IN-THE-FLOW AGENTIC SYSTEM OPTIMIZATION FOR EFFECTIVE PLANNING AND TOOL USE

Anonymous authors
Paper under double-blind review

ABSTRACT

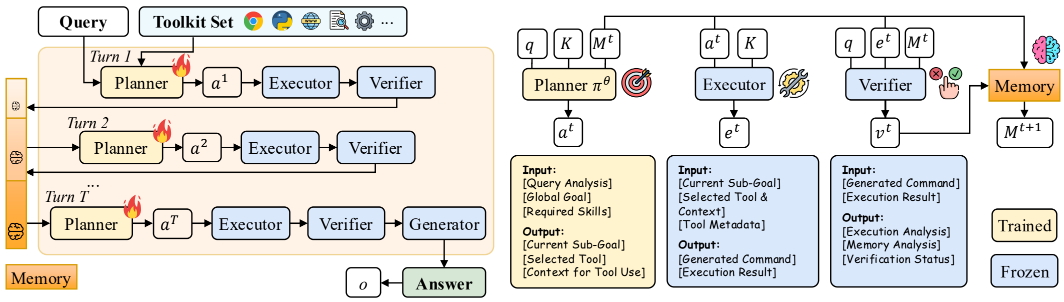
Outcome-driven reinforcement learning has advanced reasoning in large language models (LLMs), but prevailing tool-augmented methods train a single, monolithic policy. This approach scales poorly with long-horizon tasks and diverse tools, and generalizes weakly to new scenarios. Agentic systems offer a promising alternative by decomposing work across specialized modules, yet most remain training-free or rely on offline training decoupled from the live dynamics of multi-turn interaction. We introduce AGENTFLOW, a trainable agentic system that optimizes its planner directly *in the flow* of a multi-module workflow. By coordinating specialized modules (planner, executor, verifier, generator) through an evolving memory, it enables adaptive long-horizon planning and robust tool orchestration. To train the planner on-policy, we propose *Flow-based Group Refined Policy Optimization* (Flow-GRPO), an outcome-driven reinforcement learning method designed for the sparse, long-horizon credit assignment problem in agentic systems. It stabilizes training by broadcasting a single, verifiable final-outcome reward to all decisions within a trajectory and using group-normalized advantages. Through comprehensive experiments on ten benchmarks, AGENTFLOW with a 7B-scale backbone outperforms top-performing baselines with average accuracy gains of 14.9% on search, 14.0% on agentic, 14.5% on mathematical, and 4.1% on scientific tasks, even surpassing larger proprietary models like GPT-4o. Further analyses confirm the benefits of in-the-flow optimization, demonstrating improved planning, enhanced tool-calling reliability, and positive scaling trends with model size and reasoning turns. Codebase is available at `https://anonymous.4open.science/r/agentflow`.

1 INTRODUCTION

Recent advances in large language models (LLMs) have unlocked remarkable reasoning capabilities, largely driven by reinforcement learning (RL) from outcome-based feedback. By fine-tuning models to maximize verifiable rewards, LLMs like DeepSeek-R1 (Guo et al., 2025) and SimpleRL (Zeng et al., 2025b) have demonstrated sophisticated behaviors in self-correction and multi-step deduction.

A complementary line of work augments LLMs with external tools (e.g., web search, code execution) for knowledge retrieval and precise computation. Tool-integrated reasoning (TIR) extends reinforcement learning with verifiable rewards to learn *when* and *how* to call tools by interleaving reasoning (e.g., <think>) with tool invocations (e.g., <tool_call>) under full context (Jin et al., 2025; Song et al., 2025; Chen et al., 2025; Feng et al., 2025). Early systems supported only a single tool type, whereas recent work enables multi-tool settings by encoding tool metadata into prompts (Dong et al., 2025; Qian et al., 2025a; Zhang et al., 2025). However, these methods still train a *single*, monolithic policy under multi-turn full-context reasoning, which introduces scaling challenges: (i) *training* becomes increasingly unstable as horizons lengthen, tool diversity grows, and environments shift with tool feedback (Wang et al., 2025c; Mai et al., 2025; Moonshot AI, 2025; Xue et al., 2025); and (ii) *inference*-time generalization remains brittle to unseen tasks or tools (Dong et al., 2025; Hu et al., 2025b).

Agentic systems (Wu et al., 2024; Hong et al., 2024; Hu et al., 2025b) offer a promising alternative to monolithic tool-integrated reasoning models. They consist of multiple modules—often distinct LLMs with prescribed roles (e.g., planner, critic) or specialized components with dedicated



(a) AgentFlow: In-the-Flow Agentic System (b) In-the-Flow Rollout at Turn t

Figure 1: **(a)** Overview of AGENTFLOW, a trainable agentic system for in-the-flow planning and tool use. Four modules (planner, executor, verifier, generator) coordinate via a shared evolving memory M and toolset K, given a query q. The planner policy is optimized on-policy *inside* the system's multi-turn loop to enable adaptive, long-horizon reasoning. **(b)** A single state transition, showing the action a^t, execution result e^t, and verifier signal v^t that update the memory from M^t to M^{t+1}.

tools and capabilities (e.g., executor, coder)—that coordinate via shared memory and inter-module communication. By decomposing problems into sub-goals and iterating over multiple turns, these systems can tackle tasks that demand diverse tools, long horizons, or multi-stage reasoning. However, achieving robust coordination in such systems ultimately requires *training*, since handcrafted logic or static prompting cannot reliably capture when and how modules should collaborate, adapt to evolving tool outputs, or recover from early mistakes. At the same time, they introduce new *training* challenges: modules coordinate sequentially, outcome feedback propagates through long reasoning chains, and state distributions shift with evolving tool outputs. As a result, most systems remain *training-free*, relying on handcrafted logic or prompting heuristics. While some employ supervised fine-tuning or preference optimization for key modules (Motwani et al., 2024; Park et al., 2025), these off-policy approaches are decoupled from live dynamics and learn poorly from downstream successes or failures. Thus, agentic systems struggle with sparse rewards, brittle adaptation, and inefficient orchestration in dynamic environments.

To address the central challenge of learning long-horizon reasoning with sparse rewards in tool-integrated agentic systems, we introduce AGENTFLOW, a *trainable* framework for effective planning and tool use (Figure 1). AGENTFLOW comprises four specialized modules—planner, executor, verifier, and generator—that interact iteratively over multiple turns via a shared evolving memory and a toolset. The system operates *in the flow*, with each turn cycling through planning, execution, and verification. Unlike prior agentic systems, AGENTFLOW directly optimizes its planner on-policy, *inside* the live multi-turn loop, allowing it to dynamically adapt to trajectories shaped by tool calls, verifier signals, and memory updates. This evolving memory serves as a deterministic, structured record of the reasoning process, enabling transparent state tracking, controllable behavior, and bounded context growth.

To train the planner on-policy within this agentic system, we need to overcome the long-horizon credit assignment problem inherent to sparse, trajectory-level rewards. We introduce *Flow-based Group Refined Policy Optimization* (Flow-GRPO, Figure 2), an on-policy algorithm designed for this setting. Flow-GRPO operates on *in-the-flow* rollouts, which capture the full trajectory of states, actions, and tool events induced by the live system. Instead of attempting to assign credit with brittle, intermediate heuristics, we assign a single, verifiable final-outcome reward to the entire trajectory and *broadcast* it to every turn. This design effectively transforms the multi-turn reinforcement learning challenge into a series of single-turn updates: at each turn, the planner has access to the full memory context and receives a consistent reward signal aligned with global success. This approach, coupled with group-normalized advantages to stabilize training, enables robust credit assignment and allows the planner to learn effective long-horizon strategies from sparse feedback.

We evaluate AGENTFLOW on ten benchmarks across diverse reasoning domains. AGENTFLOW substantially outperforms top-performing specialized tool-integrated reasoning models and agentic systems, achieving average accuracy by 14.9% on knowledge-intensive search, 14.0% on broader agentic tasks, 14.5% on mathematical problem-solving, and 4.1% on scientific reasoning (§4.2). Notably, our 7B-backbone system even surpasses the ∼200B-parameter GPT-4o (Hurst et al., 2024) across all domains. Further analyses confirm that our in-the-flow optimization with Flow-GRPO is crucial, far surpassing offline supervised tuning (§4.3). The trained planner learns to optimize planning, enhance tool-calling reliability, and autonomously discover effective solution pathways (§4.4).

Moreover, our training approach proves highly efficient, leading to increased rewards and condensed responses compared to traditional tool-integrated RL methods (§4.5). Finally, we demonstrate that these benefits generalize, with consistent gains from scaling both the backbone model size and the inference-time turn budget (§4.6).

Our work makes three key contributions: (1) We present AGENTFLOW, a trainable agentic system that optimizes its planner directly *in the flow* of a multi-module workflow. By coordinating specialized modules through an evolving memory, it enables adaptive long-horizon planning and robust tool orchestration. (2) We introduce *Flow-GRPO*, an on-policy, outcome-driven reinforcement learning method designed to solve the sparse, long-horizon credit assignment problem in agentic systems. It stabilizes training by broadcasting a single final-outcome reward and using group-normalized advantages. (3) We demonstrate through comprehensive experiments that AGENTFLOW with a 7B backbone outperforms specialized baselines and larger proprietary models on diverse reasoning tasks. Further analyses highlight optimized planning, training efficiency, and scaling trends.

