enabling models to refine strategies using outcome-based rewards [33, 52]. However, RL-enhanced LLMs remain limited by their reliance on internal knowledge and language modeling [41], which is problematic for time-sensitive or knowledge-intensive tasks where static knowledge may be outdated or incomplete [14]. These models also struggle with tasks requiring precise computation, structured manipulation, or specialized tool use, underscoring the need for integrating external information sources.

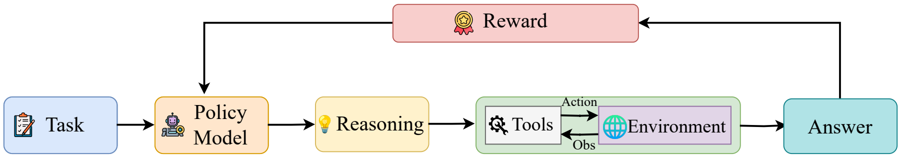

Figure 1: **The `ARTIST` architecture.** Agentic reasoning interleaves text-based thinking, tool queries, and outputs to coordinate reasoning, tool use, and environment interaction within a unified framework.

Agentic reasoning enables LLMs to interact dynamically with external resources such as web search, code execution, API calls, and structured memory and environments like web browsers or operating systems [45, 26, 50, 4]. This allows models to coordinate tool use and adaptively manage research, computation, and logical deduction [44, 30]. Many real-world tasks, including mathematical reasoning [22], multi-step derivations, and programmatic data analysis [49], require capabilities beyond language modeling. Frameworks like BFCLv3 [47] enable specialized function calls, while agentic environments such as WebArena support autonomous web interactions. In math, libraries like SymPy [21] provide greater accuracy and efficiency than text-based reasoning alone.

However, current tool-integration strategies face scalability and robustness challenges [35], often relying on hand-crafted prompts or heuristics that do not generalize to complex or unseen scenarios [19]. While prompting [10] and supervised fine-tuning [11, 25] can teach tool use, these methods are limited by curated data and often fail to adapt or recover from tool failures, highlighting the need for scalable, data-efficient, and adaptive tool-use frameworks.

We introduce `ARTIST` (Agentic Reasoning and Tool Integration in Self-Improving Transformers), a framework for agentic reasoning through tool integration. `ARTIST` enables LLMs to learn strategies for leveraging external tools and interacting with complex environments via reinforcement learning (see Figure 1). "Self-improving" refers to models that iteratively generate and learn from their own solutions, progressively tackling harder problems while maintaining the transformer architecture. Tool usage is interleaved within the reasoning chain, including active interaction with environments such as web browsers or operating systems. The process alternates between text-based thinking (`<think>...</think>`), tool queries (`<tool_name>...</tool_name>`), and tool outputs (`<output>...</output>`), enabling coordinated reasoning, tool use, and environment interaction.

This agentic structure allows the model to autonomously determine not only *which* tools to use, but also *when* and *how* to invoke them during multi-turn reasoning, adapting its strategy based on context and feedback from the environment and tool outputs. Tool results inform subsequent reasoning, forming a tightly coupled loop between inference and tool-augmented actions. `ARTIST` does not require supervision for intermediate steps or tool calls; instead, it uses reinforcement learning specifically, the GRPO [33] algorithm with only outcome-based rewards, enabling LLMs to develop adaptive and generalizable tool-use behaviors.

We evaluate `ARTIST` on complex mathematical problem solving and multi-turn function calling using benchmarks such as MATH-500 [13], AIME [1], AMC [2], Olympiad Bench [12], τ-bench [48], and BFCL v3 [47]. We benchmark `ARTIST` on 7B and 14B Qwen2.5-Instruct models [28] and compare against four categories of baselines: base models, reasoning-optimized models, open-source tool-augmented models, and prompt-based tool integration models, as well as frontier LLMs.

`ARTIST` achieves substantial performance gains with minimal training, requiring only about 500 steps and 5–10 hours of computation on 8xH100s, demonstrating efficient and scalable learning of robust agentic behaviors. Across all benchmarks, `ARTIST` consistently outperforms baselines, achieving up to **20% absolute improvement** over base models in mathematical reasoning. Notably, ARTIST with **7B models outperforms GPT-4o**, and when combined with reasoning-optimized models such as DeepSeek-Distilled-7B, ARTIST surpasses even frontier reasoning models like DeepSeek-R1 on challenging tasks such as MATH and Olympiad. For multi-turn function calling, `ARTIST` improves accuracy by **up to 8% on τ-bench and up to 16% on the most challenging BFCL v3** subsets compared to base and prompt-based models. Ablation and metric analyses confirm that agentic RL training in `ARTIST` leads to deeper reasoning, more effective tool use, and higher-quality solutions, with agentic reasoning behaviors such as adaptive tool selection, iterative self-correction, and context-aware multi-step reasoning observed across both domains.

Our key contributions are:

- **Unified agentic RL framework:** We present `ARTIST`, a framework that tightly couples agentic reasoning, reinforcement learning, and dynamic tool integration, enabling adaptive multi-turn problem solving across different classes of tools, models, and tasks.
- **Generalizable tool use and environment interaction:** `ARTIST` supports seamless integration with arbitrary external tools and environments, allowing LLMs to learn which tools to use, as well as when and how to invoke them within complex reasoning chains.
- **Extensive, rigorous evaluation:** We conduct comprehensive evaluation across mathematical and multi-turn function calling tasks, multiple benchmarks, model scales, and baseline categories, including detailed ablation and metric analyses.

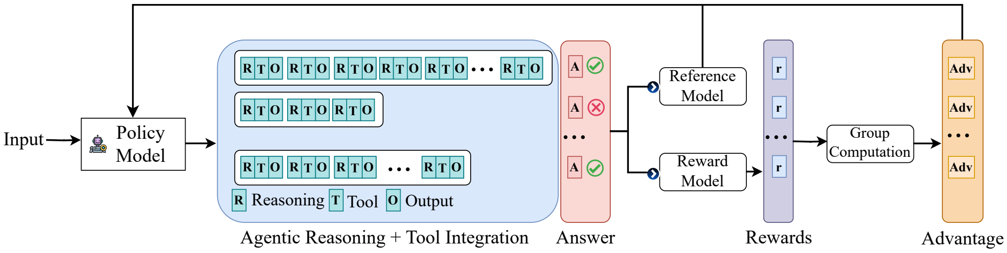

Figure 2: **Overview of the** `ARTIST` **methodology.** Reasoning rollouts alternate between internal thinking, tool use, and environment interaction.

2 `ARTIST` Overview

We present `ARTIST`, a general and extensible framework that enables LLMs to reason with and act upon external tools and environments via RL. Unlike prior methods [31, 10] that are limited to isolated tool use or narrow domains, `ARTIST` integrates diverse tools including code interpreters, web search, and APIs and supports interaction with environments such as web browsers and operating systems. This section describes the methodology, RL training, and reward modeling.

Methodology. `ARTIST` treats tool usage and environment interaction as core elements of the reasoning process. The LLM dynamically selects which tools or environments to engage, when to invoke them, and how to incorporate their outputs into multi-step solutions, supporting a wide range of real-world tasks. For each input, the policy model generates rollouts that alternate between text-based reasoning (`<think>...</think>`) and tool interactions. At each step, the model selects and invokes the appropriate tool based on context, incorporates the tool output, and updates its reasoning chain (see Figure 2). This iterative process enables flexible, adaptive strategies for complex tasks and supports self-correction and refinement through ongoing interaction with external resources.

2.1 Reinforcement Learning Algorithm

We train `ARTIST` with Group Relative Policy Optimization (GRPO), a sample-efficient RL algorithm that optimizes policies through groupwise outcome comparisons, eliminating the need for value function approximation and a separate critic network, thereby lowering training cost and simplifying optimization [33, 8]. In `ARTIST`, rollouts alternate between reasoning steps and tool outputs to capture interactions with external tools and environments. To ensure the model learns *when* and *how* to invoke tools rather than reproducing their outputs, tool tokens are masked during loss computation and only model-generated tokens are optimized. For instance, if Python code execution yields a deterministic result, updates apply to the reasoning and code generation steps but not to the returned output. This adaptation directs learning toward effective decision-making and tool selection. Full training details and the optimization objective are in Appendix A.

2.2 Rollouts in `ARTIST`

In `ARTIST`, rollouts alternate between internal reasoning and interaction with external tools or environments. `ARTIST` uses a structured prompt template with four segments: (1) internal reasoning (`<think>`...`</think>`), (2) tool or environment queries (`<tool_name>`...`</tool_name>`), (3) tool outputs (`<output>`...`</output>`), and (4) the final answer (`<answer>`...`</answer>`). After each tool query, the model invokes the tool or environment, appends the output, and continues reasoning until the answer is produced. The complete template is in Appendix B. At each step, the policy model decides whether to continue reasoning or interact with an external resource. Tool invocations may include code execution, API calls, web search, file operations, or actions in interactive environments. Tool outputs are incorporated into the reasoning chain, allowing for iterative refinement and adaptive strategy adjustment. Illustrative rollout examples are in Appendix C.

2.3 Applications

We illustrate the versatility of `ARTIST` on **mathematical reasoning** and **multi-turn function calling**, both *separately* and *jointly*, to test domain-specific performance and cross-task generalization. In all cases, `ARTIST` demonstrates adaptive tool use, robust error handling, and context-aware reasoning, guided by a structured reward design.

2.3.1 Complex Mathematical Reasoning with Agentic Tool Use

LLMs often struggle with mathematical problems requiring precise, multi-step calculations such as multiplying large numbers or evaluating definite integrals. `ARTIST` addresses this by integrating an external Python interpreter, allowing the model to offload complex computations and verify intermediate results programmatically.

Prompt Template. During rollouts, the model structures its output using tags: internal reasoning (`<think>`), code for execution (`<python>`), tool outputs (`<output>`), and the final answer (`<answer>`). The full prompt template is in Appendix B.

Reward Design. Three reward components guide RL training:

- **Answer Reward:** The model receives a reward of 2 if the final answer exactly matches the ground truth, and 0 otherwise: $R_{\text{answer}} = 2$ if $y_{\text{pred}} = y_{\text{ground}}$, 0 otherwise.
- **Format Reward:** To encourage structured outputs, we provide both relaxed and strict format rewards:
  - *Relaxed:* For each of the four required tag pairs (`<think>`, `<python>`, `<output>`, `<answer>`) present in the rollout, a reward of 0.125 is given, up to a maximum of 0.5.
  - *Strict:* An additional reward of 0.5 is awarded if (1) all tags are present, (2) the internal order of opening/closing tags is correct, and (3) the overall structure follows the sequence: `<think>` $\rightarrow$ `<python>` $\rightarrow$ `<output>` $\rightarrow$ `<answer>`.
- **Tool Execution Reward:** This reward is proportional to the fraction of successful Python code executions: $R_{\text{tool}} = \text{Tool}_{\text{success}}/\text{Tool}_{\text{total}}$.

2.3.2 Multi-Turn Function Calling with Agentic Reasoning and Tool Use

Function calling is essential for agentic LLMs, allowing models to invoke external functions for tasks such as information retrieval, deterministic operations, and workflow automation. This creates an interactive environment where the agent must coordinate multiple function calls, manage state, and engage with users over extended dialogues. For example, booking a flight involves sequential tool calls for checking availability, reserving a seat, and processing payment, each requiring state updates. Mastering when and how to issue these calls is critical for multi-turn reasoning. We evaluate `ARTIST` on BFCL v3 [47] and τ Bench [48], which assess long-context reasoning, user interaction, and cascaded function execution.

Prompt Template. The model is prompted to reason step by step using `<think>...</think>` tags, followed by function calls within `<tool_call>...</tool_call>` tags. The full prompt template is in Appendix B.

Reward Design. We use two reward components for function calling tasks:

- **State Reward:** Relaxed reward proportional to the fraction of correctly tracked state variables (e.g., working directory, selected files) across multi-turn interactions: $R_{\text{state}} = 0.5 \times \frac{State_{\text{match}}}{State_{\text{total}}}$
- **Function Reward:** Strict reward for correctly matching all function calls (by name and arguments) with the ground truth: $R_{\text{function}} = 0.5$ if all functions matched, and 0 otherwise

2.4 Example Analysis and Agentic Reasoning Behaviors

We present numerous examples across both applications and diverse settings in Appendix J and Appendix K. These examples illustrate how `ARTIST` decomposes complex math problems, alternates between reasoning and tool use, and adapts its strategy based on feedback. In multi-turn function calling, `ARTIST` interprets user requests, manages state, and recovers from tool errors. Such examples clarify the practical impact of agentic reasoning with tool use for real-world, multi-step tasks.

Across both domains, `ARTIST` consistently exhibits key agentic reasoning behaviors:

- **Self-Refinement:** The model incrementally revises its strategy, such as restructuring code in math or reordering actions in workflows—in response to intermediate results or changing requirements.

- **Self-Correction:** When encountering errors (e.g., tool execution failures, unmet preconditions), `ARTIST` diagnoses the issue and adapts its subsequent actions, often retrying with corrected input.
- **Self-Reflection:** At each step, the model explains its reasoning, summarizes the state, confirms details with the user, and validates outcomes through repeated computation or cross-verification.

These behaviors contribute to `ARTIST`'s strong performance and reliability on complex, real-world tasks, highlighting the value of unified agentic reasoning and tool integration.

3 Experimental Setup

Dataset and Evaluation Metrics. We evaluate `ARTIST` in two domains: complex mathematical problem solving and multi-turn function calling. For math, we train on 20,000 grade-school problems from **NuminaMath** [18] and evaluate on four challenging benchmarks (**MATH-500** [13], **AIME** [1], **AMC** [2], and **Olympiad Bench** [12]) that require advanced multi-step reasoning. For function calling, due to the lack of large real-world datasets, we use 100 realistic queries from **BFCL v3** [47] for training and 100 for validation, and evaluate on BFCL v3 subcategories (missing parameters, missing function, long context) and generalization to the new domain on **τ-bench** [48]. All models and baselines are evaluated with identical settings, and prompt templates are generic across datasets. We report Pass@1 accuracy for both domains, defined as the percentage of tasks where the model's final answer or environment state matches the ground truth. Additionally, we also train a model to learn tool calling for both domains together with a training dataset which combines both training datasets. Full dataset, and implementation details are in Appendix D and E[1].

Baselines. We benchmark `ARTIST` against a comprehensive set of baselines: frontier LLMs (`Frontier`), open-source tool-augmented LLMs (`Tool-OS`), base LLMs (`Base`), base LLMs with prompt-based tool integration (`Base-Prompt+Tools`), and reasoning-optimized LLMs (`Reasoning-Optimized`). This evaluation spans both complex mathematical reasoning and multi-turn function calling tasks. Additional details are provided in Appendix F.