2 PRELIMINARY

Reinforcement learning for reasoning LLMs. Recent progress in reasoning LLMs has been significantly driven by reinforcement learning from outcome feedback, using a verifiable reward signal (Shao et al., 2024; Yu et al., 2025). This paradigm fine-tunes a language model to maximize an outcome-based reward while remaining close to a reference policy. Formally, the objective is to optimize a policy LLM π_θ to generate a response o for a given query q from dataset $\mathcal{D}$:

$$\max_{\pi_\theta} \mathbb{E}_{x\sim\mathcal{D},\, o\sim\pi_\theta(\cdot|q)}\big[R(q,o)\big] - \beta\, \mathbb{D}_{\mathrm{KL}}(\pi_\theta(o \mid q) \,\|\, \pi_{\mathrm{ref}}(o \mid q)), \tag{1}$$

where $R(q,o)$ is the verifiable outcome-based reward, π_{ref} is a reference model to prevent policy collapse, and β controls KL regularization. Algorithms like Group Policy Optimization (GRPO) (Shao et al., 2024) implement this by sampling groups of responses, normalizing advantages by their rewards, and updating the policy with a clipped objective to encourage high-reward outputs.

Tool-integrated reasoning models (LLM agents). LLMs can be augmented with external tools to access knowledge and perform precise computation under reinforcement learning with outcome-based reward. As shown in Figure 8(a), the LLM *interleaves* reasoning and tool calls, producing a chain of thought within <think></think> tokens followed by tool invocations (e.g., <tool_call></tool_call>). The resulting trajectory τ is a sequence of model generations and tool observations: $\tau = \{s^1, a^1, e^1, \ldots, s^T, a^T\}$, where s^t denotes the context, a^t the generated action (thought + tool call), and e^t the tool's execution result. The policy model π_θ is then trained to maximize a final outcome reward. Prior work has explored single- and multi-tool settings for search and code execution (Jin et al., 2025; Chen et al., 2025; Feng et al., 2025; Qian et al., 2025a).

Agentic systems with tool usage. An alternative approach is the use of agentic systems (Wu et al., 2024; Hong et al., 2024; Lu et al., 2025). As shown in Figure 8(b), these frameworks deploy multiple specialized modules—often distinct LLMs with carefully designed prompts and roles—within a collaborative workflow. By decomposing tasks and assigning subproblems to modules with dedicated tools and capabilities (e.g., planner, coder, critic), they can address complex problems such as web browsing, document processing, and multi-stage programming that exceed the scope of a single model. A central limitation, however, is that these systems are typically *training-free*: modules remain frozen pre-trained models orchestrated by handcrafted logic or prompting heuristics.

3 IN-THE-FLOW AGENTIC SYSTEM OPTIMIZATION

We aim to bridge the gap between trainable but monolithic reasoning models and flexible yet static agentic systems. We present AGENTFLOW, a flexible and trainable agentic system that integrates four specialized modules with an evolving memory (§3.1). Unlike prior agentic systems, AGENTFLOW directly optimizes the planner *within* the multi-turn loop of an agentic system (§3.2).

3.1 AGENTFLOW: AN IN-THE-FLOW AGENTIC SYSTEM

We propose AGENTFLOW, a general-purpose tool-integrated agentic framework for solving complex reasoning tasks through fine-grained planning and effective tool use within a multi-turn architecture. As shown in Figure 1, the framework comprises four specialized modules—**Action Planner**

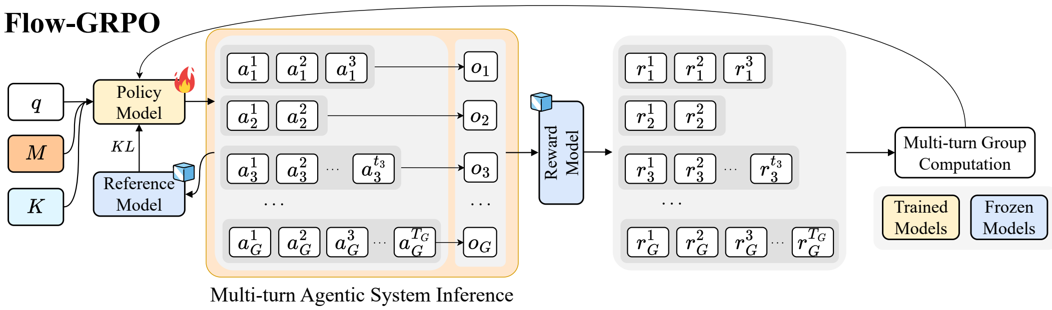


Figure 2: **Optimization for our proposed agentic system AGENTFLOW.** Given a query q, an evolving memory M, and a toolset K, the policy model generates actions that target sub-goals and select tools. It is trained via *Flow-based Group Refined Policy Optimization* (Flow-GRPO), which enables multi-turn reinforcement learning and stable optimization under collaborative dynamics.

$\mathcal{P}$, **Tool Executor** $\mathcal{E}$, **Execution Verifier** $\mathcal{V}$, and **Solution Generator** $\mathcal{G}$—coordinated by a shared evolving memory M and a toolset K. These modules interact sequentially and iteratively to perform *action planning*, *tool execution*, *context verification*, and *solution generation*, thereby enabling tool-integrated reasoning across multiple turns.

We formalize AGENTFLOW's problem-solving process as a multi-turn Markov Decision Process (MDP). Given a query q and a toolset K, the system proceeds for a variable number of turns. Let M^t denote the memory state before turn t (with M^1 initialized from q). At turn t, the planner $\mathcal{P}$ (a trainable policy π_θ) formulates a sub-goal, selects an appropriate tool $k \in K$, and retrieves relevant context from memory, producing an action: $a^t \sim \pi_\theta(a^t \mid q, K, M^t)$.

The executor $\mathcal{E}$ invokes the chosen tool with context, yielding an execution observation $e^t \sim \mathcal{E}(e^t \mid a^t, K)$. The verifier $\mathcal{V}$ then evaluates whether e^t is valid and whether the accumulated memory is sufficient to solve the query, producing a binary verification signal $v^t \sim \mathcal{V}(v^t \mid q, e^t, M^t)$. If $v^t = 0$, the memory is updated deterministically to incorporate new evidence: $M^{t+1} = f_{\text{mem}}(M^t, a^t, e^t, v^t)$, where $f_{\text{mem}}(\cdot)$ denotes the memory-update function, which records agent-process information in a concise, structured form along with contextual details such as time, turn index, and error signals.

The process repeats until $v^t = 1$ (termination) or a predefined maximum turn budget is reached. Upon termination at turn T, the solution generator $\mathcal{G}$ produces the final solution o, conditioned on the query and the accumulated memory: $o \sim \mathcal{G}(o \mid q, M^T)$.

This formulation decomposes multi-turn, tool-integrated reasoning into structured, observable transitions. After T turns, the trajectory $\tau = \{(a^t, e^t, v^t)\}_{t=1}^T$ records the history of planning, execution, and verification. The joint generative process can be written as

$$p_\theta\Big(\{a^t, e^t, v^t\}_{t=1}^T,\, o \mid q\Big) = \left[\prod_{t=1}^{T} \pi_\theta(a^t \mid q, K, M^t)\, \mathcal{E}(e^t \mid a^t, K)\, \mathcal{V}(v^t \mid q, e^t, M^t)\right] \mathcal{G}(o \mid q, M^T), \quad (2)$$

where $\{a^t, e^t, v^t\}_{t=1}^T$ are explicit realizations of the latent reasoning chain. Importantly, unlike latent thoughts behind trajectories, our memory M is an explicit and deterministic record of the reasoning process, ensuring transparency and controllability of multi-turn decisions.

3.2 IN-THE-FLOW REINFORCEMENT LEARNING OPTIMIZATION

We target tool-integrated *agentic systems* operating under *long-horizon* tasks with *sparse* rewards. In this setting, the **Action Planner** (the trainable policy of AGENTFLOW) selects a *sequence* of interdependent actions while the state (q, K, M^t) evolves with tool results and verifier feedback. Conventional *offline* training—e.g., supervised fine-tuning or preference fine-tuning on curated traces—optimizes the planner *outside* the active loop (Motwani et al., 2024; Park et al., 2025). This decoupling prevents real-time coordination with the executor, verifier, and solution generator, induces distribution shift between training and deployment, and provides limited guidance about *which* intermediate decisions truly matter. As a result, planners often adapt poorly to multi-turn dynamics; early errors cascade, and post-hoc fixes are brittle.

In-the-flow learning. To address these issues, we optimize the planner *in the flow* of execution. We roll out the full AGENTFLOW system under the current policy, collect the actual trajectory τ of states, actions, and tool events it induces, and update the policy within the agentic system using a verifiable final-outcome signal. This exposes the multi-turn credit-assignment problem directly and trains the planner on the exact states it will face at inference. Our objective, Flow-GRPO, is designed to stabilize learning under sparse, trajectory-level rewards over multiple turns.