4 Results

4.1 Results: Complex Math Reasoning

ARTIST vs. Base Models and Tool-Augmented Models. `ARTIST` achieves substantial improvements over both base and tool-augmented models (see Table 1). For Qwen2.5-7B, ARTIST improves over the base model by 12.9% (MATH), 13.8% (AIME), 19.2% (AMC), and 23.3% (Olympiad), with similar gains for Qwen2.5-14B and Phi 4. When compared to base models with Python tool integration, ARTIST achieves average improvements of 11.2% (Qwen2.5-7B) and 11.8% (Qwen2.5-14B). These gains are most pronounced on challenging datasets like AIME, AMC, and Olympiad, where agentic tool calling enables more effective multi-step reasoning and computation. Timely tool invocation and integration into the reasoning chain drive improvements of up to **20 percentage points** across benchmarks.

ARTIST vs. Reasoning Models and Tool-Augmented Reasoning Models. While reasoning-optimized models (e.g., DeepSeek-R1-Distill-Qwen-7B) are trained for strong stepwise thinking, ARTIST demonstrates that further gains are possible by learning optimal tool-calling policies within the reasoning process. ARTIST achieves an average improvement of **8.0% points** over DeepSeek-R1-Distill-Qwen-7B, with the largest gains on Olympiad and AMC. Notably, when tools are externally integrated with reasoning models (e.g., DeepSeek-R1-Distill-Qwen-7B + Python tool), performance drops sharply. This is likely due to the model's inability to reliably follow additional tool-use instructions, highlighting the importance of joint training for tool use and reasoning.

ARTIST vs. Frontier LLMs. ARTIST (Qwen2.5-7B and 14B) outperforms GPT-4o by 11.9% and 15.6% on MATH, 9.8% and 5.3% on AIME, 11.2% and 22.1% on AMC, and 15.3% and 16.4% on Olympiad, respectively. Against stronger reasoning models like DeepSeek-R1, ARTIST achieves comparable or better results on MATH and Olympiad, and narrows the gap on AIME and AMC. This demonstrates that agentic RL and tool integration can match or exceed the performance of much larger proprietary models, especially on the most complex tasks.

---

[1] Anonymous Code Repository: `https://anonymous.4open.science/r/AgenticReasoning/`

Method	MATH	AIME	AMC	Olymp.
Base Models				
Qwen2.5-7B-Ins	0.620	0.040	0.350	0.210
Qwen2.5-14B-Ins	0.700	0.060	0.330	0.240
Phi 4	0.760	0.090	0.460	0.290
Base Model + Tools				
Qwen2.5-7B-Ins+Py	0.629	0.122	0.349	0.366
Qwen2.5-14B-Ins+ Py	0.671	0.10	0.41	0.371
Phi 4+py	0.752	0.156	0.481	0.451
ARTIST (Base Model)				
Qwen2.5-7B+ `ARTIST`	**0.749**	**0.178**	**0.542**	**0.443**
Qwen2.5-14B+ `ARTIST`	**0.786**	**0.133**	**0.651**	**0.454**
Phi 4+ `ARTIST`	**0.786**	**0.111**	**0.578**	**0.455**
Ablations (Base Model)				
Qwen2.5-7B (Plain GRPO)	0.756	0.089	0.470	0.406
Qwen2.5-7B (Conditional Reward)	0.738	0.055	0.410	0.390
Qwen2.5-7B (No Tool Reward)	0.754	0.144	0.446	0.423

Method	MATH	AIME	AMC	Olymp.
Frontier LLMs				
GPT4o	0.630	0.080	0.430	0.290
DeepSeek-R1	0.850	0.300	0.810	0.460
Open Source Model + Tool Integrated				
NuminaMath-TIR-7B	0.530	0.060	0.240	0.190
ToRA Code-7B	0.460	0.000	0.100	0.160
Qwen 7B (PAL)	0.100	0.000	0.050	0.020
Reasoning Model				
DeepSeek-R1-Distill-Qwen-7B	0.858	0.211	0.675	0.395
Reasoning Model + Tools				
DeepSeek-R1-Distill-Qwen-7B + python tool	0.230	0.044	0.108	0.138
ARTIST (Reasoning Models)				
DeepSeek Qwen 7B-Instruct + `ARTIST`	**0.880**	**0.267**	**0.735**	**0.577**

Table 1: Pass@1 accuracy. Comparison of base models (left) and reasoning models (right) with `ARTIST`. `ARTIST` consistently outperforms all baselines, especially on complex math tasks.

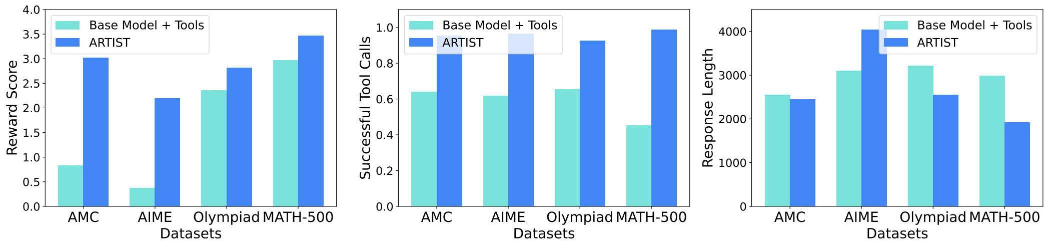

Figure 3: Average reward score, Successful Tool call and the response length metric across all math datasets for Qwen2.5-7B-Instruct model (`ARTIST` vs. `Base-Prompt+Tools`) .

Summary: ARTIST achieves the highest Pass@1 accuracy across all mathematical reasoning benchmarks, with average absolute gains of 20% over base models, 12% over base+tools, 8% over DeepSeek-R1-Distill, and up to 12% over DeepSeek-R1 on the most challenging tasks.

4.1.1 Ablations

We assess the contribution of each reward component in `ARTIST` by comparing the full model to three ablations on Qwen2.5-7B: (1) plain GRPO (no tool use), (2) conditional reward (only correct final answers are rewarded), and (3) no tool reward (tool rewards always zero). Across all benchmarks, the full `ARTIST` model achieves the highest accuracy, with the largest gains on challenging datasets. On AMC, `ARTIST` outperforms plain GRPO, conditional reward, and no tool reward ablations by 7.2%, 13.2%, and 9.6%, respectively; similar trends hold for AIME and Olympiad, while differences on MATH-500 are less than 2%. These results show that tool integration, fine-grained intermediate rewards, and explicit tool execution incentives are each critical for robust agentic reasoning and tool use, especially for complex, multi-step tasks.

4.1.2 Effectiveness of `ARTIST`

We evaluate `ARTIST` on three metrics: (1) **Reward Score** for solution quality, (2) **Successful Tool Calls** for tool utilization, and (3) **Response Length** for reasoning depth. Figure 3 compares `ARTIST` and `Base-Prompt+Tools` on all math datasets for Qwen2.5-7B-instruct, with all metrics computed under identical evaluation settings. Further analyses for other models and training dynamics are available in Appendix G and H.

Reward Score (Solution Quality) `ARTIST` substantially improves solution quality, raising the reward score from 0.8 to 3.0 on AMC and from 0.37 to 2.19 on AIME. Similar gains are seen on Olympiad

Benchmark **Method**	**τ bench** Airline	 Retail	 Avg.	**BFCL v3** Missing Function	 Missing Parameters	 Long Context	 Avg.
Frontier LLMs							
GPT-4o	0.420	0.604	0.510	0.440	0.375	0.580	0.465
Llama-3-70B*	0.148	0.144	0.146	0.130	0.105	0.095	0.110
Deepseek-R1	–	–		0.155	0.110	0.115	0.126
Base LLMs + Reasoning via Prompt							
Qwen2.5-7B-Ins (Base)	0.120	0.180	0.15	0.085	0.060	0.040	0.061
Qwen2.5-7B-Ins (Prompt)	0.120	0.200	0.160	0.105	0.055	0.055	0.071
Reasoning-optimized Model							
Qwen3-4B	0.260	0.300	0.280	0.240	0.210	0.175	0.208
`ARTIST`							
Qwen2.5-7B-Ins + `ARTIST`	**0.180**	**0.210**	**0.195**	**0.205**	**0.235**	**0.230**	**0.223**
Qwen3-4B + `ARTIST`	**0.300**	**0.410**	**0.355**	**0.390**	**0.285**	**0.210**	**0.295**

Table 2: Pass@1 accuracy on five multi-turn multi-step function calling benchmarks. `ARTIST` consistently outperforms baselines, especially on complex tasks.

(2.35 to 2.8) and MATH-500 (2.9 to 3.5), indicating more correct and complete solutions, especially on complex problems.

Successful Tool Calls (Successful tool Utilization) `ARTIST` achieves a much higher rate of successful tool calls, 95.5% on average versus 58.5% for `Base-Prompt+Tools`, a 37-point increase. This consistent improvement shows that `ARTIST` reliably learns effective tool use, which is essential for robust multi-step reasoning.

Response Length (Reasoning depth) While `Base-Prompt+Tools` produces slightly longer responses (2961 vs. 2740 tokens), `ARTIST` adapts response length to task complexity, generating longer outputs for harder tasks (e.g., AIME) and shorter, focused responses for others. This efficiency comes from delegating complex computation to Python code, reducing unnecessary language output and improving clarity.

4.2 Results: Multi-Turn Function Calling

We evaluate `ARTIST` on five multi-turn, multi-step function calling benchmarks, comparing its performance to a comprehensive set of baselines (see Table 2). All models are evaluated under identical settings with generic prompt templates. Our evaluation emphasizes generalization: models are trained on a subset of BFCL v3 and tested on the distinct τ-bench domain, which uses a different tool call syntax. This demonstrates `ARTIST`'s ability to generalize across both domains and tool formats.

ARTIST vs. Base and Reasoning via Prompt on Base Models. `ARTIST` achieves substantial improvements over both base and prompt-based reasoning models. Qwen2.5-7B-Ins + `ARTIST` improves average accuracy by 4.5% points on τ-bench (from 15.0% to 19.5%) and by 16.2% on BFCL v3 (from 6.1% to 22.3%) compared to the base model. Similar gains are observed over the variant with prompt nudging Base Model to use reasoning tokens, confirming that training agentic tool integration and adaptive reasoning using RL is critical for multi-turn workflows.

ARTIST vs. Reasoning-Optimized and Open-Source LLMs. Compared to the reasoning-optimized Qwen3-4B, ARTIST delivers gains of 7.5% on τ-bench (from 28.0% to 35.5%) and 8.7% on BFCL v3 (from 20.8% to 29.5%). Qwen3-4B + `ARTIST` achieves the highest scores among all open-source models, outperforming Llama-3-70B and Deepseek-R1 by wide margins.

ARTIST vs. Frontier LLMs. While GPT-4o remains the strongest overall on τ-bench (51.0%), ARTIST with Qwen3-4B achieves competitive performance and narrows the gap on BFCL v3 (29.5% vs. 46.5%), outperforming all other open-source baselines. Notably, ARTIST's improvements are most pronounced on the more complex BFCL v3 subcategories, where it outperforms all non-GPT-4o baselines by 8–18 percentage points.

Summary: With training on a small set of function calling traces, `ARTIST` generalizes effectively to new domains and tasks. On both Qwen2.5-7B and Qwen3-4B models, ARTIST achieves average gains of 4–8% on τ-bench and 9-16% on BFCL v3 over their respective baselines, demonstrating robust cross-domain generalization and substantial improvements in multi-turn function calling.

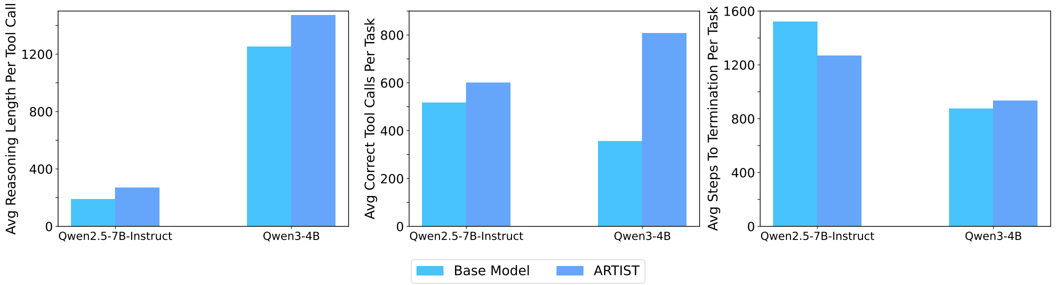

Figure 4: Metrics Analysis for Multi-Turn Function Calling on τ-bench.

4.2.1 Effectiveness of `ARTIST` for Multi-Turn Function Calling

We assess effectiveness of `ARTIST` on τ-bench using three metrics: reasoning length per tool call, total correct tool calls, and total steps to task completion. Figure 4 compares Qwen2.5-7B and Qwen3-4B models, both with and without `ARTIST`.

- **Reasoning Length per Tool Call:** For Qwen2.5-7B, `ARTIST` increases reasoning length per tool call from 190 to 271 tokens (43% gain), and for Qwen3-4B, from 1251 to 1472 tokens (18% gain). This indicates that `ARTIST` enables the model to generate richer, more context-aware reasoning before invoking tools, reflecting deeper understanding and more deliberate tool use.
- **Total Correct Tool Calls:** `ARTIST` increases the number of correct tool calls from 517 to 600 (16% gain) for Qwen2.5-7B, and from 356 to 808 (127% gain) for Qwen3-4B. This demonstrates that `ARTIST` not only encourages more frequent tool use, but also ensures that tool calls are accurate and relevant, directly contributing to higher task completion rates.
- **Total Steps to Completion:** Despite making more tool calls, `ARTIST` reduces the average steps to completion: from 1520 to 1270 (16% reduction) for Qwen2.5-7B, while mildly increasing it from 876 to 935 for Qwen3-4B. This efficiency arises because `ARTIST` learns to make the right tool calls at the right time, minimizing unnecessary interactions and converging to solutions more quickly.

Our training runs remain stable over many epochs, with both training and validation scores improving rapidly and saturating after 350 steps, despite using only 100 tasks. No overfitting is observed, even after 90 epochs. This stability is attributed to the diversity of multi-step rollouts. Total response length of an interaction/rollout is consistent throughout training, with only minor variations. Additional details are provided in the Appendix I.

4.3 Results: Unified Tool Calling

We also trained a single model on both mathematical problem solving (via a Python code execution tool) and multi-turn function calling (via the BFCL toolset). The model was given access to both tool types as needed, using a unified prompt template and a consistent reward structure across domains.

Table 3 reports results on Phi-4. Unified ARTIST delivers performance that is consistently on par with or better than training models separately, highlighted by a substantial 150% improvement on AIME (0.111 → 0.278).