As established in §3.1, rollouts in AGENTFLOW define a finite-horizon MDP with a variable horizon T. At turn t, the planner observes the state (q, K, M^t), selects an action a^t, the executor and verifier return (e^t, v^t), and the memory updates deterministically to M^{t+1}.

Policy optimization objective. The planner policy π_θ is trained to maximize the expected return over on-policy rollouts. Let $R(\tau)$ be the reward for a complete trajectory τ. The objective is:

$$\mathcal{J}(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\big[R(\tau)\big], \qquad \theta^\star = \arg\max_\theta \mathcal{J}(\theta), \tag{3}$$

where a rollout τ is the sequence of decisions $\{a^t\}_{t=1}^T$ generated on-policy by π_θ.

Final-outcome reward. Assigning credit to intermediate actions is challenging because each a^t influences the final solution only indirectly, and their value may only emerge after several turns (e.g., error or improvement accumulation). To avoid brittle local feedback, we adopt a *final-outcome-based reward*: every action within a rollout receives the same global reward signal, based on the correctness of the final solution o with respect to query q and ground truth y^*:

$$r = R(a^t) = \bar{R}(o, q, y^*), \quad \forall t = 1, \ldots, T, \tag{4}$$

where $\bar{R}(o, q, y^*) \in \{0, 1\}$ is assigned by an LLM-as-judge rubric for semantic, numeric, and option-level equivalence (see §F.3). This propagates a trajectory-level success signal back through the reasoning chain, aligning every decision a^t with global correctness.

Objective function. We formalize **Flow**-based **G**roup **R**efined **P**olicy **O**ptimization for the planner. The goal is to optimize the policy π_θ by maximizing the expected return over a group of parallel rollouts. For each query-label pair from training corpus $(q, y^*) \sim \mathcal{D}$, we sample a group of G on-policy trajectories $\{\tau_i\}_{i=1}^G$ by running the current behavior policy $\pi_{\theta_{\text{old}}}$ inside AGENTFLOW, where $\tau_i = \{a_i^1, \ldots a_i^{T_i}, o_i\}$. Let $s_i^t = (q, K, M_i^t)$ be the state at turn t of rollout i, a_i^t the planner's action (a token sequence of length $|a_i^t|$), and o_i the final response. This structure is key to addressing the long-horizon credit assignment challenge: by broadcasting a single trajectory-level reward to all turns, we effectively decompose the *multi-turn RL* problem into *a set of independent, single-turn* policy updates; we provide a formal proof of this equivalence and analyze its convergence properties in §C. Each update for an action a_i^t is conditioned on the full historical context encapsulated in the state s_i^t and receives the same global success signal, simplifying optimization. The objective is

$$\mathcal{J}_{\text{Flow-GRPO}}(\theta) = \mathbb{E}_{(q,y^*)\sim\mathcal{D},\ \{\tau_i\}_{i=1}^G\sim\pi_{\theta_{\text{old}}}}$$
$$\left[\frac{1}{G}\sum_{i=1}^{G}\frac{1}{T_i}\sum_{t=1}^{T_i}\frac{1}{|a_i^t|}\sum_{j=1}^{|a_i^t|}\min\Big\{\rho_{i,j}^t A_i^t,\ \text{clip}(\rho_{i,j}^t,\ 1-\epsilon,\ 1+\epsilon)\, A_i^t\Big\}\ -\ \beta\,\mathbb{D}_{\text{KL}}\big(\pi_\theta \,\|\, \pi_{\text{ref}}\big)\right], \tag{5}$$

where T_i is the (variable) number of turns in rollout i, and

$$\rho_{i,j}^t = \frac{\pi_\theta\big(a_{i,j}^t \mid s_i^t, a_{i,1:j-1}^t\big)}{\pi_{\theta_{\text{old}}}\big(a_{i,j}^t \mid s_i^t, a_{i,1:j-1}^t\big)} \tag{6}$$

is the token-level importance ratio for the j-th token of a_i^t, $\epsilon > 0$ is the PPO clipping parameter, and $\beta > 0$ controls the KL penalty to a fixed reference policy π_{ref}.

Group-normalized advantages. Because the reward in Eq. 4 is a single trajectory-level signal, the per-turn advantage A_i^t is constant over t within a rollout i. We reduce variance and sharpen credit assignment across the group by using a *group-normalized* advantage:

$$A_i^t = \frac{\bar{R}(o_i, q, y^*) - \text{mean}\left(\{\bar{R}(o_k, q, y^*)\}_{k=1}^G\right)}{\text{std}\left(\{\bar{R}(o_k, q, y^*)\}_{k=1}^G\right)}. \tag{7}$$

Summary. Flow-GRPO performs on-policy, in-the-flow optimization of the planner inside the active multi-turn loop of AGENTFLOW. By broadcasting a single trajectory-level outcome to every turn and normalizing advantages across parallel rollouts, it effectively transforms the long-horizon credit assignment problem into a series of tractable, single-turn updates. Each update is conditioned on the explicit memory state and optimized using a token-level PPO objective, with KL regularization and clipping to further stabilize learning. This conversion enables effective optimization under sparse, trajectory-level rewards, addressing a central challenge in training tool-integrated agentic systems.

		Search Intensive						Agentic	
Model	**Size**	**Bamboogle**	**2Wiki**	**HotpotQA**	**Musique**	**Avg.**	**Δ**	**GAIA**	**Δ**
Qwen-2.5-7B-Instruct	7B-Inst	12.0	23.0	21.0	6.0	15.5	↑ 41.8	3.2	↑ 29.9
Qwen-2.5-14B-Instruct	14B-Inst	21.6	26.7	20.0	8.0	19.1	↑ 38.2	5.5	↑ 27.6
Qwen-2.5-32B-Instruct	32B-Inst	24.0	26.7	27.0	6.0	20.9	↑ 36.4	9.5	↑ 23.6
Llama-3.3-70B-Instruct	70B-Inst	18.4	22.7	52.0	16.0	27.3	↑ 30.0	3.2	↑ 29.9
GPT-4o-mini (Hurst et al., 2024)	∼8B	40.8	35.6	41.0	15.0	33.1	↑ 24.2	7.1	↑ 26.0
GPT-4o (Hurst et al., 2024)	∼200B	68.8	49.5	54.0	24.0	49.1	↑ 8.2	17.3	↑ 15.8
Supervised Fine-Tuning (SFT)	7B-Inst	12.0	25.9	22.0	6.6	16.6	↑ 40.7	3.2	↑ 29.9
Iter-RetGen (Shao et al., 2023)	7B-Inst	36.8	33.6	37.4	17.8	31.4	↑ 25.9	3.9	↑ 29.2
Search-R1 (Jin et al., 2025)	7B-Inst	43.2	38.2	37.0	14.6	33.3	↑ 24.0	19.1	↑ 14.0
ZeroSearch (Sun et al., 2025)	7B-Base	27.8	35.2	34.6	18.0	28.9	↑ 28.4	16.5	↑ 16.6
ReSearch (Chen et al., 2025)	7B-Base	42.4	47.6	43.5	22.3	39.0	↑ 18.3	17.3	↑ 15.8
StepSearch (Wang et al., 2025d)	7B-Base	40.0	36.6	38.6	22.6	34.5	↑ 22.8	–	–
VerlTool (Jiang et al., 2025)	7B-Base	46.4	45.3	44.8	19.3	39.0	↑ 18.3	11.2	↑ 21.9
AutoGen (Wu et al., 2024)	7B-Inst	59.6	44.0	50.0	15.9	42.4	↑ 14.9	6.3	↑ 26.8
AgentFlow	7B-Inst	58.4	60.0	51.3	19.2	47.2	↑ 12.1	17.2	↑ 15.9
AgentFlow (w/ Flow-GRPO)	7B-Inst	**69.6**	**77.2**	**57.0**	**25.3**	**57.3**	–	**33.1**	–

Table 1: **Performance comparison on search-intensive and agentic tasks.** 7B-Base refers to Qwen-2.5-7B-Base and 7B-Inst refers to Qwen-2.5-7B-Instruct. AutoGen and our AgentFlow method are agentic systems, which use Qwen-2.5-7B-Instruct for the LLM-powered agents and tools for fair comparison. We visualize the gains of AgentFlow to the each baseline in the Δ columns.

		Math Reasoning					Scientific Reasoning			
Model	**Size**	**AIME24**	**AMC23**	**GameOf24**	**Avg.**	**Δ**	**GPQA**	**MedQA**	**Avg.**	**Δ**
Qwen-2.5-7B-Instruct	7B-Inst	6.7	47.5	33.0	29.1	↑ 22.5	34.0	66.0	50.0	↑ 13.5
Qwen-2.5-14B-Instruct	14B-Inst	6.7	60.0	25.0	30.6	↑ 21.0	31.0	75.0	53.0	↑ 10.5
Llama-3.3-70B-Instruct	70B-Inst	6.7	47.5	31.0	28.4	↑ 23.1	35.0	67.0	51.0	↑ 12.5
Llama-3.1-405B-Instruct	405B-Inst	26.7	47.5	23.0	32.4	↑ 19.1	30.0	62.0	46.0	↑ 17.5
GPT-4o-mini (Hurst et al., 2024)	∼8B	13.3	57.5	16.0	28.9	↑ 22.6	27.0	66.0	46.5	↑ 17.0
GPT-4o (Hurst et al., 2024)	∼200B	13.3	60.0	32.0	35.1	↑ 16.4	31.0	60.0	45.5	↑ 18.0
Supervised Fine-Tuning (SFT)	7B-Inst	6.7	47.5	33.0	29.1	↑ 22.5	34.0	66.0	50.0	↑ 13.5
SimpleRL-reason (Zeng et al., 2025b)	7B-Base	16.7	60.0	33.0	36.6	↑ 15.0	45.0	65.0	50.0	↑ 13.5
Open-Reasoner-Zero (Hu et al., 2025a)	7B-Base	16.7	54.9	32.0	34.5	↑ 17.0	34.0	54.0	44.0	↑ 19.5
General-Reasoner (Ma et al., 2025)	7B-Base	13.3	55.0	33.0	33.8	↑ 17.7	35.5	61.0	48.3	↑ 15.2
Luffy (Yan et al., 2025)	7B-Inst	30.7	44.8	33.0	36.2	↑ 15.3	34.0	77.0	55.5	↑ 8.0
TIR (Yang et al., 2024b)	7B-Inst	10.0	50.0	33.0	31.0	↑ 20.5	42.0	76.8	59.4	↑ 4.1
ToRL (Li et al., 2025b)	7B-Inst	20.0	60.0	31.0	37.0	↑ 14.5	35.0	76.5	55.8	↑ 7.7
AutoGen (Wu et al., 2024)	7B-Inst	13.3	57.5	24.0	31.6	↑ 19.9	42.0	72.0	57.0	↑ 6.5
AgentFlow	7B-Inst	16.7	47.4	31.0	31.7	↑ 19.8	37.0	76.0	56.5	↑ 7.0
AgentFlow (w/ Flow-GRPO)	7B-Inst	**40.0**	**61.5**	**53.0**	**51.5**	–	**47.0**	**80.0**	**63.5**	–

Table 2: **Performance comparison of mathematical and scientific reasoning tasks.**

4 Experiments

4.1 Experimental Setup

In our main experiments, all modules—Action Planner, Tool Executor, Executive Verifier, and Solution Generator—are instantiated with the *Qwen2.5-7B-Instruct* model (Yang et al., 2024a). Among these, only the *Action Planner* is trainable. The system operates with five interactive tools: *Base Generator* is an instance of *Qwen2.5-7B-Instruct* that acts as the default reasoning engine if the planner decides not to use an external tool; *Python Coder* generates and executes Python code given a query and returns the execution result; *Google Search* searches the web and returns a summarization of Top-K search results; *Wikipedia Search* searches articles matching a given query and returns a summarization; and *Web Search* returns summarized information from a given web page. During the RL fine-tuning phase, we mix data from Search-R1 (Jin et al., 2025) and DeepMath (He et al., 2025) as training data, which provides paired question-answer examples across search and mathematical domains. We use a batch size of 32 with 8 rollouts per sample.

To comprehensively evaluate tool-use capabilities of AgentFlow, we conduct experiments on four types of reasoning tasks: (1) *Knowledge-intensive search* including Bamboogle (Press et al., 2023), 2Wiki (Ho et al., 2020), HotpotQA (Yang et al., 2018), and Musique (Trivedi et al., 2022); (2) *Agentic reasoning* such as GAIA (Mialon et al., 2023) (where we adopt the textual split); (3) *Logic-dense mathematical reasoning* including AIME2024 (Art of Problem Solving, 2025), AMC23 (MAA, 2023), and GameOf24 (Lightman et al., 2023); and (4) *Scientific reasoning* including GPQA (Rein et al., 2024) and MedQA (Yang et al., 2024c). To mitigate randomness, we report the average accuracy across three trials for all experiments. More experimental details are in §D.

4.2 MAIN RESULTS

Baselines. As presented in Tables 1 and 2, we include five categories of baselines: (1) *Open-source LLMs*: Qwen2.5 (Yang et al., 2024a), Llama-3.1, and Llama-3.3 (Dubey et al., 2024); (2) *Proprietary LLMs*: GPT-4o-mini and GPT-4o; (3) *Reasoning LLMs*: supervised fine-tuning (Yang et al., 2024b), SimpleRL-reason, Open-Reasoner-Zero, General-Reasoner, and LUFFY; (4) *Tool-integrated reasoning LLMs*: both search-enhanced, including Iter-RetGen, Search-R1, ZeroSearch, ReSearch, StepSearch, and VerlTool, and code-enhanced, including TIR and ToRL; (5) *Training-free agentic system*: AutoGen. More details on baseline implementations are in §D.3.

Key insights. Overall, AGENTFLOW consistently outperforms all baseline models. The advantages are significant: compared to the best-performing 7B models without tool integration, AGENTFLOW achieves absolute gains of up to 41.8% on search, 29.9% on agentic reasoning, and 15.0% on math. Even against specialized tool-integrated models, AGENTFLOW maintains a decisive lead, outperforming the top models by 18.3% in search, 14.0% on agentic reasoning, 15.6% in math, and 7.0% in science. Highlighting the efficiency of our agentic approach, our 7B-backbone AGENTFLOW even outperforms the ∼200B parameter GPT-4o across all domains, with advantages ranging from 8.2% to 18.0%. A detailed analysis follows in §E.1.

4.3 TRAINING STRATEGIES ON THE PLANNER

We conduct an ablation study to analyze the impact of different training strategies for the *Action Planner* module in AGENTFLOW, with results reported in Table 3. The executor, verifier, and generator modules remain fixed as Qwen2.5-7B-Instruct, consistent with our main setup (§4.1).

Planner Model	Training	Bamboogle	2Wiki	GAIA	AIME24	AMC23	GameOf24	Avg.
Qwen-2.5-7B	Frozen	58.4	60.0	17.2	16.7	47.4	31.0	38.5
GPT-4o	Frozen	65.0 ↑ 6.6	70.0 ↑ 10.0	23.6 ↑ 6.4	16.7 ↑ 0.0	48.7 ↑ 1.3	42.0 ↑ 11.0	44.3 ↑ 5.8
Qwen-2.5-7B	SFT	30.4 ↓ 28.0	32.7 ↓ 27.3	6.3 ↓ 10.9	3.3 ↓ 13.4	37.5 ↓ 9.9	7.0 ↓ 24.0	19.5 ↓ 19.0
Qwen-2.5-7B	Flow-GRPO	69.6 ↑ 11.2	77.2 ↑ 17.2	33.1 ↑ 15.9	40.0 ↑ 23.3	61.5 ↑ 14.1	53.0 ↑ 22.0	55.7 ↑ 17.2

Table 3: Performance comparison of AGENTFLOW across different training methods.

A more capable planner is beneficial, but has limits. Replacing the frozen *Qwen2.5-7B-Instruct* baseline with a stronger proprietary model, GPT-4o, yields only a modest 5.8% average gain. This indicates a key bottleneck that, while a more powerful model improves planning, its static nature prevents co-adaptation with the live dynamics of AGENTFLOW.

Offline SFT leads to performance collapse, while in-the-flow RL is crucial. The limitations of a static planner are further exposed when distilling GPT-4o's behavior via offline supervised fine-tuning (SFT) on its trajectories as *Action Planner* in AGENTFLOW. This results in a catastrophic performance collapse, with an average accuracy drop of 19.0% compared to the frozen baseline. This failure arises from the token-level imitation objective of SFT, which misaligns with trajectory-level task success and prevents the planner from adapting to dynamic tool feedback or recovering from compounding errors. In contrast, training the planner with our on-policy Flow-GRPO method proves highly effective: by optimizing for the final outcome, the planner learns to handle long-horizon workflows, achieving a 17.2% average gain over the frozen baseline.

4.4 IN-DEPTH ANALYSIS OF OPTIMIZED PLANNING

Flow-GRPO optimizes tool usage. We compare tool usage distributions before and after in-the-flow RL training. Figure 3 shows results on two knowledge-intensive tasks, 2Wiki and MedQA, which exhibit distinct optimization patterns alongside improved task accuracy. For 2Wiki, which requires broad factual knowledge, Flow-GRPO optimizes the planner to increase Google Search usage by 42%. In contrast, for the specialized MedQA benchmark, which requires deep, domain-specific information retrieval, fine-tuning shifts the planner away from general tools, reducing Google Search calls (66.2→10.9%) in favor of in-document Web Search (0→19.5%) and specialized Wikipedia Search (0→59.8%). This demonstrates that the planner learns to select task-appropriate tools.