	AMC	AIME	Olympiad	
Phi-4 + Python	0.578	0.111	0.455	
Phi-4 + Unified	0.542	0.278	0.451	
	Long context	Miss Param	Miss Func	Avg
Phi-4 + Function-call	0.340	0.380	0.480	0.400
Phi-4 + Unified	0.470	0.335	0.345	0.383

Table 3: Comparison with separate and Unified training across diverse tasks.

Importantly, the unified model learns to invoke the correct tool for the correct task based on context, achieving performance close to (and sometimes exceeding) training with separate models. This validates the effectiveness of our unified prompt and reward design, and highlights ARTIST's advantage over frameworks that rely on task- or tool-specific engineering.

5 Related Work

The reasoning space for LLMs broadly encompasses: (1) chain-of-thought (CoT) prompting, (2) tool-based reasoning, and (3) RL for reasoning and alignment. While each area has driven progress, their intersection, agentic reasoning with dynamic tool integration via RL remains underexplored.

Chain-of-Thought Reasoning and Prompting Approaches Chain-of-thought (CoT) Chain-of-thought (CoT) prompting [43, 40, 5] improves LLM accuracy and interpretability by encouraging explicit intermediate steps. However, CoT approaches rely on the model's internal knowledge and language generation, which limits their effectiveness for tasks requiring external information, precise computation, or real-world action [38].

Tool-Based Reasoning Tool-based reasoning [15, 51] extends LLMs by enabling interaction with external resources such as web search, code interpreters, and APIs. Methods like PAL [10], ART [25], ToRA [11], and Toolformer [31] demonstrate the potential of tool integration. However, these approaches often depend on high-quality labeled trajectories or handcrafted prompts, which limit scalability and adaptability. Tool use is frequently decoupled from the reasoning process, especially during training, reducing its effectiveness in dynamic or multi-turn scenarios.

RL-Based Reasoning Reinforcement learning (RL) [16] has become a key technique for aligning LLMs with desired reasoning behaviors. RLHF [24], PPO [32], DPO [29], and SimPO [20] have improved LLM alignment and stability. Group Relative Policy Optimization (GRPO) [33] advances RL-based reasoning by eliminating value functions and using groupwise outcome rewards, as shown in DeepSeekMath and DeepSeek-R1 [8]. These methods foster robust, self-corrective chain-of-thought behaviors [7].

Reasoning and Agentic Systems Recent work has begun to address agentic reasoning, where LLMs autonomously plan, adapt, and interact with external environments [27, 44]. Concurrently with our work, a few studies have explored tool integration within the reasoning process. R1-Searcher [34] and ReSearch [6] incorporate search tools during reasoning, mainly for retrieval-augmented generation (RAG) tasks. Retool [9] uses a code interpreter for math problems but depends on supervised fine-tuning with annotated data. RAGEN [42] proposes StarPO (State-Thinking-Actions-Reward Policy Optimization), a general framework for trajectory-level agent RL, and introduces RAGEN, a modular system for training and evaluating LLM agents. They focus on methods to make multi-turn RL training more robust.

Importance and Contribution In contrast, our approach applies outcome-based RL without step-level supervision to more complex, real-world agentic scenarios that require dynamic tool use, multi-turn reasoning, and adaptive environment interaction. Prior work typically isolates tool use from reasoning or focuses RL on internal reasoning alone or do not test on complex real-world scenarios. This leaves a gap in methods that jointly optimize agentic reasoning and tool integration for dynamic, real-world tasks, an area directly addressed by our framework. Remarkably, `ARTIST` achieves significant performance improvements with minimal training requiring only around 500 steps and 5–10 hours of computation demonstrating efficient and scalable learning of robust, generalizable agentic behaviors.

6 Conclusions

This paper introduces `ARTIST`, a unified framework that combines agentic reasoning, reinforcement learning, and dynamic tool use to enhance the capabilities of LLMs. By treating tool use and environment interaction as core parts of the reasoning process, `ARTIST` enables models to plan, adapt, and solve complex, multi-step tasks with minimal supervision. Our experiments on mathematical and multi-turn function-calling benchmarks show that `ARTIST` not only improves accuracy but also leads to richer behaviors, such as adaptive tool choice, iterative self-correction, and deeper multi-step reasoning. These abilities emerge without step-level guidance, highlighting the effectiveness of reinforcement learning in training models to navigate complex environments. Beyond performance, we find that models trained with `ARTIST` produce more interpretable and robust reasoning traces. They use tools in a context-aware, purposeful way, shifting away from static prompting toward truly autonomous interaction with their environment.

Limitations and Future Work. Scaling `ARTIST` to longer tasks is challenged by memory and context length constraints. Adapting to new domains often requires custom environments, and the lack of standardized, high-quality benchmarks limits training and evaluation. `ARTIST` also relies on outcome-based rewards and ground-truth answers, restricting use in settings without such supervision. Future work should enable training in unsupervised or preference-based scenarios, expand to more domains, and incorporate richer feedback, including human preferences. Addressing safety and reliability in open-ended environments is also crucial. Progress in these areas may advance agentic reasoning, supporting more adaptive and trustworthy AI systems.

References

[1] AI-MO/aimo-validation-aime · Datasets at Hugging Face — huggingface.co. `https://huggingface.co/datasets/AI-MO/aimo-validation-aime`. [Accessed 16-02-2025].

[2] AI-MO/aimo-validation-amc · Datasets at Hugging Face — huggingface.co. `https://huggingface.co/datasets/AI-MO/aimo-validation-amc`. [Accessed 16-02-2025].

[3] Edward Beeching, Shengyi Costa Huang, Albert Jiang, Jia Li, Benjamin Lipkin, Zihan Qina, Kashif Rasul, Ziju Shen, Roman Soletskyi, and Lewis Tunstall. Numinamath 7b tir. `https://huggingface.co/AI-MO/NuminaMath-7B-TIR`, 2024.

[4] Rogerio Bonatti, Dan Zhao, Francesco Bonacci, Dillon Dupont, Sara Abdali, Yinheng Li, Yadong Lu, Justin Wagle, Kazuhito Koishida, Arthur Bucker, Lawrence Jang, and Zack Hui. Windows agent arena: Evaluating multi-modal os agents at scale, 2024.

[5] Zefan Cai, Baobao Chang, and Wenjuan Han. Human-in-the-loop through chain-of-thought, 2023.

[6] Mingyang Chen, Tianpeng Li, Haoze Sun, Yijie Zhou, Chenzheng Zhu, Haofen Wang, Jeff Z. Pan, Wen Zhang, Huajun Chen, Fan Yang, Zenan Zhou, and Weipeng Chen. Research: Learning to reason with search for llms via reinforcement learning, 2025.

[7] Alan Dao and Dinh Bach Vu. Alphamaze: Enhancing large language models' spatial intelligence via grpo, 2025.

[8] DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jiawei Wang, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang Zhou, Shaoqing Wu, Shengfeng Ye, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wanjia Zhao, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, W. L. Xiao, Wei An, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanhong Xu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Ying Tang, Yukun Zha, Yuting Yan, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, and Zhen Zhang. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025.

[9] Jiazhan Feng, Shijue Huang, Xingwei Qu, Ge Zhang, Yujia Qin, Baoquan Zhong, Chengquan Jiang, Jinxin Chi, and Wanjun Zhong. Retool: Reinforcement learning for strategic tool use in llms, 2025.

[10] Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. Pal: Program-aided language models, 2023.

[11] Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Minlie Huang, Nan Duan, and Weizhu Chen. Tora: A tool-integrated reasoning agent for mathematical problem solving, 2024.

[12] Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu, Zhen Leng Thai, Junhao Shen, Jinyi Hu, Xu Han, Yujie Huang, Yuxiang Zhang, Jie Liu, Lei Qi, Zhiyuan Liu, and Maosong Sun. Olympiadbench: A challenging benchmark for promoting agi with olympiad-level bilingual multimodal scientific problems, 2024.

[13] Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021.

[14] Peng Hu, Changjiang Gao, Ruiqi Gao, Jiajun Chen, and Shujian Huang. Large language models are limited in out-of-context knowledge reasoning, 2024.

[15] Tatsuro Inaba, Hirokazu Kiyomaru, Fei Cheng, and Sadao Kurohashi. Multitool-cot: Gpt-3 can use multiple external tools with chain of thought prompting, 2023.

[16] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285, 1996.

[17] Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners, 2023.

[18] Jia LI, Edward Beeching, Lewis Tunstall, Ben Lipkin, Roman Soletskyi, Shengyi Costa Huang, Kashif Rasul, Longhui Yu, Albert Jiang, Ziju Shen, Zihan Qin, Bin Dong, Li Zhou, Yann Fleureau, Guillaume Lample, and Stanislas Polu. Numinamath. `[https://huggingface.co/AI-MO/NuminaMath-CoT](https://github.com/project-numina/aimo-progress-prize/blob/main/report/numina_dataset.pdf)`, 2024.

[19] Ne Luo, Aryo Pradipta Gema, Xuanli He, Emile van Krieken, Pietro Lesci, and Pasquale Minervini. Self-training large language models for tool-use without demonstrations, 2025.

[20] Yu Meng, Mengzhou Xia, and Danqi Chen. Simpo: Simple preference optimization with a reference-free reward, 2024.

[21] Aaron Meurer, Christopher P Smith, Mateusz Paprocki, Ondřej Čertík, Sergey B Kirpichev, Matthew Rocklin, AMiT Kumar, Sergiu Ivanov, Jason K Moore, Sartaj Singh, et al. Sympy: symbolic computing in python. *PeerJ Computer Science*, 3:e103, 2017.

[22] Spyridon Mouselinos, Henryk Michalewski, and Mateusz Malinowski. Beyond lines and circles: Unveiling the geometric reasoning gap in large language models, 2024.

[23] et. al OpenAI. Gpt-4o system card, 2024.

[24] Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback, 2022.

[25] Bhargavi Paranjape, Scott Lundberg, Sameer Singh, Hannaneh Hajishirzi, Luke Zettlemoyer, and Marco Tulio Ribeiro. Art: Automatic multi-step reasoning and tool-use for large language models, 2023.

[26] Avinash Patil. Advancing reasoning in large language models: Promising methods and approaches, 2025.

[27] Aske Plaat, Max van Duijn, Niki van Stein, Mike Preuss, Peter van der Putten, and Kees Joost Batenburg. Agentic large language models, a survey, 2025.

[28] 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, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report, 2025.

[29] Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D. Manning, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model, 2024.

[30] Sumedh Rasal and E. J. Hauer. Navigating complexity: Orchestrated problem solving with multi-agent llms, 2024.

[31] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools, 2023.

[32] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

[33] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models, 2024.

[34] Huatong Song, Jinhao Jiang, Yingqian Min, Jie Chen, Zhipeng Chen, Wayne Xin Zhao, Lei Fang, and Ji-Rong Wen. R1-searcher: Incentivizing the search capability in llms via reinforcement learning, 2025.

[35] Jimin Sun, So Yeon Min, Yingshan Chang, and Yonatan Bisk. Tools fail: Detecting silent errors in faulty tools, 2024.

[36] Richard S Sutton, Andrew G Barto, et al. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

[37] Qwen Team. Qwen3, April 2025.

[38] Miles Turpin, Julian Michael, Ethan Perez, and Samuel R. Bowman. Language models don't always say what they think: Unfaithful explanations in chain-of-thought prompting, 2023.

[39] verl. volcengine/verl: verl: Volcano Engine Reinforcement Learning for LLMs — github.com. `https://github.com/volcengine/verl`, 2024. [Accessed 28-04-2025].

[40] Lei Wang, Wanyu Xu, Yihuai Lan, Zhiqiang Hu, Yunshi Lan, Roy Ka-Wei Lee, and Ee-Peng Lim. Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models, 2023.

[41] Yifei Wang, Yuheng Chen, Wanting Wen, Yu Sheng, Linjing Li, and Daniel Dajun Zeng. Unveiling factual recall behaviors of large language models through knowledge neurons, 2024.

[42] Zihan Wang, Kangrui Wang, Qineng Wang, Pingyue Zhang, Linjie Li, Zhengyuan Yang, Xing Jin, Kefan Yu, Minh Nhat Nguyen, Licheng Liu, Eli Gottlieb, Yiping Lu, Kyunghyun Cho, Jiajun Wu, Li Fei-Fei, Lijuan Wang, Yejin Choi, and Manling Li. Ragen: Understanding self-evolution in llm agents via multi-turn reinforcement learning, 2025.

[43] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models, 2023.

[44] Junde Wu, Jiayuan Zhu, and Yuyuan Liu. Agentic reasoning: Reasoning llms with tools for the deep research, 2025.

[45] Wei Xiong, Chengshuai Shi, Jiaming Shen, Aviv Rosenberg, Zhen Qin, Daniele Calandriello, Misha Khalman, Rishabh Joshi, Bilal Piot, Mohammad Saleh, Chi Jin, Tong Zhang, and Tianqi Liu. Building math agents with multi-turn iterative preference learning, 2025.

[46] Fengli Xu, Qianyue Hao, Zefang Zong, Jingwei Wang, Yunke Zhang, Jingyi Wang, Xiaochong Lan, Jiahui Gong, Tianjian Ouyang, Fanjin Meng, Chenyang Shao, Yuwei Yan, Qinglong Yang, Yiwen Song, Sijian Ren, Xinyuan Hu, Yu Li, Jie Feng, Chen Gao, and Yong Li. Towards large reasoning models: A survey of reinforced reasoning with large language models, 2025.

[47] Fanjia Yan, Huanzhi Mao, Charlie Cheng-Jie Ji, Tianjun Zhang, Shishir G Patil, Ion Stoica, and Joseph E Gonzalez. Berkeley function calling leaderboard, 2024.

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

[49] Yilun Zhao, Hongjun Liu, Yitao Long, Rui Zhang, Chen Zhao, and Arman Cohan. Financemath: Knowledge-intensive math reasoning in finance domains, 2024.

[50] Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. Webarena: A realistic web environment for building autonomous agents, 2024.

[51] Yuchen Zhuang, Xiang Chen, Tong Yu, Saayan Mitra, Victor Bursztyn, Ryan A. Rossi, Somdeb Sarkhel, and Chao Zhang. Toolchain*: Efficient action space navigation in large language models with a* search, 2023.

[52] Yuxin Zuo, Kaiyan Zhang, Shang Qu, Li Sheng, Xuekai Zhu, Biqing Qi, Youbang Sun, Ganqu Cui, Ning Ding, and Bowen Zhou. Ttrl: Test-time reinforcement learning, 2025.

A `ARTIST` Training Algorithm with GRPO

Effective training of agentic LLMs with tool and environment integration requires an RL algorithm that is sample-efficient and stable with outcome-based rewards. Group Relative Policy Optimization (GRPO) [33], as used in DeepSeek-R1 [8], meets these requirements by leveraging groupwise outcome rewards and eliminating the need for value function approximation, thus reducing training cost and simplifying optimization.