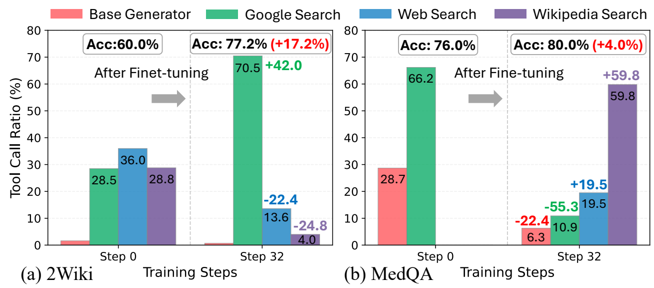


Figure 3: Tool call ratio change by Flow-GRPO fine-tuning.

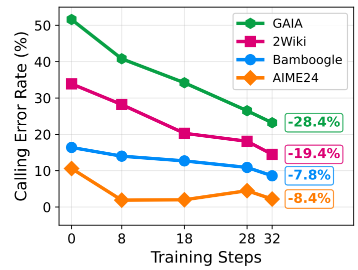


Figure 4: Calling error rate.

Flow-GRPO enhances tool-calling efficacy. A key aspect of the model's improvement is its increased reliability in tool usage. As shown in Figure 4, the tool-calling error rate consistently decreases across tasks during training, with a reduction of up to 28.4% on AIME24. This trend indicates that the training process not only teaches the model *which* tool to use but also *how* to invoke it correctly with proper arguments and format, leading to more robust and effective tool integration.

Flow-GRPO incentivizes autonomous discovery of new solutions. We further investigate qualitative examples in §G. These cases show that AGENTFLOW, trained with Flow-GRPO, develops enhanced capabilities for task planning and tool use. The planner exhibits adaptive efficiency, more stable self-correction, and dynamic tool selection during the step-by-step problem-solving process, autonomously discovering effective solution pathways.

4.5 TRAINING EFFICIENCY ANALYSIS

Optimized planning with increased rewards and condensed responses. We analyze the training dynamics of the AGENTFLOW planner on AIME24 by tracking its average reward and response length (Figure 5a). Training rewards steadily increase, indicating effective policy improvement via Flow-GRPO. Meanwhile, response length, after an initial exploratory rise, progressively shortens and stabilizes. This shows the planner learns to balance conciseness and informativeness, avoiding unnecessarily long outputs.

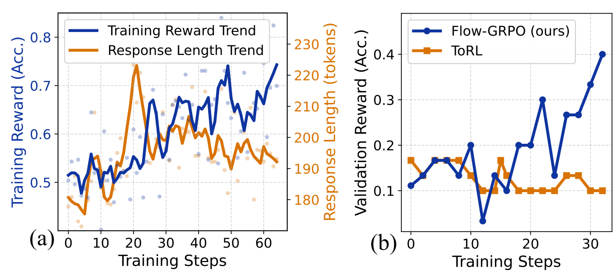


Figure 5: Training dynamics and efficiency of Flow-GRPO.

Flow-GRPO efficiency over tool-integrated reasoning RL. We compare AGENTFLOW (trained with Flow-GRPO) against a monolithic tool-integrated reasoning baseline (ToRL) on AIME24. As shown in Figure 5(b), AGENTFLOW achieves sustained performance gains, with validation accuracy growing steadily. In contrast, ToRL's performance quickly stagnates and trends downwards, highlighting the superior efficiency of our agentic training approach, which uses decomposition and stable credit assignment to avoid the instability.

4.6 SCALING TRENDS IN AGENTFLOW

Training scaling in backbone size. We study how backbone LLM scale affects AGENTFLOW's performance and the efficacy of Flow-GRPO. We build two versions of the system: one using *Qwen2.5-3B-Instruct* and another using *Qwen2.5-7B-Instruct* for all four modules (planner, executor, verifier, and generator) and tools. In both, only the planner is fine-tuned with Flow-GRPO. As shown in Figure 6, Flow-GRPO fine-tuning consistently improves performance across tasks for both backbones. This demonstrates that our in-the-flow optimization is effective across model capacities, enhancing AGENTFLOW regardless of LLM size.

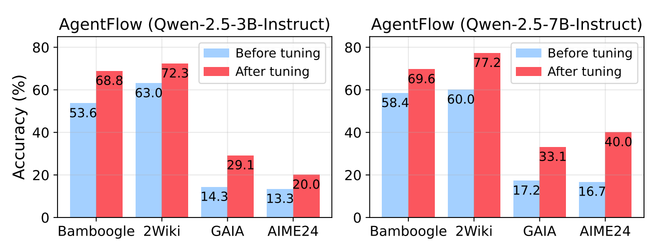


Figure 6: Flow-GRPO fine-tuning offers consistent gains on AGENTFLOW as the backbone model size scales from 3B to 7B.

Inference scaling in turn budgets. We investigate how the maximum allowed turns (T_{max}) affect reasoning depth and final performance of AGENTFLOW during test-time inference with the Qwen2.5-7B-Instruct backbone. As shown in Figure 7, increasing T_{max} from 3 to 10 consistently improves outcomes across all tasks, accompanied by a rise in average turns consumed. On knowledge-intensive benchmarks such as 2Wiki and GAIA, a larger turn budget enables AGENTFLOW for deeper information retrieval. On mathematical benchmarks like GameOf24 and AIME24, it supports decomposed sub-goals, alternative strategies, and refinement of errors. Final performance peaks at $T_{\text{max}} = 10$ for all tasks, confirming that a longer reasoning horizon benefits the system without causing degenerate loops. This validates that AGENTFLOW adapts its turn allocation to problem complexity to achieve better solutions through iterative refinement.

Turns (T_{max})	**3**	**5**	**7**	**10**
2Wiki	2.22	3.18	3.81	4.44
GameOf24	1.63	2.12	2.36	2.67
AIME24	1.63	1.63	1.86	1.90
GAIA	2.43	3.46	4.28	5.42

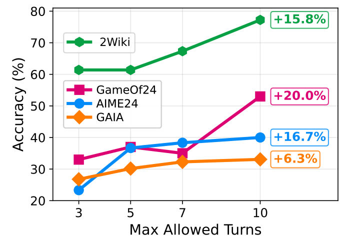


Figure 7: Average turns and accuracy with increased T_{max}.

5 RELATED WORK

Reinforcement learning (RL) from outcome-based rewards has become a dominant paradigm for training LLMs to use external tools. Much of this work trains a single, monolithic policy to interleave reasoning with tool calls. This strategy has proven effective in specialized, single-tool settings, such as code execution for mathematical problems (Mai et al., 2025; Xue et al., 2025; Feng et al., 2025; Li et al., 2025b) and web search for knowledge-intensive questions (Chen et al., 2025; Jin et al., 2025; Song et al., 2025; Li et al., 2025a; Sun et al., 2025). Recent efforts have extended this monolithic framework to multi-tool environments by focusing on data synthesis (Dong et al., 2025), unified training infrastructure (Jiang et al., 2025), and principled reward design (Qian et al., 2025a; Zhang et al., 2025). However, this monolithic approach scales poorly as task complexity and planning horizons grow. The central challenge is long-horizon credit assignment; attributing a final outcome to specific intermediate tool calls remains difficult, even with fine-grained, turn-level rewards (Zeng et al., 2025a; Wang et al., 2025d). This difficulty leads to training instability and brittle inference-time generalization, manifesting as strategic deficiencies like tool overuse or "cognitive offloading" (Wang et al., 2025b; Qian et al., 2025b), suboptimal personalization (Cheng et al., 2025), and poor alignment with user preferences for tool invocation (Huang et al., 2025).

Agentic systems with tool use. Agentic systems offer an alternative to monolithic models by decomposing tasks across specialized modules. Many such systems are training-free, orchestrating pre-trained LLMs with handcrafted logic and prompting, as seen in frameworks like AutoGen (Wu et al., 2024), MetaGPT (Hong et al., 2024), and OctoTools (Lu et al., 2025). This static approach, however, limits their ability to learn and adapt collaborative strategies from experience. Recognizing this, recent work explores training these systems to improve coordination (Deng et al., 2025; Liao et al., 2025). However, most training paradigms are *offline*, relying on supervised fine-tuning or preference optimization on static datasets (Motwani et al., 2024; Park et al., 2025). These methods are decoupled from the live, multi-turn dynamics of the system, preventing modules from learning to adapt to evolving tool outputs or recover from early mistakes. Training directly *in the flow* with on-policy RL is difficult due to sparse rewards and long-horizon credit assignment, where feedback is delayed across long reasoning chains and shifting state distributions (Wang et al., 2025c). Consequently, these systems often suffer from brittle adaptation and require complex reward shaping to learn effectively (Wang et al., 2025a).

6 CONCLUSION

We presented AGENTFLOW, a trainable, *in-the-flow* agentic system that coordinates four specialized modules through an evolving memory, and optimizes its planner directly *inside* the multi-turn loop. To enable stable on-policy learning under long-horizon reasoning with sparse rewards, we introduced Flow-GRPO, which broadcasts a single, verifiable trajectory-level outcome to all decisions and uses group-normalized advantages for robust credit assignment. Comprehensive experiments demonstrate AGENTFLOW's strong cross-domain performance, surpassing specialized baselines and even larger proprietary models. In-depth analyses confirm the method improves planning effectiveness and tool reliability, with clear scaling benefits.

ETHICS STATEMENT

We affirm compliance with the ICLR Code of Ethics. Our research exclusively utilizes publicly available benchmarks, and our methodology does not involve human subjects, personally identifiable information, or proprietary user data. We adhere to the licensing and usage terms of all datasets employed in this study. The agentic system interacts with external tools, for which we have implemented safeguards to ensure responsible use. Web-based tools, such as Google Search and Wikipedia Search, are used solely to access public information while respecting platform terms of service and API rate limits. All code execution is performed within a sandboxed local environment with restricted network access to mitigate the security risks of executing model-generated code.

We acknowledge two primary ethical considerations. First, the use of an LLM-as-judge for reward signaling could introduce or amplify biases. To mitigate this, we employ a structured, rubric-based evaluation protocol, report results averaged over multiple random seeds to ensure robustness, and conduct detailed analyses of failure modes. Second, advanced agentic systems pose a risk of misuse in harmful automation. To address this, our work and the released codebase are intentionally focused on benign research domains (e.g., mathematics, scientific reasoning). We document the intended scope and limitations to discourage misuse.

In the interest of transparency and research integrity, we will release our codebase, model prompts, and experimental configurations to support reproducibility. The authors declare no conflicts of interest. All funding sources and affiliations will be fully disclosed in the camera-ready version.

REPRODUCIBILITY STATEMENT

To ensure the reproducibility of our work, we provide comprehensive documentation and resources. Our full codebase, including end-to-end scripts for training and evaluation, is available at `https://anonymous.4open.science/r/agentflow`. This repository contains all configuration files (hyperparameters, model IDs, rollout settings), prompt templates for the planner, executor, verifier, and generator modules (§F.1), tool metadata (§F.2), and the LLM-as-judge evaluation rubric (§F.3). Our experimental setup, including datasets, baselines, and evaluation protocols, is detailed in §4, with specific implementation details provided in §D and §D.2. For our theoretical contributions, a mathematical analysis of Flow-GRPO, including proofs and convergence guarantees, is presented in §C.

REFERENCES

Art of Problem Solving. Aime problems and solutions, 2025. URL `https://artofproblemsolving.com/wiki/index.php/AIME_Problems_and_Solutions`. 6, 21

Mingyang Chen, Tianpeng Li, Haoze Sun, Yijie Zhou, Chenzheng Zhu, Haofen Wang, Jeff Z Pan, Wen Zhang, Huajun Chen, Fan Yang, et al. ReSearch: Learning to reason with search for llms via reinforcement learning. *arXiv preprint arXiv:2503.19470*, 2025. 1, 3, 6, 9, 20

Zihao Cheng, Hongru Wang, Zeming Liu, Yuhang Guo, Yuanfang Guo, Yunhong Wang, and Haifeng Wang. ToolSpectrum: Towards personalized tool utilization for large language models. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 20679–20699, 2025. 9

Yingfan Deng, Anhao Zhou, Yuan Yuan, Xian Zhang, Yifei Zou, and Dongxiao Yu. Pe-ma: Parameter-efficient co-evolution of multi-agent systems. *arXiv preprint arXiv:2506.11803*, 2025. 9

Guanting Dong, Yifei Chen, Xiaoxi Li, Jiajie Jin, Hongjin Qian, Yutao Zhu, Hangyu Mao, Guorui Zhou, Zhicheng Dou, and Ji-Rong Wen. Tool-star: Empowering llm-brained multi-tool reasoner via reinforcement learning. *arXiv preprint arXiv:2505.16410*, 2025. 1, 9

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024. 7, 19

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. *arXiv preprint arXiv:2504.11536*, 2025. 1, 3, 9

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025. 1

Zhiwei He, Tian Liang, Jiahao Xu, Qiuzhi Liu, Xingyu Chen, Yue Wang, Linfeng Song, Dian Yu, Zhenwen Liang, Wenxuan Wang, et al. Deepmath-103k: A large-scale, challenging, decontaminated, and verifiable mathematical dataset for advancing reasoning. *arXiv preprint arXiv:2504.11456*, 2025. 6

Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. Constructing a multi-hop qa dataset for comprehensive evaluation of reasoning steps. In *Proceedings of the 28th International Conference on Computational Linguistics (COLING)*, pp. 6609–6625, 2020. 6, 21

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Ceyao Zhang, Jinlin Wang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, et al. MetaGPT: Meta programming for a multi-agent collaborative framework. In *International Conference on Learning Representations (ICLR)*, 2024. 1, 3, 9

Jingcheng Hu, Yinmin Zhang, Qi Han, Daxin Jiang, Xiangyu Zhang, and Heung-Yeung Shum. Open-reasoner-zero: An open source approach to scaling up reinforcement learning on the base model. *arXiv preprint arXiv:2503.24290*, 2025a. 6, 20

Mengkang Hu, Yuhang Zhou, Wendong Fan, Yuzhou Nie, Bowei Xia, Tao Sun, Ziyu Ye, Zhaoxuan Jin, Yingru Li, Qiguang Chen, et al. Owl: Optimized workforce learning for general multi-agent assistance in real-world task automation. *arXiv preprint arXiv:2505.23885*, 2025b. 1

Chengrui Huang, Shen Gao, Zhengliang Shi, Dongsheng Wang, and Shuo Shang. TTPA: Token-level tool-use preference alignment training framework with fine-grained evaluation. *arXiv preprint arXiv:2505.20016*, 2025. 9

Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. GPT-4o system card. *arXiv preprint arXiv:2410.21276*, 2024. 2, 6, 19

Dongfu Jiang, Yi Lu, Zhuofeng Li, Zhiheng Lyu, Ping Nie, Haozhe Wang, Alex Su, Hui Chen, Kai Zou, Chao Du, et al. VerlTool: Towards holistic agentic reinforcement learning with tool use. *arXiv preprint arXiv:2509.01055*, 2025. 6, 9, 20

Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan Arik, Dong Wang, Hamed Zamani, and Jiawei Han. Search-R1: Training llms to reason and leverage search engines with reinforcement learning. *arXiv preprint arXiv:2503.09516*, 2025. 1, 3, 6, 9, 19, 20

Di Jin, Eileen Pan, Nassim Oufattole, Wei-Hung Weng, Hanyi Fang, and Peter Szolovits. What disease does this patient have? a large-scale open domain question answering dataset from medical exams. *Applied Sciences*, 11(14):6421, 2021. 21

Xiaoxi Li, Guanting Dong, Jiajie Jin, Yuyao Zhang, Yujia Zhou, Yutao Zhu, Peitian Zhang, and Zhicheng Dou. Search-o1: Agentic search-enhanced large reasoning models. *arXiv preprint arXiv:2501.05366*, 2025a. 9

Xuefeng Li, Haoyang Zou, and Pengfei Liu. ToRL: Scaling tool-integrated rl. *arXiv preprint arXiv:2503.23383*, 2025b. 6, 9, 20

Junwei Liao, Muning Wen, Jun Wang, and Weinan Zhang. Marft: Multi-agent reinforcement fine-tuning. *arXiv preprint arXiv:2504.16129*, 2025. 9

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *The Twelfth International Conference on Learning Representations (ICLR)*, 2023. 6

Nathan Lile. Math twenty four (24s game) dataset. https://huggingface.co/datasets/nlile/24-game, 2024. 21

Pan Lu, Bowen Chen, Sheng Liu, Rahul Thapa, Joseph Boen, and James Zou. OctoTools: An agentic framework with extensible tools for complex reasoning. *arXiv preprint arXiv:2502.11271*, 2025. 3, 9

Xueguang Ma, Qian Liu, Dongfu Jiang, Ge Zhang, Zejun Ma, and Wenhu Chen. General-reasoner: Advancing llm reasoning across all domains. *arXiv preprint arXiv:2505.14652*, 2025. 6, 20

MAA. American mathematics competitions. In *American Mathematics Competitions*, 2023. 6, 21

Xinji Mai, Haotian Xu, Xing W, Weinong Wang, Yingying Zhang, and Wenqiang Zhang. Agent RL Scaling Law: Agent RL with Spontaneous Code Execution for Mathematical Problem Solving. *arXiv preprint arXiv:2505.07773*, 2025. 1, 9

Grégoire Mialon, Clémentine Fourrier, Thomas Wolf, Yann LeCun, and Thomas Scialom. Gaia: a benchmark for general ai assistants. In *The Twelfth International Conference on Learning Representations (ICLR)*, 2023. 6, 21

Moonshot AI. Kimi-Researcher: End-to-End RL Training for Emerging Agentic Capabilities. https://moonshotai.github.io/Kimi-Researcher/, June 2025. 1

Sumeet Ramesh Motwani, Chandler Smith, Rocktim Jyoti Das, Rafael Rafailov, Ivan Laptev, Philip HS Torr, Fabio Pizzati, Ronald Clark, and Christian Schroeder de Witt. Malt: Improving reasoning with multi-agent llm training. *arXiv preprint arXiv:2412.01928*, 2024. 2, 4, 9