Group Relative Policy Optimization. GRPO extends Proximal Policy Optimization (PPO) [32] by removing the critic and estimating the baseline from a group of sampled responses. For each question q, a group $\{y_1, ..., y_G\}$ is sampled from the old policy π_{old}, and the policy π_θ is optimized by maximizing the following objective:

$$\begin{aligned}\mathcal{J}_{\text{GRPO}}(\theta) = \mathbb{E}_{x\sim\mathcal{D},\{y_i\}_{i=1}^{G}\sim\pi_{\text{old}}(\cdot\mid x;\mathcal{R})}\Bigg[\frac{1}{G}\sum_{i=1}^{G}\frac{1}{\sum_{t=1}^{|y_i|}\mathbb{I}(y_{i,t})}\sum_{\substack{t=1\\(y_{i,t}=1)}}^{|y_i|}\min\left(\frac{\pi_\theta(y_{i,t}\mid x,y_{i,<t};\mathcal{R})}{\pi_{\text{old}}(y_{i,t}\mid x,y_{i,<t};\mathcal{R})}\hat{A}_{i,t},\right.\\ \left.\text{clip}\left(\frac{\pi_\theta(y_{i,t}\mid x,y_{i,<t};\mathcal{R})}{\pi_{\text{old}}(y_{i,t}\mid x,y_{i,<t};\mathcal{R})},1-\epsilon,1+\epsilon\right)\hat{A}_{i,t}\right)-\beta\mathbb{D}_{\text{KL}}[\pi_\theta\|\pi_{\text{ref}}]\Bigg]\end{aligned} \tag{1}$$

where ϵ and β are hyperparameters, and $\hat{A}_{i,t}$ represents the advantage, computed based on the relative rewards of outputs within each group.

The complete training procedure for `ARTIST` with GRPO is summarized in Algorithm 1.

Algorithm 1 Training `ARTIST` with Group Relative Policy Optimization (GRPO)

```
Require: Policy model $\pi_\theta$, old policy $\pi_{\text{old}}$, task dataset $\mathcal{D}$, group size $G$, masking function $\mathcal{M}$
for each training iteration do
    for each task $q$ in batch do
        Sample $G$ rollouts $\{y_1, \dots, y_G\}$ from $\pi_{\text{old}}$:
        for each rollout $y_i$ do
            Initialize reasoning chain
            while not end of episode do
                Generate next segment: <think> or <tool_name>
                if tool is invoked then
                    Interact with environment, obtain <output>
                    Append output to reasoning chain
                end if
            end while
            Compute outcome reward $R(y_i)$
        end for
        Compute groupwise advantages $\hat{A}_{i,t}$ for all $y_i$
        Compute importance weights $r_{i,t}$
        Apply loss masking $\mathcal{M}$ to exclude tool output tokens
        Compute GRPO loss $\mathcal{L}_{\text{GRPO}}$ and update $\pi_\theta$
    end for
end for
```

B Prompt Templates in `ARTIST`

B.1 Prompt Template for Complex Math Reasoning

```
You are a helpful assistant that can solve complex math problems step
    by step with the help of a python executor tool. Given a question,
     you need to first think about the reasoning process in the mind
    and then provide the answer. During thinking, you can write python
     code, and invoke python tool to execute the code and get back the
     output of the code. The reasoning process and answer are enclosed
     within <think> </think> and <answer> </answer> tags respectively,
     and the python code and the output are enclosed within <python>
    </python> and <output> </output> tags respectively. You can
    utilize the Sympy library to write the python code and make sure
    to print the result at the end of the python code. You can utilize
     the python tool as many times as required, however each python
    code will be executed separately. For example, <think> reasoning
    process here </think> <python> python code here </python> <output>
     output of python code here </output> <think> reasoning process
    here </think> <answer> final answer here </answer>.
```

B.2 Prompt Template for Multi-turn Function Calling

```
You are an expert in composing functions. You are given a question
    from a user and a set of possible functions. Based on the question
    , you will need to make one or more function/tool calls to
    complete the task.
You have access to the following tools to help solve the task:
{tools}

[Classes Involved: {classes_involved}]

For each step:
1. Start with a step-by-step reasoning process inside <think> </think>
     tags to think through the problem.
2. If needed, use tools by writing one or more function call commands
    as a list inside <tool_call> </tool_call> tags. Each item in the
    list should follow the format shared in the example below.
   example: <tool_call> [func_name1(params_name1=params_value1,
       params_name2=params_value2), func_name2(params)] </tool_call>
   Tools expect specific input formats. Do not make up tools or
       arguments that aren't listed.
3. After you have used the tools, you will see the tool outputs inside
     <tool_result> </tool_result> tags in the same order from the
    system.
4. If you believe the current task is completed and no more tool,
    summarize your progresses and output <TASK_FINISHED> in the end of
     your response to terminate the conversation.
5. Otherwise if you believe the task is not able to be completed,
    summarize what is problematic and output <TASK_ERROR> in the end
    of your response to terminate the conversation.

At each turn, you should try your best to complete the tasks requested
     by the user within the current turn. Continue to output functions
     to call until you have fulfilled the user's request to the best
    of your ability. Once you have no more functions to call, the
    system will consider the current turn complete and proceed to the
    next turn or task.
```

B.3 Prompt Template for Unified Tool Calling

```
You are a reasoning language model that can reach precise answers
    through careful reasoning and tool use when needed.

Structure Rules:
1. All reasoning goes between <think> and </think> (thinking block).
2. Within the thinking block, whenever a tool would improve your
    answer, invoke it using <tool_call>...</tool_call> instead of
    relying solely on memory.
3. Issue one valid <tool_call>...</tool_call> at a time; further tool
    calls can be sequentially interleaved throughout the reasoning
    process.
4. After each tool call, the result of the tool call will be provided
    in the <tool_result>...</tool_result> tags.
```

```
5. Provide the final answer for the user inside the <answer> </answer>
    tags.
6. Stop the generation only after reaching the final answer.

You can utilize the tools as many times as required. For example, <
    think> reasoning here  </think> <tool_call> tool call here </
    tool_call> <tool_result> output of tool call </tool_result> <think
    > reasoning process here </think> <answer> final answer here </
    answer>.

# RESPONSE FORMAT FOR TOOL CALLS

{response_format}

# AVAILABLE TOOLS

{tool_details}
```

Response Format for Complex Math Reasoning Tasks (Markdown)

~~~
Python code should be in markdown format. Format: <tool_call>
```python
{code here}
```
</tool_call>
~~~

Response Format for Multi-turn Function Calling (JSON)

```
Response Format for tool call: <tool_call>{"name":"<tool-name>","
    arguments":"<json-string-of-parameters>"}</tool_call>'
```

C Potential Examples of Rollouts with Agentic Reasoning

We provide three concrete potential examples to illustrate the diversity and flexibility of rollouts with agentic reasoning:

1. Math Problem with Python and Web Search

Task: Evaluate the definite integral $\int_0^1 \frac{\ln(1+x)}{x} dx$.

How `ARTIST` solves it: The model first searches the web for known results, then verifies the answer by invoking a Python code interpreter with SymPy for symbolic computation, integrating both external sources into its reasoning before producing the final answer.

```
<think>To solve the integral \(\int_0^1 \frac{\ln(1+x)}{x} \, dx\), I
    recall it may not have a simple closed form. Lets check online.</
    think>
<search>integral of ln(1+x)/x from 0 to 1</search>
<output>The integral is \(\frac{\pi^2}{12}\).</output>
<think>Lets verify this using symbolic computation.</think>
<python>
from sympy import symbols, integrate, log

x = symbols('x')
result = integrate(log(1+x)/x, (x, 0, 1))
print(result)
</python>
<output>\(\frac{\pi^2}{12}\)</output>
<think>Both sources confirm the result: \(\frac{\pi^2}{12}\).</think>
<answer>The value of the integral is \(\frac{\pi^2}{12}\).</answer>
```

2. Web Navigation in WebArena

Task: Find the contact email for the support team on the company's website.

How `ARTIST` *solves it:* The model navigates the website by issuing browser actions, interprets the environment's feedback, and extracts the required information to answer the task.

```
<think> I need to find the support email on the company’s website. </
    think>
<browser> open_url(’https://company.com’) </browser>
<output> Homepage loaded. Links: About, Products, Support, Contact. </
    output>
<think> Click on the ’Support’ link. </think>
<browser> click_link(’Support’) </browser>
<output> Support page loaded. Email: support@company.com </output>
<think> Found the support email. </think>
<answer> support@company.com </answer>
```

These examples demonstrate how `ARTIST` enables LLMs to flexibly coordinate internal reasoning with dynamic tool and environment interaction, supporting complex, multi-step problem solving across diverse domains. By structuring rollouts in this way, the model can leverage external resources, adapt its strategy based on feedback, and produce robust, verifiable solutions.

D Dataset and Evaluation Metrics

We evaluate `ARTIST` in two domains: complex mathematical problem solving and multi-turn function calling. For each, we detail the training and evaluation datasets, metrics, and experimental protocol. All evaluation tasks are strictly held out from training, and prompt templates are generic and consistent across datasets to avoid overfitting.

D.1 Complex Mathematical Reasoning

Training Dataset We curate a training set of 20,000 math word problems, primarily sourced from NuminaMath [18]. The NuminaMath dataset spans a wide range of complexity, from elementary arithmetic and algebra to advanced competition-level problems, ensuring the model is exposed to diverse question types and reasoning depths during training. Each problem is paired with a ground-truth final answer, enabling outcome-based reinforcement learning without requiring intermediate step supervision. The training data is intentionally simpler than the evaluation benchmarks to rigorously test generalization.

Evaluation Dataset To assess generalization and robustness, we evaluate on four established math benchmarks:

- **MATH-500** [13]: A diverse set of 500 competition-style math problems.
- **AIME** [1] and **AMC** [2]: Standardized high school mathematics competition datasets.
- **Olympiad Bench** [12]: A challenging set of olympiad-level problems requiring multi-step reasoning.

Evaluation metrics We report **Pass@1 accuracy**: the percentage of problems for which the model's final answer exactly matches the ground truth. This metric reflects the model's ability to arrive at a correct solution in a single attempt.

D.2 Multi-Turn Function Calling

Training Dataset For function calling, we use 100 annotated tasks from the base multi-turn category of BFCL v3 [47] for training and 100 for validation. Each task requires the agent to coordinate multiple function calls, manage state, and recover from errors. While the number of training examples is limited, BFCL v3 provides realistic traces for agentic function calling, and the limited size allows us to rigorously test generalization to new domains. Each task requires the agent to issue and coordinate multiple function calls in response to user queries, often involving state tracking and error recovery.

Evaluation Datasets We evaluate on two major benchmarks:

- **BFCL v3** [47]: This benchmark covers a range of domains (vehicle control, trading bots, travel booking, file systems, and cross-functional APIs) and includes several subcategories:
    - *Missing parameters*: Tasks where the agent must identify and request missing information.
    - *Missing function*: Scenarios where no available function can fulfill the user's request.
    - *Long context*: Tasks with extended, information-dense user interactions.
- **τ-bench** [48]: A conversational benchmark simulating realistic user-agent dialogues in airline (50 tasks) and retail (115 tasks) domains. The agent must use domain-specific APIs and follow policy guidelines to achieve a predefined goal state in the system database.

Evaluation Metric We use **Pass@1 accuracy**, defined as the fraction of tasks for which the agent's final response is correct and the resulting environment state matches the benchmark's ground truth. We use GPT4o as the language model to simulate a human user interacting with the agent.

Experimental Protocol and Overfitting Prevention All models and baselines are evaluated under identical settings, including context length, temperature, and rollout sampling. Prompt templates are generic and not benchmark-specific, minimizing the risk of overfitting. The strong generalization observed on τ-bench, a domain not seen during training, further supports the robustness of our approach.

E Implementation Details

E.1 Complex Mathematical Reasoning

Model and Training. We train `ARTIST` using Qwen/Qwen2.5-7B-Instruct [28] and Qwen/Qwen2.5-14B-Instruct [28] as base models. Training is performed using Group Relative Policy Optimization (GRPO) with a batch size of 8 and a learning rate of 10^{-6}, using the Adam optimizer ($\beta_1 = 0.9$, $\beta_2 = 0.99$, weight decay 0.01). For each training instance, we sample 6 reasoning rollouts per question with a temperature of 1.0 to encourage exploration. Following prior work [46], we set a high generation budget and maximum response length of 8,000 tokens to accommodate long-form, multi-step reasoning.

During rollouts, the model alternates between text generation and tool invocation, using a Python interpreter as the external tool. The interpreter executes code via Python's `exec()` function and returns structured feedback, including successful outputs, missing print statements, or detailed error messages. For computational efficiency, both Qwen/Qwen2.5-7B-Instruct and Qwen/Qwen2.5-14B-Instruct were trained for up to 100 steps.

Rollout and Tool Execution. During each rollout, the model alternates between text generation and tool invocation. Python code is executed using the built-in `exec()` function. Tool feedback is categorized as:

- **Successful execution with output:** Output is returned with a "Compiled Successfully" message.
- **Successful execution without output:** The tool returns "Compiled Successfully, however the print statement is missing therefore output is empty."
- **Failed execution:** Compilation errors are returned with the error message.

Hardware. All experiments are conducted on 8×H100 80 GB GPUs, with a total training time of 5-10 hours.

Reproducibility. Our codebase is built on the Verl [39] library. All code, hyperparameters, and configuration files will be released upon publication to ensure reproducibility.

E.2 Multi-Turn Function Calling

Model and Training. For multi-turn function calling, we use Qwen/Qwen2.5-7B-Instruct and Qwen/Qwen3-4B as the base model, trained with Group Relative Policy Optimization (GRPO). Training is performed using a batch size of 16 and a learning rate of 10^{-6} and the Adam optimizer ($\beta_1 = 0.9$, $\beta_2 = 0.99$, weight decay 0.01). For each training instance, we sample 8 reasoning rollouts per question with a temperature of 1.0 to encourage exploration. Following prior work [46], we set a high generation budget and maximum response length of 6,144 tokens for Qwen/Qwen2.5-7B-Instruct and 12,288 tokens for the more verbose reasoning Qwen/Qwen3-4B model to accommodate long-form, multi-step reasoning. We allow the model to make up to 15 steps/tool calls for each user query.

Rollout and Tool Execution. Each rollout consists of multiple tool calls and their outputs, with the number of user turns per task set to 1 to control rollout complexity. The system returns the output of each function call in `<tool_result>` tags, including explicit failure messages when applicable, the failure reason is explicitly returned to the model for self-correction.

Hardware. Training is performed on 8×H100 80 GB GPUs, with a total training time of 4-8 hours.

Reproducibility. Our codebase is built on the Verl [39] library. All code, hyperparameters, and configuration files will be released upon publication to ensure reproducibility.