Chanwoo Park, Seungju Han, Xingzhi Guo, A. Ozdaglar, Kaiqing Zhang, and Joo-Kyung Kim. MAPoRL: Multi-agent post-co-training for collaborative large language models with reinforcement learning. In *Annual Meeting of the Association for Computational Linguistics (ACL*, 2025. URL https://api.semanticscholar.org/CorpusId:276580906. 2, 4, 9

Ofir Press, Muru Zhang, Sewon Min, Ludwig Schmidt, Noah A Smith, and Mike Lewis. Measuring and narrowing the compositionality gap in language models. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 5687–5711, 2023. 6, 21

Cheng Qian, Emre Can Acikgoz, Qi He, Hongru Wang, Xiusi Chen, Dilek Hakkani-Tür, Gokhan Tur, and Heng Ji. ToolRL: Reward is all tool learning needs. *arXiv preprint arXiv:2504.13958*, 2025a. 1, 3, 9

Cheng Qian, Emre Can Acikgoz, Hongru Wang, Xiusi Chen, Avirup Sil, Dilek Hakkani-Tür, Gokhan Tur, and Heng Ji. SMART: Self-aware agent for tool overuse mitigation. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 4604–4621, 2025b. 9

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R Bowman. Gpqa: A graduate-level google-proof q&a benchmark. In *First Conference on Language Modeling*, 2024. 6, 21

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International Conference on Machine Learning (ICML)*, pp. 1889–1897. PMLR, 2015. 18

Zhihong Shao, Yeyun Gong, Yelong Shen, Minlie Huang, Nan Duan, and Weizhu Chen. Enhancing retrieval-augmented large language models with iterative retrieval-generation synergy. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 9248–9274, 2023. 6, 20

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024. 3

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. *arXiv preprint arXiv:2503.05592*, 2025. 1, 9

Hao Sun, Zile Qiao, Jiayan Guo, Xuanbo Fan, Yingyan Hou, Yong Jiang, Pengjun Xie, Yan Zhang, Fei Huang, and Jingren Zhou. Zerosearch: Incentivize the search capability of llms without searching. *arXiv preprint arXiv:2505.04588*, 2025. 6, 9, 20

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. Musique: Multihop questions via single-hop question composition. *Transactions of the Association for Computational Linguistics (TACL)*, 10:539–554, 2022. 6, 21

Hanlin Wang, Chak Tou Leong, Jiashuo Wang, Jian Wang, and Wenjie Li. SPA-RL: Reinforcing llm agents via stepwise progress attribution. *arXiv preprint arXiv:2505.20732*, 2025a. 9

Hongru Wang, Cheng Qian, Wanjun Zhong, Xiusi Chen, Jiahao Qiu, Shijue Huang, Bowen Jin, Mengdi Wang, Kam-Fai Wong, and Heng Ji. Acting less is reasoning more! teaching model to act efficiently. *arXiv preprint arXiv:2504.14870*, 2025b. URL `https://arxiv.org/pdf/2504.14870`. 9

Zihan Wang, Kangrui Wang, Qineng Wang, Pingyue Zhang, Linjie Li, Zhengyuan Yang, Xing Jin, Kefan Yu, Minh Nhat Nguyen, Licheng Liu, et al. RAGEN: Understanding self-evolution in llm agents via multi-turn reinforcement learning. *arXiv preprint arXiv:2504.20073*, 2025c. 1, 9

Ziliang Wang, Xuhui Zheng, Kang An, Cijun Ouyang, Jialu Cai, Yuhang Wang, and Yichao Wu. Stepsearch: Igniting llms search ability via step-wise proximal policy optimization. *arXiv preprint arXiv:2505.15107*, 2025d. 6, 9, 20

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, et al. Autogen: Enabling next-gen llm applications via multi-agent conversations. In *First Conference on Language Modeling (COLM)*, 2024. 1, 3, 6, 9, 20

Zhenghai Xue, Longtao Zheng, Qian Liu, Yingru Li, Xiaosen Zheng, Zejun Ma, and Bo An. Simpletir: End-to-end reinforcement learning for multi-turn tool-integrated reasoning. *arXiv preprint arXiv:2509.02479*, 2025. 1, 9

Jianhao Yan, Yafu Li, Zican Hu, Zhi Wang, Ganqu Cui, Xiaoye Qu, Yu Cheng, and Yue Zhang. Learning to reason under off-policy guidance. *arXiv preprint arXiv:2504.14945*, 2025. 6, 20

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. *arXiv preprint arXiv:2412.15115*, 2024a. 6, 7, 19

An Yang, Beichen Zhang, Binyuan Hui, Bofei Gao, Bowen Yu, Chengpeng Li, Dayiheng Liu, Jianhong Tu, Jingren Zhou, Junyang Lin, et al. Qwen2. 5-math technical report: Toward mathematical expert model via self-improvement. *arXiv preprint arXiv:2409.12122*, 2024b. 6, 7, 20

Hang Yang, Hao Chen, Hui Guo, Yineng Chen, Ching-Sheng Lin, Shu Hu, Jinrong Hu, Xi Wu, and Xin Wang. Llm-medqa: Enhancing medical question answering through case studies in large language models. *arXiv preprint arXiv:2501.05464*, 2024c. 7

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D Manning. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 2369–2380, 2018. 6, 21

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, et al. Dapo: An open-source llm reinforcement learning system at scale. *arXiv preprint arXiv:2503.14476*, 2025. 3

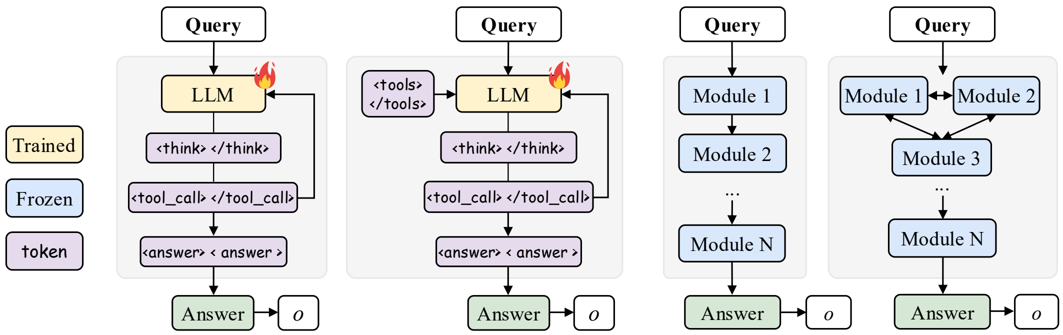
Siliang Zeng, Quan Wei, William Brown, Oana Frunza, Yuriy Nevmyvaka, and Mingyi Hong. Reinforcing multi-turn reasoning in llm agents via turn-level credit assignment. *arXiv preprint arXiv:2505.11821*, 2025a. 9

Weihao Zeng, Yuzhen Huang, Qian Liu, Wei Liu, Keqing He, Zejun Ma, and Junxian He. Simplerl-zoo: Investigating and taming zero reinforcement learning for open base models in the wild. *arXiv preprint arXiv:2503.18892*, 2025b. 1, 6, 19, 20

Shaokun Zhang, Yi Dong, Jieyu Zhang, Jan Kautz, Bryan Catanzaro, Andrew Tao, Qingyun Wu, Zhiding Yu, and Guilin Liu. Nemotron-research-tool-n1: Tool-using language models with reinforced reasoning. *arXiv preprint arXiv:2505.00024*, 2025. 1, 9

TABLE OF CONTENTS

A Two Paradigms of LLMs with Tool Use



(a) Tool-Integrated Reasoning Models (LLM Agents) (b) Training-Free Agentic Systems

Figure 8: **Comparison of two paradigms of LLMs with tool use.** (a) Monolithic tool-integrated reasoning models train a single policy to interleave reasoning (e.g., `<think>`) and tool calls (e.g., `<tool_call>`) within a single, full-context trajectory. (b) Agentic systems decompose tasks across multiple specialized modules (e.g., planner, coder) that collaborate. These systems are typically training-free, orchestrated by handcrafted logic or prompting.

B Training Algorithm of AgentFlow

We provide a detailed flowchart of the overall training algorithm of AgentFlow in Algorithm 1.

Algorithm 1 In-the-Flow Optimization for Agentic Systems

Require: Dataset $\mathcal{D}$, Action Planner policy π_θ, Tool Executor $\mathcal{E}$, Executive Verifier $\mathcal{V}$, Solution Generator $\mathcal{G}$, Toolset K, and Shared Evolving Memory M
Ensure: Optimized Action Planner parameters $\theta^\star$

for each training iteration **do**
  for each query–label pair $(q, y^*) \sim \mathcal{D}$ **do**
    1. In-the-Flow Rollout Generation
    Initialize: $t \leftarrow 1$, $M^t \leftarrow q$
    repeat
      $a^t \sim \pi_\theta(a^t \mid q, K, M^t)$ {*Plan Action*}
      $e^t \sim \mathcal{E}(e^t \mid a^t, K)$ {*Execute Action*}
      $v^t \sim \mathcal{V}(v^t \mid q, e^t, M^t)$ {*Verify Result*}
      $M^{t+1} = f_{\text{mem}}(M^t, a^t, e^t, v^t)$ {*Update Memory*}
      $t \leftarrow t + 1$
    until termination condition met
    $o \sim \mathcal{G}(o \mid q, M^T)$ {*Generate Final Solution*}
    2. Reward Computation
    $R(a^t) = \bar{R}(o, q, y^*), \quad \forall t = 1, \ldots, T$
    3. Policy Update
    Update the Action Planner policy π_θ by maximizing the Flow-GRPO objective (Eq. 5)
  end for
end for
return optimized parameters $\theta^\star$

C MATHEMATICAL ANALYSIS OF FLOW-GRPO

C.1 PRELIMINARIES AND NOTATION

We adopt the notation from the paper to formalize our analysis.