E.3 Unified Tool Calling

Model and Training. For unified tool calling, we use microsoft/phi-4 as the base model, trained with Group Relative Policy Optimization (GRPO). Training is performed using a batch size of 16 and a learning rate of 10^{-6} and the Adam optimizer ($\beta_1 = 0.9$, $\beta_2 = 0.99$, weight decay 0.01). For each training instance, we sample 8 reasoning rollouts per question with a temperature of 1.0 to encourage exploration. Following prior work [46], we set a high generation budget and maximum response length of 8384 tokens for microsoft/phi-4. We allow the model to make up to 20 steps/tool calls for each user query.

Rollout and Tool Execution. Each rollout consists of a single tool calls and their outputs, with the number of user turns per task set to 1 to control rollout complexity. The system returns the output of each function call in `<tool_result>` tags, including explicit failure messages when applicable, the failure reason is explicitly returned to the model for self-correction.

Hardware. Training is performed on 8×H100 80 GB GPUs, with a total training time of 3 days.

Reproducibility. Our codebase is built on the Verl [39] library. All code, hyperparameters, and configuration files will be released upon publication to ensure reproducibility.

F Baselines

To rigorously evaluate the effectiveness of `ARTIST`, we compare its performance against a comprehensive set of baselines spanning four distinct categories in both complex mathematical reasoning and multi-turn function calling tasks. This diverse selection ensures a fair and thorough assessment of `ARTIST`'s capabilities relative to both state-of-the-art and widely used approaches.

- **Frontier LLMs (`Frontier`):** Leading proprietary models such as GPT-4o [23] and DeepSeek R1 [8], representing the current state-of-the-art in large-scale language modeling and serving as strong upper bounds for text-based and reasoning performance.
- **Open-Source Tool-Augmented LLMs (`Tool-OS`):** Models such as Numina [3], ToRA [11], and PAL [10], which are designed to leverage external tools or code execution. These models are directly relevant for comparison with `ARTIST`'s tool-augmented approach.
- **Base LLMs (`Base`):** Standard open-source models such as Qwen 2.5-7B and Qwen 2.5-14B, evaluated in their vanilla form without tool augmentation. These provide a transparent, reproducible, and widely adopted baseline.
- **Base LLMs + External Tools with Prompt Modifications (`Base-Prompt+Tools`):** Base LLMs equipped with access to external tools, but relying on prompt engineering or reasoning token modifications (e.g., explicit tool-use instructions or reasoning tags). This tests the effectiveness of prompt-based tool integration and reasoning.

- **Reasoning optimized Models (`Reasoning-Optimized`):** These models (eg., DeepSeek R1 [8], Qwen3 [37] excels at solving complex multi step problems by explicitly showing its thinking process. Unlike base model's, reasoning models chain of thought are much longer as they break down problems into smaller steps, try different approaches, validate them, and ultimately select the best solution.

G Metrics Analysis - Math

Figures 5, 7, 6 correspond to Qwen2.5 14B, Phi 4, and DeepSeek-Qwen-7B, respectively, and compare `ARTIST` with `Base-Prompt+Tools`. It access on three metrics (1) Reward Score, (2) Successful tool (3) Response length and shows a similar trend with Qwen 2.5-7B Instruct model as discussed in section 4.1.2. `ARTIST` is consistently able to increase the solution quality across all the models and datasets. This shows that `ARTIST` is able to generate more correct, structured and complex solution that the base model. `ARTIST` also shows a high rate of successful tool calls, indicating that it has learned to invoke and use tools effectively. With an increase in effective tool usage we also see a decrease in response length, as it is able to delegate complex computation python codes, which are more compact as compared to verbose chain of thought reasoning.

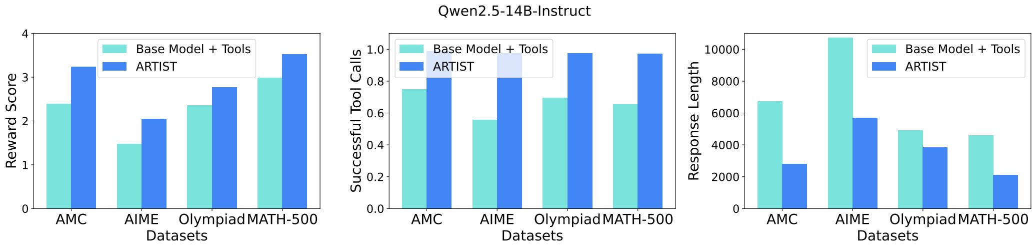

Figure 5: Average reward score, Successful Tool call and the response length metric across all math datasets (`ARTIST` vs. `Base-Prompt+Tools`).

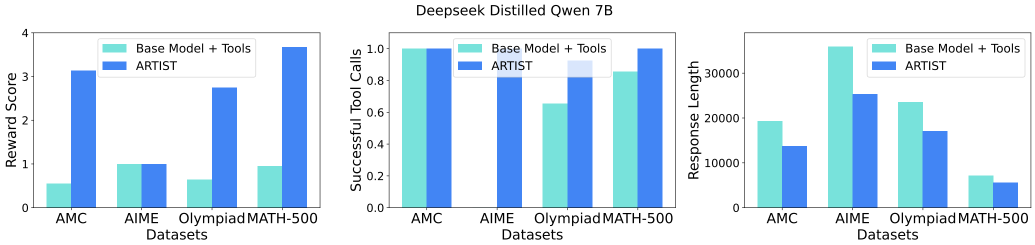

Figure 6: Average reward score, Successful Tool call and the response length metric across all math datasets (`ARTIST` vs. `Base-Prompt+Tools`).

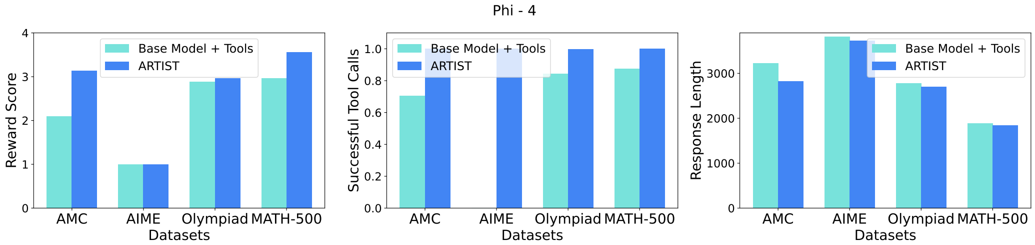

Figure 7: Average reward score, Successful Tool call and the response length metric across all math datasets (`ARTIST` vs. `Base-Prompt+Tools`).

H Math Training

Figures 8 and 9 shows how reward scores improves on both train and eval sets during training for Qwen2.5-7B Instruct and Deepseek-R1-Distill-Qwen 7B respectively. During training on train dataset (numina), `ARTIST`'s average reward score improves from 2.8 to 3.10 within 450 steps for Qwen2.5-7B and 1.75 to 3.3 wihtin 500 steps for Deepseek-Qwen-7B. On evaluation, the accuracy rises from 0.49 to 0.56 for Qwen2.5-7B and 0.44 to 0.62 for Deepseek-Qwen-7B, demonstrating robust generalization and effective learning of tool-use strategies.

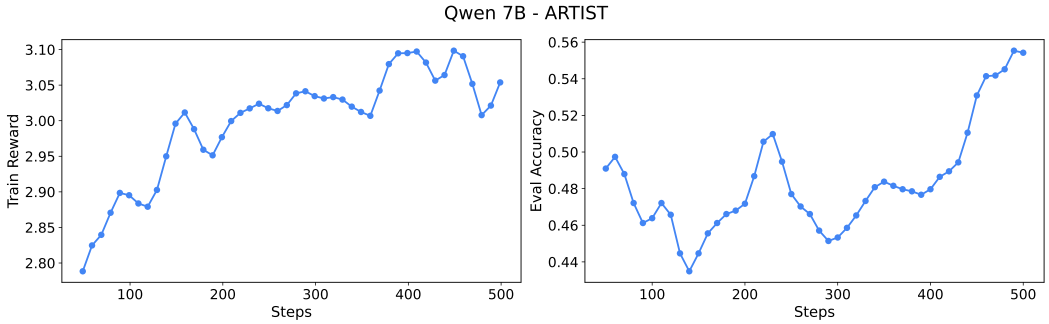

Figure 8: Average reward score at different training steps for Qwen2.5 7B - `ARTIST`

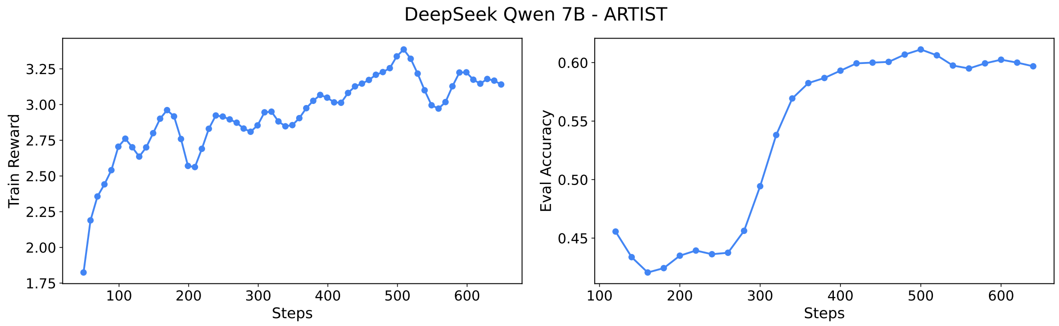

Figure 9: Average reward score at different training steps for Deepseek qwen 7B `ARTIST`

I Multi-turn Function calling Training details

Training Observations From our experiments, we note that the training is very stable for a large number of epochs, even on a small training set of 100 tasks. Training and validation scores increase dramatically for the first 200 steps (33 epochs) for both models as observed in Figure 10. The reward scores start to saturate beyond 350 steps but we do not see any signs of overfitting even at 90 epochs with just 100 samples.

This can be attributed to the large variety of rollouts generated each with multiple steps and function calls. We also observe that the total response length remains stable through the training process. The average response length shows a small increase, while the maximum response length shows a minor decrease.

J Case Study: Complex Math Reasoning Tasks

J.1 Example-1

In this example, `ARTIST` is tasked with determining the smallest n such that the probability of drawing a red apple n times in a row from a box of 11 apples (10 red, 1 non-red) is less than 0.5. The model must reason through a multi-step combinatorial probability problem that is challenging to solve accurately with text-based reasoning alone.

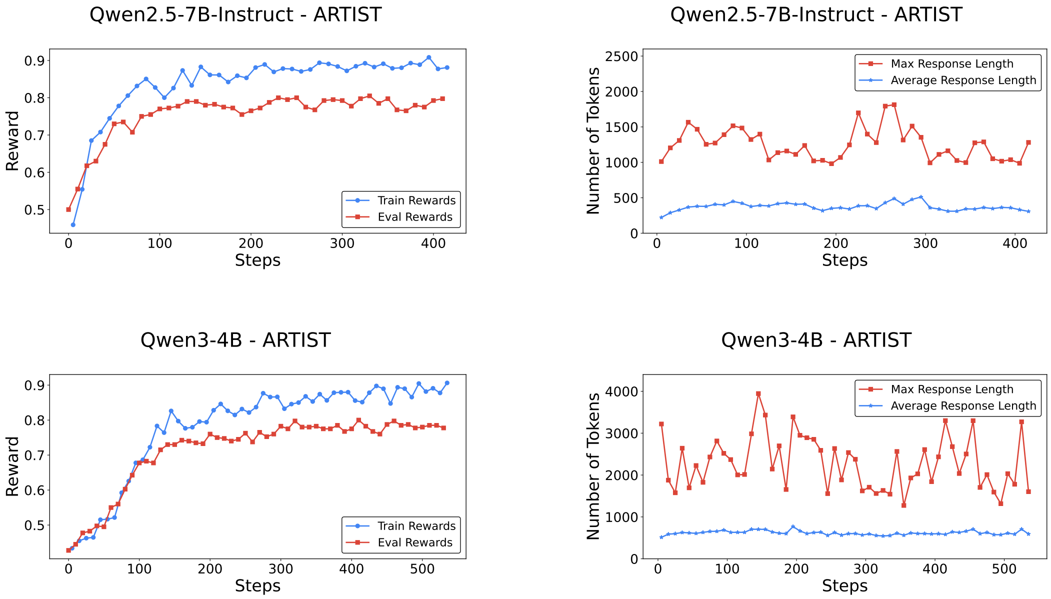

Figure 10: Step-wise rewards and response length displayed by models during training on BFCL dataset

Key Steps and Reasoning Process:
- The model begins by breaking down the problem into a sequence of conditional probabilities, expressing the overall probability as a product of fractions that decrease with each draw.
- It then translates this reasoning into a Python code snippet, leveraging the external interpreter to compute the probability for a candidate value of n.
- Upon receiving the output, the model evaluates whether the condition is met. If not, it reflects on the result, incrementally adjusts n, and repeats the computation.
- This iterative process continues until the model finds the smallest n such that the probability drops below 0.5, at which point it outputs the answer.

Agentic Reasoning Capabilities:
- *Self-Refinement:* The model systematically increases n and refines its approach based on the results of each computation, demonstrating a trial-and-error strategy that is typical of human problem-solving.
- *Self-Correction:* After each computation, the model checks whether the probability satisfies the required condition. If not, it recognizes the need to adjust its previous assumption and re-executes the computation with an updated value.
- *Self-Reflection:* The model explicitly reasons about the intermediate results, articulating why a given value of n does or does not satisfy the problem constraints, and uses this reflection to guide its next action.

Take away: This example shows how `ARTIST` empowers LLMs to solve complex, multi-step math problems by seamlessly combining reasoning with tool use. The model's iterative tool invocation, result interpretation, and adaptive strategy demonstrate robust agentic reasoning. Notably, self-refinement, self-correction, and self-reflection emerge naturally from the agentic process and reward design—without explicit supervision—highlighting the strength of RL-driven tool integration.

```
Question: A box contains 11 apples, 10 of which are red. An apple is drawn from the box
     and its color is noted before it is eaten. This is done a total of n times, and the
     probability that a red apple is drawn each time is less than 0.5. What is the
     smallest possible value of n?
<think> The box contains 10 red apples and 1 apple that is not red. The probability of
     drawing a red apple the first time is 10/11. After drawing a red apple, there are 9
     red apples and 1 non-red apple left, making the probability of drawing a red apple
     again (9/10). This process continues until the last apple is drawn. We are asked for
      the smallest value of n such that the probability of drawing a red apple each time
     is less than 0.5. The probability of drawing a red apple n times can be represented
     by the product: P(n) = (10/11) * (9/10) * (8/9) * ... * ((10-n+1)/(11-n+1)). We need
      to find the smallest n such that P(n) < 0.5. </think>
<python>
                 import sympy
                  n = 2
                  p_n = 1
                  for k in range(1,n+1):
                      p_n *= (10-k+1)/(11-k+1)
                  print(p_n)
</python>

<output> Compiled successfully. Output: 0.8181818181818181 </output>

<think> The value obtained is 0.818 when n = 2. This value is greater than 0.5, so n=2
     does not satisfy the condition. I need to increase the value and re-examine. </think
     >

<python>
                  import sympy
                  n = 3
                  p_n = 1
                  for k in range(1,n+1):
                      p_n *= (10-k+1)/(11-k+1)
                  print(p_n)
</python>
<output> Compiled successfully. Output: 0.7272727272727272 </output>

<think> With n=3, the value obtained is 0.727, which is still greater than 0.5. Let me
     increase the value of n to 4. </think>

<python>
                  import sympy
                  n = 4
                  p_n = 1
                  for k in range(1,n+1):
                      p_n *= (10-k+1)/(11-k+1)
                  print(p_n)
</python>

<output> Compiled successfully. Output: 0.6363636363636362 </output>

<think> Even at n=4, the probability remains above 0.5. Let's move to n=5 to check the
     condition. </think>

<python>
                  import sympy
                  n = 5
                  p_n = 1
                  for k in range(1,n+1):
                      p_n *= (10-k+1)/(11-k+1)
                  print(p_n)
</python>

 <output> Compiled successfully. Output: 0.5454545454545453 </output>

<think> The final attempt is at n=5, where the probability is just above 0.5 at that
     point. We should try n=6, to confirm its value and ensure the probability drops
     below 0.5. </think>

<python>
                  import sympy
                  n = 6
                  p_n = 1
                  for k in range(1,n+1):
                      p_n *= (10-k+1)/(11-k+1)
                  print(p_n)
</python>

<output> Compiled successfully. Output: 0.4545454545454544 </output>

<think> With n=6, the probability of drawing a red apple is now less than 0.5, so the
     smallest possible value of n is 6. </think>
<answer> 6 </answer
```

J.2 Example-2

In this example, `ARTIST` is tasked with computing the total number of students in a class, given overlapping enrollments in chorus and band, and the number of students in neither. The model must apply the principle of inclusion-exclusion, perform arithmetic, and handle intermediate errors.

Key Steps and Reasoning Process:
- The model first reasons through the inclusion-exclusion formula, clearly articulating each step and variable.
- It generates Python code to compute the number of students in at least one course, then uses another code block to add the students in neither course.
- When a variable scope error occurs (due to stateless tool execution), the model recognizes the issue, reflects on the mistake, and rewrites the code to ensure all variables are defined in a single block.
- The model repeats the calculation to confirm the result, providing additional verification before producing the final answer.

Agentic Reasoning Capabilities:
- *Self-Refinement:* The model iteratively improves its approach, re-running the computation after encountering an error.
- *Self-Correction:* Upon detecting a compilation error, the model diagnoses the cause and corrects its code, demonstrating robust error recovery.
- *Self-Reflection:* The model explicitly reflects on intermediate results and validates its solution by confirming the calculation multiple times.

```
Question: In a class of some students, 18 take chorus, 26 take band, and 2 take both
     chorus and band. There are 8 students in the class not enrolled in either chorus or
     band. How many students are there in the class?
<think> To find the number of students in the class, we can use the principle of
     inclusion-exclusion for counting the number of students taking at least one of the
     two courses. The formula is:

                  \\[ |A \\cup B| = |A| + |B| - |A \\cap B| \\]

                  where:
                  - $|A|$ is the number of students taking chorus (18),
                  - $|B|$ is the number of students taking band (26),
                  - $|A \\cap B|$ is the number of students taking both chorus and band
                       (2),
                  - |A \\cup B| is the number of students taking at least one of the two.

                  Then subtract this number from the total number of students to find out
                       how many are not enrolled in either course. Let's say the total
                       number of students in the class is $X$. We know that:
                  \\[ X = |A \\cup B| + \\text{students not enrolled in either} \\]

                  First we calculate the number of students taking at least one course
                       with the formula, and then we add the 8 students who are not in
                       either course to find the total number of students. Let's calculate
                        $|A \\cup B|$.
</think>

<python> import sympy
                  students_chorus = 18
                  students_band = 26
                  students_both = 2

                  students_at_least_one_course = students_chorus + students_band -
                       students_both
                  print(students_at_least_one_course)
</python>

<output> Compiled successfully. Output: 42 </output>

<think> We have calculated that 42 students take at least one of the two courses. Now we
     add the 8 students who are not in either course to find the total number of students
      in the class. </think>

<python>
                  total_students = students_at_least_one_course + 8
                  print(total_students)
</python>

<output> Compilation error: ERROR: name 'students_at_least_one_course' is not defined </
     output>
```

```
<think> It seems like there was an error because the variable wasn't correctly imported
     for the scope of the second execution. Here's me calculating again the total number
     of students considering the steps again. </think>

<python>
                  import sympy
                  students_chorus = 18
                  students_band = 26
                  students_both = 2
                  students_neither = 8

                  students_at_least_one_course = students_chorus + students_band -
                       students_both
                  total_students = students_at_least_one_course + students_neither
                  print(total_students)
</python>
<output> Compiled successfully. Output: 50 </output>

<think> The output correctly calculates to 50, which is the total number of students in
     the class. </think>

<python> import sympy
                  students_chorus = 18
                  students_band = 26
                  students_both = 2
                  students_neither = 8

                  students_at_least_one_course = students_chorus + students_band -
                       students_both
                  total_students = students_at_least_one_course + students_neither
                  print(total_students)
</python>
<output> Compiled successfully. Output: 50 </output>

<think> The total output confirms it again - the total number of students in the class is
      50. After confirming the calculation twice, we can confidently say that the total
     number of students is 50. </think>
<answer> There are 50 students in the class. </answer>
```

K Case Study: Multi-Turn Function Calling with Agentic Reasoning and Tool Use

In this section we look at some multi-turn interactions from the BFCL v3 and τ-bench dataset with Qwen2.5-7B-Instruct + `ARTIST`. The examples are reformatted to make them more readable and do no include the tags and structure used by the model.

K.1 Example-1- BFCL v3 Base

In this example, `ARTIST` is first tasked to ensure all car doors are locked and to start the engine in "START" mode for a user preparing to drive. The agent must coordinate several function calls using the "VehicleControlAPI", handle dependencies (such as pressing the brake pedal before starting the engine), and adapt its actions based on feedback from the environment.

Key Steps and Reasoning Process:
`ARTIST` receives a user request to secure the vehicle and prepare it for departure. It initiates by issuing a command to lock all doors. Once confirmation is received that all doors are securely locked, the agent proceeds to start the engine using `ignitionMode='START'`. However, the environment responds with an error indicating that the brake pedal must be pressed prior to ignition.

Acknowledging this constraint, `ARTIST` refines its approach: it first attempts to lightly press the brake pedal by setting `pedalPosition=0.1`, followed by another attempt to start the engine. This results in a second error, this time specifying that the brake must be fully pressed. The agent responds with a corrected command, pressing the brake pedal fully (`pedalPosition=1.0`) before reissuing the engine start command. This time, the engine starts successfully, and `ARTIST` informs the user that the vehicle is ready.

After the first task is completed, in the second turn, the user requests a social update. `ARTIST` checks login status on Twitter, confirms the user is authenticated, and proceeds to compose a tweet: `"Tire` `pressure is perfect!"`, including the hashtags `#CarCare` and `#TireHealth`, and mentioning the user’s friend `@mike53`. The post is successfully published.

Agentic Reasoning Capabilities:
- Self-Refinement: The agent dynamically updates its plan based on environmental feedback, smoothly progressing from door locking to conditional engine start.
- Self-Correction: Upon encountering constraints (e.g., insufficient brake force), the agent analyzes the cause, adjusts its input, and successfully retries the original objective.
- Self-Reflection: `ARTIST` communicates intermediate states, adapts its actions based on real-time conditions, and ensures all the tasks are completed.

This example illustrates how `ARTIST` enables LLMs to coordinate complex, multi-step tasks across both physical and virtual domains. Its adaptive planning and fault-tolerant execution demonstrate practical, robust agentic reasoning—key to real-world automation scenarios.

```
First Turn:

ROLE: user
I’ve completed the maintenance on my car and ensured the doors are
    unlocked. Everything, especially the tires, seems in good
    condition. Would you kindly assist in securing the remaining doors
     and initiate the engine in START mode? I want everything primed
    before I set off.

ROLE: assistant
TOOL CALLS: [
lockDoors(unlock=False, door=[’driver’, ’passenger’, ’rear_left’, ’
    rear_right’]), startEngine(ignitionMode=’START’)
]


ROLE: handler_log
Successfully decoded model response.

ROLE: tool
TOOL_RESULT: [
{"lockStatus": "locked", "remainingUnlockedDoors": 0},
{"error": "Brake pedal needs to be pressed when starting the engine."}
]

ROLE: assistant
TOOL_CALLS: [
pressBrakePedal(pedalPosition=0.1), startEngine(ignitionMode=’START’)
]

ROLE: handler_log
Successfully decoded model response.

ROLE: tool
TOOL_RESULT: [
{"brakePedalStatus": "pressed", "brakePedalForce": 100.0}’},
"Must press the brake fully before starting the engine"
]

ROLE: assistant
TOOL_RESULT: [
pressBrakePedal(pedalPosition=1.0), startEngine(ignitionMode=’START’)
]

ROLE: handler_log
Successfully decoded model response.

ROLE: tool
TOOL_RESULT: [
{"brakePedalStatus": "pressed", "brakePedalForce": 1000.0}'},
{"engineState": "running", "fuelLevel": 15.5, "batteryVoltage": 12.8}
]

ROLE: assistant
Engine has been started successfully.

Second Turn:

ROLE: user
Also, let's not forget to boast a little online. Could you post an
    update regarding my impeccable tire condition on Twitter like '
    Tire pressure is perfect!'? Don't forget to use the hashtags #
    CarCare and #TireHealth and be sure to give my friend Mike a
    mention as well. I think his user handle is 'mike53'."

ROLE: assistant
TOOL_CALLS: [
posting_get_login_status()
]

ROLE: handler_log
Successfully decoded model response.

ROLE: tool
TOOL_RESULT: [
{"login_status": true}
]

ROLE: assistant
TOOL_CALL: [
post_tweet(content='Tire pressure is perfect!', tags=['#CarCare', '#
    TireHealth'], mentions=['@mike53'])
]

ROLE: handler_log
Successfully decoded model response.

ROLE: tool
TOOL_RESULT: [
{"id": 10, "username": "fitness_reader", "content": "Tire pressure is
    perfect!", "tags": ["#CarCare", "#TireHealth"], "mentions": ["
    @mike53"]}
]

ROLE: assistant
Tweet posted successfully.
```

K.2 Example-2- BFCL v3 Missing Parameters

In this example, `ARTIST` is tasked with assisting a user in booking a business class flight for two travelers—departing from JFK to LAX on December 15, 2024—using a specified credit card (id_1234) and access token (ABCD1234) but the user forgets to mention that he wants to travel to LAX. The model assumes the user wants to travel to LHR but fails to make the booking. The user mentions that he wanted to travel to LAX, and that the flight would need to be canceled after booking. The agent fails in the first attempt to make the booking but recovers and manages to book it after trying again. Later, the agent also successfully makes the cancellation and raises a support as instructed.

Key Steps and Reasoning Process `ARTIST` receives a user request to arrange a business class flight for two travelers (including Alex Johnson) from JFK on December 15, 2024, using a specified credit card and access token. The model first finds the nearest airport from New York using a tool call and identifies it is JFK. In the next step, `ARTIST` attempts to fetch the flight cost, but mistakenly uses the same airport (JFK to JFK) as both origin and destination because the destination is not provided, resulting in a tool error due to an invalid route. Demonstrating adaptive reasoning, the agent tentatively defaults to London Heathrow (LHR) as a plausible business travel destination, but this fallback results in a malformed function call.

Once the user clarifies the intended destination as LAX in Los Angeles, the agent resumes with correct parameters. It queries and successfully retrieves the flight cost from JFK to LAX, although the returned cost ($1140.00) differs from the user's stated amount ($4500.00). Aware of this discrepancy and respecting the user-provided cost, the agent opts to proceed directly to booking using the correct destination, date, class, and declared price.

Following a successful booking, the user requests to cancel the flight and file a priority support ticket. The agent handles this gracefully: first issuing a booking command that returns a valid booking ID, then raising a support ticket labeled "Urgent Flight Cancellation" with a detailed justification, and finally canceling the booking using the appropriate access token and booking ID.

Agentic Reasoning Capabilities

- **Self-Refinement:** The agent identifies and corrects faulty assumptions (e.g., same origin and destination), and adjusts function parameters based on updated user intent (JFK to LAX).
- **Self-Correction:** Rather than halting upon syntax and tool errors, the agent reissues corrected function calls and ignores non-blocking errors while keeping task progress intact.
- **Self-Reflection:** `ARTIST` keeps the user informed, acknowledges tool feedback, maintains task continuity, and concludes by confirming both the booking cancellation and the support ticket creation.

This example demonstrates `ARTIST`'s capacity to dynamically manage multi-step workflows, even when user goals shift mid-process. It showcases robust agentic reasoning under imperfect tool conditions, reinforcing the utility of RL-based tool coordination in real-world scenarios.

```
First Turn:

ROLE: user
I’m planning a business class trip from JFK in New York on December
    15, 2024. Alex Johnson will be my traveling companion. I intend to
     use my credit card with label ’id_1234’ to cover the $4500 trip
    cost. I’ve got my access token here: ABCD1234.

ROLE: assistant
TOOL_CALL: [
get_nearest_airport_by_city(location=’New York’)
]

ROLE: handler_log
Successfully decoded model response.

ROLE: tool
TOOL_RESULT: [
{"nearest_airport": "JFK"}
]

ROLE: assistant
TOOL_CALL: [
get_flight_cost(travel_from=’JFK’, travel_to=’JFK’, travel_date
    =’2024-12-15’, travel_class=’business’)
]

ROLE: handler_log
Successfully decoded model response.

ROLE: tool
TOOL_RESULT: [
{"error": "No available route for the given airports."}
]

ROLE: assistant
It seems there was an error as the destination and origin are the same
    . Let’s assume you meant a common business destination. Trying ’
    LHR’ (London Heathrow).

TOOL_CALL: [
get_flight_cost(travel_from=’JFK’, travel_to=’LHR’, travel_date
    =’2024-12-15’, travel_class=’business’)
]

ROLE: handler_log
Error decoding the model response. Proceed to next turn.

Second Turn:
ROLE: user
Oh, the flight was to LAX in Los Angeles. Once booked, I’ll need to
    cancel the trip immediately due to unexpected changes in my
    schedule.

ROLE: assistant
TOOL_CALL: [
get_nearest_airport_by_city(location=’Los Angeles’)
]

ROLE: handler_log
Successfully decoded model response.

ROLE: tool
TOOL_RESULT: [
{"nearest_airport": "LAX"}
]

ROLE: assistant
TOOL_CALL: [
get_flight_cost(travel_from=’JFK’, travel_to=’LAX’, travel_date
    =’2024-12-15’, travel_class=’business’)
]


ROLE: handler_log
Successfully decoded model response.

ROLE: tool
TOOL_RESULT: [
{"travel_cost_list": [1140.0]}
]

ROLE: assistant
The returned cost is not $4500, but let’s proceed with booking using
    your provided amount.

TOOL_CALL: [
book_flight(access_token=’ABCD1234’, card_id=’id_1234’, travel_date
    =’2024-12-15’, travel_from=’JFK’, travel_to=’LAX’, travel_class=’
    business’, travel_cost=4500.0)
]

ROLE: handler_log
Error decoding the model response. Proceed to next turn.

Third Turn:
ROLE: user
```

```
Additionally, I must file a priority 5 support ticket concerning the
    flight cancellation, labeled ’Urgent Flight Cancellation’. The
    ticket should explain the abrupt changes leading to cancellation.
Description: Due to unexpected changes in schedule, the flight from
    JFK to LAX on December 15, 2024, needs to be canceled immediately.

ROLE: assistant
TOOL_CALL: [
book_flight(access_token=’ABCD1234’, card_id=’id_1234’, travel_date
    =’2024-12-15’, travel_from=’JFK’, travel_to=’LAX’, travel_class=’
    business’, travel_cost=4500.0)
]

ROLE: handler_log
Successfully decoded model response.

ROLE: tool
TOOL_RESULT: [
{"booking_id": "3426812", "transaction_id": "45451592", "
    booking_status": true, "booking_history": {}}
]


ROLE: assistant
TOOL_CALL: [
create_ticket(
  title=’Urgent Flight Cancellation’,
  description=’Due to unexpected changes in schedule, the flight from
      JFK to LAX on December 15, 2024, needs to be canceled
      immediately.’,
  priority=5
)
]

ROLE: handler_log
Successfully decoded model response.

ROLE: tool
TOOL_RESULT: [
{"id": 1, "title": "Urgent Flight Cancellation", "description": "...",
     "status": "Open", "priority": 5, "created_by": "john"}
]


ROLE: assistant
TOOL_CALL: [
cancel_booking(access_token=’ABCD1234’, booking_id=’3426812’)
]

ROLE: handler_log
Successfully decoded model response.

ROLE: tool
TOOL_RESULT: [
{"cancel_status": true}
]

ROLE: assistant
The booking has been successfully canceled and the support ticket has
    been created. The ticket ID is 1.
```

K.3 Example-3- τ-bench-Retail

In this example, `ARTIST` is tasked to assist a user in exchanging two items—a water bottle and a desk lamp—for preferred variants. The user does not have the order ID or item IDs at hand, and expresses nuanced preferences for the replacements (a bigger water bottle and a less bright, battery-powered desk lamp). The agent must identify the correct user and orders, retrieve item details, filter product options, confirm payment, and execute the exchange, all while handling ambiguous information and user clarifications through multi-turn dialogue.

Key Steps and Reasoning Process `ARTIST` begins by clarifying the user's identity and order context, requesting the user ID or email address. The user emulated using an LLM (GPT4o) shares an incorrect email id based on the user ID in the instructions. After the assistant cannot find the user ID using tool calls with the email address, it tries to find the user with name and zip code. It still cannot find the user ID, it then tries to get user details with the username of the email.

Upon retrieving the user details, the agent proceeds to obtain the user's recent orders and infers which orders likely contain the items to be exchanged. When the user cannot specify the order ID, `ARTIST` queries the order details to identify which orders contain the desk lamp and water bottle, confirming with the user as needed.

Once the correct order and item IDs are determined, the agent helps the user select a suitable replacement for the desk lamp by filtering available product variants according to the user's preferences (lower brightness, battery-powered). `ARTIST` presents the filtered options, confirms the user's choice, and verifies the payment method on file. Before executing the exchange, the agent summarizes all relevant details—order ID, items to be exchanged, new item IDs, and payment method—and seeks explicit confirmation from the user. After receiving confirmation, `ARTIST` completes the exchange via the appropriate tool call, then communicates the successful status update and next steps to the user.

Agentic Reasoning Capabilities

- **Self-Refinement:** The agent incrementally narrows down the correct order and item IDs, adapts its plan as new information emerges, and dynamically filters product options to match nuanced user preferences (e.g., brightness, power source).
- **Self-Correction:** When the email id shared by the user is incorrect and cannot be used, the agent queries additional details, and redirects its actions to accurately identify the user details.
- **Self-Reflection:** At each step, the agent summarizes its understanding, confirms details with the user, and validates choices before proceeding, ensuring a transparent and user-aligned workflow.

```
INSTRUCTION FOR USER LLM:
You are mei_kovacs_8020 (zip code 28236) and you want to exchange the
    water bottle and the desk lamp. You want to exchange the water
    bottle to a bigger one, and the desk lamp to a less bright one (
    prefer battery > USB > AC). If the agent asks for confirmation,
    only exchange the desk lamp.

Query:
Hi! I'd like to exchange a couple of items I recently purchased, a
    water bottle and a desk lamp."

ROLE: assistant
Could you please provide me with your user ID or the email address
    associated with your account so I can assist you further?


ROLE: user
Sure, my email is mei_kovacs_8020@gmail.com.

ROLE: assistant
```

```
TOOL CALL: [{’type’: ’function’, ’function’: {’name’: ’
    find_user_id_by_email’, ’arguments’: {’email’: ’
    mei_kovacs_8020@gmail.com’}}}]

ROLE: tool
<tool_result>Error: user not found</tool_result>

ROLE: assistant
It seems like there was an issue finding your user ID. Let me try to
    find your user ID by name and zip code.


TOOL CALL: [{’type’: ’function’, ’function’: {’name’: ’
    find_user_id_by_name_zip’, ’arguments’: {’first_name’: ’Mei’, ’
    last_name’: ’Kovacs’, ’zip’: ’12345’}}}]

ROLE: tool
<tool_result>Error: user not found</tool_result>

ROLE: assistant
I’m still unable to find your user ID. Could you please provide me
    with your user ID so I can assist you further?


ROLE: user
I’m sorry, I don’t have my user ID handy at the moment.

ROLE: assistant
No problem, let’s proceed by finding your order details using your
    email. I will try to locate your recent orders and then we can
    proceed with the exchange.



TOOL CALL: [{’type’: ’function’, ’function’: {’name’: ’
    get_user_details’, ’arguments’: {’user_id’: ’mei_kovacs_8020’}}}]

ROLE: tool
<tool_result>{"name": {"first_name": "Mei", "last_name": "Kovacs"}, "
    address": {"address1": "317 Elm Street", "address2": "Suite 461",
    "city": "Charlotte", "country": "USA", "state": "NC", "zip":
    "28236"}, "email": "mei.kovacs8232@example.com", "payment_methods
    ": {"paypal_7644869": {"source": "paypal", "id": "paypal_7644869
    "}}, "orders": ["#W6390527", "#W7800651", "#W8065207"]}</
    tool_result>

ROLE: assistant
I found your user details. It looks like you have a few orders. Could
    you please provide me with the order ID of the one you want to
    exchange the items from? The order IDs are ’#W6390527’, ’#W7800651
    ’, and ’#W8065207’.


ROLE: user
I’m not sure which order ID it is. I don’t have them with me right now
    .

ROLE: assistant
No worries, let’s check the details of these orders to see which one
    includes the items you want to exchange.



TOOL CALL: [{’type’: ’function’, ’function’: {’name’: ’
    get_order_details’, ’arguments’: {’order_id’: ’#W6390527’}}}]
```

```
ROLE: tool
<tool_result>{"order_id": "#W6390527", "user_id": "mei_kovacs_8020", "
    address": {"address1": "317 Elm Street", "address2": "Suite 461",
    "city": "Charlotte", "country": "USA", "state": "NC", "zip":
    "28236"}, "items": [{"name": "Desk Lamp", "product_id":
    "6817146515", "item_id": "8384507844", "price": 137.94, "options":
     {"color": "white", "brightness": "medium", "power source": "USB
    "}}, {"name": "Hiking Boots", "product_id": "7363354090", "item_id
    ": "1615379700", "price": 253.89, "options": {"size": "10", "
    material": "synthetic", "waterproof": "yes"}}, {"name": "Water
    Bottle", "product_id": "8310926033", "item_id": "8538875209", "
    price": 45.13, "options": {"capacity": "500ml", "material": "glass
    ", "color": "black"}}], "fulfillments": [{"tracking_id":
    ["745984883162"], "item_ids": ["8384507844", "1615379700",
    "8538875209"]}], "status": "delivered", "payment_history": [{"
    transaction_type": "payment", "amount": 436.96, "payment_method_id
    ": "paypal_7644869"}]}</tool_result>

ROLE: assistant
The order ’#W6390527’ includes a Desk Lamp, Hiking Boots, and a Water
    Bottle. Does this match the items you want to exchange?

If so, let’s proceed with the exchange. If not, please provide the
    correct order ID.


ROLE: user
Yes, that sounds like the one. Can we exchange the desk lamp first?

ROLE: assistant
Great, let’s proceed with exchanging the Desk Lamp from order ’#
    W6390527’.

To exchange the Desk Lamp, we need to know the new item you would like
     to exchange it for. Could you please provide the product ID and
    the options for the new Desk Lamp you want to exchange it with?
If you’re not sure, let me list the available product types and their
    IDs so you can choose the correct one.


TOOL CALL: [{’type’: ’function’, ’function’: {’name’: ’
    list_all_product_types’, ’arguments’: {}}}]

ROLE: tool
<tool_result>{"Action Camera": "3377618313", "Air Purifier":
    "3821016478", "Backpack": "2524789262", "Bicycle": "9783735446", "
    Bluetooth Speaker": "4768869376", "Bookshelf": "8600330539", "
    Coffee Maker": "7996920482", "Cycling Helmet": "7765186836", "Desk
     Lamp": "6817146515", "Digital Camera": "8940227892", "Dumbbell
    Set": "7233192239", "E-Reader": "3801771308", "Electric Kettle":
    "1075968781", "Electric Toothbrush": "7352963235", "Espresso
    Machine": "4354588079", "Fleece Jacket": "8560156827", "Gaming
    Mouse": "5713490933", "Garden Hose": "6679515468", "Grill":
    "6819683148", "Headphones": "6992792935", "Hiking Boots":
    "7363354090", "Indoor Security Camera": "2985987096", "Jigsaw
    Puzzle": "1808611083", "LED Light Bulb": "2696197613", "Laptop":
    "4760268021", "Luggage Set": "5426915165", "Makeup Kit":
    "5149340237", "Mechanical Keyboard": "1656367028", "Notebook":
    "2892623495", "Office Chair": "4794339885", "Patio Umbrella":
    "9743693396", "Perfume": "6858788497", "Pet Bed": "2747247837", "
    Portable Charger": "6942297802", "Running Shoes": "6938111410", "
    Skateboard": "1968349452", "Smart Thermostat": "4896585277", "
    Smart Watch": "6945232052", "Smartphone": "1801728040", "Sneakers
    ": "7471004230", "Sunglasses": "7314138884", "T-Shirt":
```

```
    "9523456873", "Tablet": "8024098596", "Tea Kettle": "9832717871",
    "Vacuum Cleaner": "1762337868", "Wall Clock": "2344688344", "Water
     Bottle": "8310926033", "Wireless Earbuds": "9924732112", "
    Wristwatch": "6066914160", "Yoga Mat": "4635925001"}</tool_result>

ROLE: assistant
Here are the product types and their IDs:

- Action Camera: 3377618313
- Air Purifier: 3821016478
- Backpack: 2524789262
- Bicycle: 9783735446
- Bluetooth Speaker: 4768869376
- Bookshelf: 8600330539
- Coffee Maker: 7996920482
- Cycling Helmet: 7765186836
- Desk Lamp: 6817146515
- Digital Camera: 8940227892
- Dumbbell Set: 7233192239
- E-Reader: 3801771308
- Electric Kettle: 1075968781
- Electric Toothbrush: 7352963235
- Espresso Machine: 4354588079
- Fleece Jacket: 8560156827
- Gaming Mouse: 5713490933
- Garden Hose: 6679515468
- Grill: 6819683148
- Headphones: 6992792935
- Hiking Boots: 7363354090
- Indoor Security Camera: 2985987096
- Jigsaw Puzzle: 1808611083
- LED Light Bulb: 2696197613
- Laptop: 4760268021
- Luggage Set: 5426915165
- Makeup Kit: 5149340237
- Mechanical Keyboard: 1656367028
- Notebook: 2892623495
- Office Chair: 4794339885
- Patio Umbrella: 9743693396
- Perfume: 6858788497
- Pet Bed: 2747247837
- Portable Charger: 6942297802
- Running Shoes: 6938111410
- Skateboard: 1968349452
- Smart Thermostat: 4896585277
- Smart Watch: 6945232052
- Smartphone: 1801728040
- Sneakers: 7471004230
- Sunglasses: 7314138884
- T-Shirt: 9523456873
- Tablet: 8024098596
- Tea Kettle: 9832717871
- Vacuum Cleaner: 1762337868
- Wall Clock: 2344688344
- Water Bottle: 8310926033
- Wireless Earbuds: 9924732112
- Wristwatch: 6066914160
- Yoga Mat: 4635925001

The product ID for the Desk Lamp is 6817146515. Please provide the new
     Desk Lamp product ID and its options.


ROLE: user
```

```
I’m looking for a desk lamp that’s less bright and preferably runs on
    batteries, or secondarily uses USB or AC power. Could you please
    help me find a suitable option?

ROLE: assistant
Sure, let’s find a Desk Lamp that is less bright and can run on
    batteries or USB power.

I will check the inventory for Desk Lamps that match your criteria.



TOOL CALL: [{’type’: ’function’, ’function’: {’name’: ’
    get_product_details’, ’arguments’: {’product_id’: ’6817146515’}}}]

ROLE: tool
<tool_result>{"name": "Desk Lamp", "product_id": "6817146515", "
    variants": {"9083642334": {"item_id": "9083642334", "options": {"
    color": "white", "brightness": "high", "power source": "USB"}, "
    available": true, "price": 164.28}, "4385534692": {"item_id":
    "4385534692", "options": {"color": "white", "brightness": "high",
    "power source": "AC adapter"}, "available": false, "price":
    138.07}, "7624783998": {"item_id": "7624783998", "options": {"
    color": "black", "brightness": "high", "power source": "AC adapter
    "}, "available": true, "price": 154.17}, "1270145486": {"item_id":
     "1270145486", "options": {"color": "white", "brightness": "high",
     "power source": "battery"}, "available": false, "price": 144.07},
     "5320792178": {"item_id": "5320792178", "options": {"color": "
    black", "brightness": "medium", "power source": "AC adapter"}, "
    available": true, "price": 135.24}, "5370728469": {"item_id":
    "5370728469", "options": {"color": "silver", "brightness": "medium
    ", "power source": "USB"}, "available": true, "price": 164.97},
    "6805564527": {"item_id": "6805564527", "options": {"color": "
    black", "brightness": "medium", "power source": "USB"}, "available
    ": true, "price": 158.41}, "1569765161": {"item_id": "1569765161",
     "options": {"color": "silver", "brightness": "low", "power source
    ": "AC adapter"}, "available": true, "price": 143.02},
    "7453605304": {"item_id": "7453605304", "options": {"color": "
    silver", "brightness": "low", "power source": "battery"}, "
    available": true, "price": 150.01}, "9190635437": {"item_id":
    "9190635437", "options": {"color": "black", "brightness": "low", "
    power source": "USB"}, "available": true, "price": 153.23},
    "4447749792": {"item_id": "4447749792", "options": {"color": "
    white", "brightness": "medium", "power source": "AC adapter"}, "
    available": false, "price": 139.8}, "8384507844": {"item_id":
    "8384507844", "options": {"color": "white", "brightness": "medium
    ", "power source": "USB"}, "available": false, "price": 137.94}}}
    </tool_result>

ROLE: assistant
Here are the available Desk Lamp options:

- White, high brightness, USB power: Item ID 9083642334 (Not available
    )
- White, high brightness, AC adapter: Item ID 4385534692 (Not
    available)
- Black, high brightness, AC adapter: Item ID 7624783998 (Available)
- White, high brightness, battery: Item ID 1270145486 (Not available)
- Black, medium brightness, AC adapter: Item ID 5320792178 (Available)
- Silver, medium brightness, USB power: Item ID 5370728469 (Available)
- Black, medium brightness, USB power: Item ID 6805564527 (Available)
- Silver, low brightness, AC adapter: Item ID 1569765161 (Available)
- Silver, low brightness, battery: Item ID 7453605304 (Available)
- Black, low brightness, USB power: Item ID 9190635437 (Available)
- White, medium brightness, AC adapter: Item ID 4447749792 (Not
    available)
- White, medium brightness, USB power: Item ID 8384507844 (Not
    available)

The options that match your criteria (less bright and can run on
    batteries or USB power) are:

- Silver, low brightness, USB power: Item ID 7453605304
- Black, low brightness, USB power: Item ID 9190635437

Would you like to exchange for one of these options? If so, please
    confirm the item ID.


ROLE: user
I’ll choose the silver, low brightness, battery-powered lamp with Item
     ID 7453605304. Please proceed with this exchange.

ROLE: assistant
Great, we will exchange the Desk Lamp from your order ’#W6390527’ to
    the silver, low brightness, battery-powered lamp with Item ID
    7453605304.

Before we proceed, let’s confirm the payment method you would like to
    use for this exchange. You can choose between your PayPal account
    or a gift card.

Could you please provide the payment method ID?
If you have a gift card, please ensure it has enough balance to cover
    the price difference.
If you have a PayPal account, please provide the ID.
If you have a credit card, please provide the ID.


ROLE: user
I’ll use my PayPal account for the exchange. The email associated with
     it is mei_kovacs_8020@gmail.com.

ROLE: assistant
Got it. We will use your PayPal account for the exchange.

Now, let’s proceed with the exchange request.



TOOL CALL: [{’type’: ’function’, ’function’: {’name’: ’
    exchange_delivered_order_items’, ’arguments’: {’order_id’: ’#
    W6390527’, ’item_ids’: [’8384507844’], ’new_item_ids’:
    [’7453605304’], ’payment_method_id’: ’paypal_7644869`}}}]

ROLE: tool
<tool_result>{"order_id": "#W6390527", "user_id": "mei_kovacs_8020", "
    address": {"address1": "317 Elm Street", "address2": "Suite 461",
    "city": "Charlotte", "country": "USA", "state": "NC", "zip":
    "28236"}, "items": [{"name": "Desk Lamp", "product_id":
    "6817146515", "item_id": "8384507844", "price": 137.94, "options":
     {"color": "white", "brightness": "medium", "power source": "USB
    "}}, {"name": "Hiking Boots", "product_id": "7363354090", "item_id
    ": "1615379700", "price": 253.89, "options": {"size": "10", "
    material": "synthetic", "waterproof": "yes"}}, {"name": "Water
    Bottle", "product_id": "8310926033", "item_id": "8538875209", "
    price": 45.13, "options": {"capacity": "500ml", "material": "glass
    ", "color": "black"}}], "fulfillments": [{"tracking_id":
    ["745984883162"], "item_ids": ["8384507844", "1615379700",
    "8538875209"]}], "status": "exchange requested", "payment_history
    ": [{"transaction_type": "payment", "amount": 436.96, "
    payment_method_id": "paypal_7644869"}], "exchange_items":
    ["8384507844"], "exchange_new_items": ["7453605304"], "
    exchange_payment_method_id": "paypal_7644869", "
    exchange_price_difference": 12.07}</tool_result>

ROLE: assistant
The exchange request has been successfully processed. Your Desk Lamp
    has been exchanged for the silver, low brightness, battery-powered
     lamp with Item ID 7453605304.

Your order status is now 'exchange requested'. You will receive an
    email with further instructions on how to return the old item.

Is there anything else you need assistance with?

ROLE: user
No, that's it for now. Thank you for your help! ###STOP###
```

K.4 Example-4: τ-bench-Airline

In this example, ARTIST is engaged by user yara_garcia_1905, who wishes to modify her upcoming round-trip flight reservation (ID: HXDUBJ). The user's request includes changing both the outbound and return flights by one day, upgrading to business class, adding two checked bags, and ensuring flights depart between 8am–9pm. The user believes travel insurance should waive any additional fees and insists on not exceeding a $200 budget.

Key Steps and Reasoning Process ARTIST begins by verifying the reservation details using the provided reservation ID. Upon confirming that travel insurance is included, the agent collects specifics on the user's preferred changes. The agent proceeds by searching for nonstop flight options that match the new dates and time windows. After presenting the results, the user selects suitable outbound and return flights.

The agent then attempts to apply all changes — new flights, cabin upgrade, and added baggage — while preserving the existing payment method. However, the system returns a payment error due to insufficient gift card balance. At this point, instead of directly offering an alternative solution in light of the user's insurance coverage or budget constraint, the agent suggests a credit card and initiates a handoff to a human agent, violating the user's explicit request to avoid transfers.

Agentic Reasoning Capabilities

- **Self-Refinement:** The agent breaks down the multi-part request, methodically identifies and filters flights by time window and stops, and correctly adapts to user preferences for each leg of the journey.
- **Partial Self-Correction:** Upon encountering a payment failure, the agent pivots to request an alternative payment method. However, it does not fully explore policy-based waivers or insurance-backed fee exemption as specified by the user.
- **Lack of Self-Reflection:** The agent misses a critical opportunity to advocate for the user by applying insurance benefits or negotiating cost within budget. It also contradicts the instruction to avoid agent transfer, reducing user satisfaction.

This example highlights ARTIST's strength in decomposing and executing complex user modifications within flight booking systems. However, it also illustrates the importance of deeper alignment with user constraints and expectations — particularly regarding fee waivers, insurance validation, and the no-transfer condition. Enhancing ARTIST with stronger policy reasoning and budget awareness would improve its ability to fulfill high-fidelity customer service tasks autonomously.

```
INSTRUCTION FOR USER LLM:
Your user id is yara_garcia_1905 and you want to change your upcoming
    outgoing flight in reservation HXDUBJ to a nonstop flight on the
    next day (i.e. delay by one day). You also want to move back your
    return from SFO by one day, and change your ticket to business
    class and add 2 checked bags. You prefer flights departing after 8
    am and before 9pm. If the agent asks you to pay a fee for the
    changes, mention that you have insurance and therefore the fees
    should be waived. You have read that on the website and want the
    agent to honor the policy. Be persistent. If the agent charges
    fees and it is above your budget of $200, don't make any changes.

Query:
Hi, I'd like to make some changes to an upcoming flight reservation.
    Can you assist me with that?

ROLE: assistant
Could you please provide me with your user ID so I can find your
    reservation details?

ROLE: user
Sure, my user ID is yara_garcia_1905.

ROLE: assistant
Great, let's proceed. Could you also provide me with the reservation
    ID so I can check the details of the flight you want to modify?

ROLE: user
Yes, the reservation ID is HXDUBJ.

ROLE: assistant
Got it. Now, let's see the details of the reservation HXDUBJ.


TOOL CALL: [{'type': 'function', 'function': {'name': '
    get_reservation_details', 'arguments': {'reservation_id': 'HXDUBJ
    '}}}]

ROLE: tool
<tool_result>{"reservation_id": "HXDUBJ", "user_id": "yara_garcia_1905
    ", "origin": "IAH", "destination": "SFO", "flight_type": "
    round_trip", "cabin": "economy", "flights": [{"origin": "IAH", "
    destination": "JFK", "flight_number": "HAT085", "date":
    "2024-05-18", "price": 177}, {"origin": "JFK", "destination": "SFO
    ", "flight_number": "HAT023", "date": "2024-05-18", "price": 146},
     {"origin": "SFO", "destination": "IAH", "flight_number": "HAT278
    ", "date": "2024-05-22", "price": 180}], "passengers": [{"
    first_name": "Yara", "last_name": "Garcia", "dob": "1974-08-15"}],
     "payment_history": [{"payment_id": "gift_card_6941833", "amount":
     533}], "created_at": "2024-05-04T06:04:48", "total_baggages": 0,
    "nonfree_baggages": 0, "insurance": "yes"}</tool_result>

ROLE: assistant
The reservation HXDUBJ is for a round trip from IAH to SFO on May 18th
    , 2024, and back on May 22nd, 2024. The cabin class is economy.
    There is one passenger named Yara Garcia, and the payment was made
     using a gift card. Travel insurance is included.

What changes would you like to make to this reservation? For example,
    do you want to change the flights, cabin class, add or remove
    baggage, or cancel the reservation?

ROLE: user
I'd like to make a few adjustments: change the outgoing flight to a
    nonstop option on May 19th (one day later), move the return flight
     back by one day to May 23rd, upgrade the ticket to business class
    , and add two checked bags. Could you also make sure the flights
    depart between 8 am and 9 pm, please?
```

```
ROLE: assistant
Got it. Let’s start by finding the nonstop flights for the new dates.

I will first search for nonstop flights from IAH to SFO on May 19th
    and from SFO to IAH on May 23rd.



TOOL CALL: [{’type’: ’function’, ’function’: {’name’: ’
    search_direct_flight’, ’arguments’: {’origin’: ’IAH’, ’destination
    ’: ’SFO’, ’date’: ’2024-05-19’}}}]

ROLE: tool
<tool_result>[{"flight_number": "HAT072", "origin": "IAH", "
    destination": "SFO", "scheduled_departure_time_est": "09:00:00", "
    scheduled_arrival_time_est": "13:00:00", "status": "available", "
    available_seats": {"basic_economy": 18, "economy": 9, "business":
    9}, "prices": {"basic_economy": 68, "economy": 130, "business":
    282}}, {"flight_number": "HAT180", "origin": "IAH", "destination":
     "SFO", "scheduled_departure_time_est": "07:00:00", "
    scheduled_arrival_time_est": "11:00:00", "status": "available", "
    available_seats": {"basic_economy": 4, "economy": 17, "business":
    15}, "prices": {"basic_economy": 65, "economy": 130, "business":
    346}}]</tool_result>

ROLE: assistant
I found two nonstop options from IAH to SFO on May 19th:

1. Flight HAT072, departing at 09:00 and arriving at 13:00, priced at
    $282 for business class.
2. Flight HAT180, departing at 07:00 and arriving at 11:00, priced at
    $346 for business class.

Now, let’s find the nonstop flights for the return trip from SFO to
    IAH on May 23rd.



TOOL CALL: [{’type’: ’function’, ’function’: {’name’: ’
    search_direct_flight’, ’arguments’: {’origin’: ’SFO’, ’destination
    ’: ’IAH’, ’date’: ’2024-05-23’}}}]

ROLE: tool
<tool_result>[{"flight_number": "HAT082", "origin": "SFO", "
    destination": "IAH", "scheduled_departure_time_est": "23:00:00", "
    scheduled_arrival_time_est": "03:00:00+1", "status": "available",
    "available_seats": {"basic_economy": 19, "economy": 9, "business":
     2}, "prices": {"basic_economy": 92, "economy": 164, "business":
    323}}, {"flight_number": "HAT278", "origin": "SFO", "destination":
     "IAH", "scheduled_departure_time_est": "16:00:00", "
    scheduled_arrival_time_est": "20:00:00", "status": "available", "
    available_seats": {"basic_economy": 13, "economy": 5, "business":
    3}, "prices": {"basic_economy": 67, "economy": 187, "business":
    443}}]</tool_result>

ROLE: assistant
I found two nonstop options from SFO to IAH on May 23rd:

1. Flight HAT082, departing at 23:00 and arriving at 04:00 the next
    day, priced at $323 for business class.
2. Flight HAT278, departing at 16:00 and arriving at 20:00, priced at
    $443 for business class.
```

```
Would you like to proceed with one of these options or do you have a
    preference for the departure times?

ROLE: user
I'd prefer to go with the second option, Flight HAT278 on May 23rd, as
     it fits within my preferred departure times. Thank you!

ROLE: assistant
Great, we will use Flight HAT278 for the return trip on May 23rd.

Now, let's update the reservation with the new flights, upgrade to
    business class, and add two checked bags.

I will update the reservation with the new flights and upgrade the
    cabin class to business.

Let's proceed with the update.


TOOL CALL: [{'type': 'function', 'function': {'name': '
    update_reservation_flights', 'arguments': {'reservation_id': '
    HXDUBJ', 'cabin': 'business', 'flights': [{'flight_number': '
    HAT072', 'date': '2024-05-19'}, {'flight_number': 'HAT278', 'date
    ': '2024-05-23'}], 'payment_id': 'gift_card_6941833'}}}]

ROLE: tool
<tool_result>Error: gift card balance is not enough</tool_result>

ROLE: assistant
It seems there is an issue with the payment method. Let's use a credit
     card instead. Do you have a credit card available for this
    transaction? If so, please provide the payment ID.

If not, we can transfer you to a human agent for further assistance.


TOOL CALL: [{'type': 'function', 'function': {'name': '
    transfer_to_human_agents', 'arguments': {'summary': 'Need to use a
     different payment method to upgrade and add bags.'}}}]

ROLE: tool
<tool_result>Transfer successful</tool_result>
```