Definition C.1 (Core Components)**.** Here we list core definition of variables.

Symbol and Description	
π_θ	The trainable planner policy, parameterized by θ.
$\pi_{\theta_{\text{old}}}$	The behavior policy used to sample trajectories.
s_t	The state at turn t, defined as $s_t = (q, K, M_t)$.
a_t	The action (a sequence of tokens) generated at state s_t, where $a_t \sim \pi_\theta(\cdot \mid s_t)$.
τ	A trajectory of states and actions over T time steps, defined as $\tau = \{(s_t, a_t)\}_{t=1}^{T}$.
$R(\tau)$	The outcome-based reward for trajectory τ, where $R(\tau) \in \{0, 1\}$.
A_τ	The group-normalized advantage for trajectory τ. A crucial property is that the advantage is constant for all timesteps within a trajectory defined in Eq. 7: $A_t = A_\tau, \quad \forall (s_t, a_t) \in \tau$.
$\rho_{t,j}$	The token-level importance sampling ratio, defined as: $$\rho^i_{t,j} = \frac{\pi_\theta(a^t_{i,j} \mid s^t_i, a^t_{i,1:j-1})}{\pi_{\theta_{\text{old}}}(a^t_{i,j} \mid s^t_i, a^t_{i,1:j-1})}.$$
$L_{\text{clip}}(\rho, A)$	The PPO clipped objective term, defined as $L_{\text{clip}}(\rho, A) = \min(\rho A, \text{clip}(\rho, 1-\epsilon, 1+\epsilon)A)$.

Definition C.2 (Objective Functions)**.** The *global policy objective* is the expected trajectory-level reward:

$$\mathcal{J}(\theta) := \mathbb{E}_{\tau \sim \pi_\theta}[R(\tau)]. \tag{8}$$

The *single-turn optimization objective* for a given state s_t is defined as:

$$\mathcal{J}_{\text{local}}(\theta; s_t) := \mathbb{E}_{a_t \sim \pi_{\theta_{\text{old}}}(\cdot|s_t)} \left[\frac{1}{|a_t|} \sum_{j=1}^{|a_t|} L_{\text{clip}}(\rho_{t,j}, A_t) \right]. \tag{9}$$

The full Flow-GRPO objective function in the multi-turn setting is given by:

$$\mathcal{J}_{\text{Flow-GRPO}}(\theta) := \mathbb{E}_{\substack{(q,y^*)\sim\mathcal{D} \\ \{\tau_i\}_{i=1}^{G}\sim\pi_{\theta_{\text{old}}}}} \left[\frac{1}{G} \sum_{i=1}^{G} \frac{1}{T_i} \sum_{t=1}^{T_i} \frac{1}{|a^i_t|} \sum_{j=1}^{|a^i_t|} L_{\text{clip}}(\rho^i_{t,j}, A_i) \right] - \beta \mathbb{D}_{\text{KL}}(\pi_\theta \| \pi_{\text{ref}}). \tag{10}$$

C.2 EQUIVALENCE PROOF FOR OPTIMIZATION OBJECTIVES

Theorem C.1. *In Flow-GRPO, maximizing the global multi-turn objective is mathematically equivalent to maximizing the expected token-level local objective at each time step under the on-policy induced state distribution, given standard sampling assumptions (trajectories sampled i.i.d. from the policy with fixed finite turn T).*

Proof. Let's denote the clipping part of the Flow-GRPO objective as $\mathcal{J}_{\text{clip}}(\theta)$.

First, by the linearity of expectation, we can simplify the expectation over a group of G trajectories. Since the trajectories $\{\tau_i\}$ are sampled independently and identically (i.i.d.) from the behavior policy $\pi_{\theta_{\text{old}}}$, the expectation of their average is equal to the expectation over a single trajectory.

$$\mathcal{J}_{\text{clip}}(\theta) = \mathbb{E}_{(q,y^*)\sim\mathcal{D}} \left[\mathbb{E}_{\{\tau_i\}_{i=1}^{G}\sim\pi_{\theta_{\text{old}}}} \left[\frac{1}{G} \sum_{i=1}^{G} \frac{1}{T_i} \sum_{t=1}^{T_i} \left(\frac{1}{|a^i_t|} \sum_{j=1}^{|a^i_t|} L_{\text{clip}}(\rho^i_{t,j}, A_i) \right) \right] \right] \tag{11}$$

$$= \mathbb{E}_{(q,y^*)\sim\mathcal{D}} \left[\mathbb{E}_{\tau\sim\pi_{\theta_{\text{old}}}(\cdot|q)} \left[\frac{1}{T} \sum_{t=1}^{T} \left(\frac{1}{|a_t|} \sum_{j=1}^{|a_t|} L_{\text{clip}}(\rho_{t,j}, A_\tau) \right) \right] \right]. \tag{12}$$

Here, $\tau = \{(s_t, a_t)\}_{t=1}^{T}$ represents a single, arbitrarily sampled trajectory with advantage A_τ.

Next, we can re-interpret the expectation over trajectories as an expectation over the state-visitation distribution induced by the policy $\pi_{\theta_{\text{old}}}$. Let $d^{\pi_{\theta_{\text{old}}}}$ be the on-policy distribution of states visited, where each state s_t in a trajectory of length T is weighted by $1/T$. The expectation can be rewritten as:

$$\mathcal{J}_{\text{clip}}(\theta) = \mathbb{E}_{(q,y^*)\sim\mathcal{D}} \left[\mathbb{E}_{s_t \sim d^{\pi_{\theta_{\text{old}}}}} \left[\mathbb{E}_{a_t \sim \pi_{\theta_{\text{old}}}(\cdot|s_t)} \left[\frac{1}{|a_t|} \sum_{j=1}^{|a_t|} L_{\text{clip}}(\rho_{t,j}, A_t) \right] \right] \right]. \tag{13}$$

Note that A_t is the advantage corresponding to the trajectory from which s_t was sampled.

We now recognize that the inner expectation is precisely the definition of the local, per-state objective, $\mathcal{J}_{\text{local}}(\theta; s_t)$.

$$\mathcal{J}_{\text{clip}}(\theta) = \mathbb{E}_{(q,y^*)\sim\mathcal{D},\ s_t \sim d^{\pi_{\theta_{\text{old}}}}} \left[\mathcal{J}_{\text{local}}(\theta; s_t) \right]. \tag{14}$$

Adding the KL-divergence term back, we arrive at the final equivalence:

$$\mathcal{J}_{\text{Flow-GRPO}}(\theta) = \mathbb{E}_{(q,y^*)\sim\mathcal{D},\ s_t \sim d^{\pi_{\theta_{\text{old}}}}} \left[\mathcal{J}_{\text{local}}(\theta; s_t) \right] - \beta D_{KL}(\pi_\theta \| \pi_{\text{ref}}). \tag{15}$$

This proves that maximizing the global multi-turn Flow-GRPO objective is equivalent to maximizing the expected token-level local objective at each time step under the on-policy induced state distribution. □

C.3 Convergence Analysis

Having established the structural validity of the objective, we now analyze its convergence properties. The analysis builds on the monotonic improvement guarantee provided by trust-region methods (Schulman et al., 2015).

Lemma C.2 (Policy Performance Difference)**.** *For two policies π_θ and $\pi_{\theta_{\text{old}}}$, the difference in expected return can be expressed as:*

$$\mathcal{J}(\theta) - \mathcal{J}(\theta_{\text{old}}) = \mathbb{E}_{\tau\sim\pi_\theta} \left[\sum_{t=1}^{T} A_{\theta_{\text{old}}}(s_t, a_t) \right], \tag{16}$$

where $A_{\theta_{\text{old}}}$ is the advantage function under the old policy.

This lemma enables the construction of a lower bound on policy improvement.

Theorem C.3 (Monotonic Improvement Guarantee)**.** *Define the surrogate objective*

$$\mathcal{L}_{\theta_{\text{old}}}(\theta) = \mathbb{E}_{\tau\sim\pi_{\theta_{\text{old}}}} \left[\sum_{t=1}^{T} \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} A_{\theta_{\text{old}}}(s_t, a_t) \right]. \tag{17}$$

Then the performance improvement satisfies the lower bound

$$\mathcal{J}(\theta) - \mathcal{J}(\theta_{\text{old}}) \;\; \geq \;\; \mathcal{L}_{\theta_{\text{old}}}(\theta) - C \cdot \bar{\mathbb{D}}_{\text{KL}}(\pi_{\theta_{\text{old}}}, \pi_\theta), \tag{18}$$

where $C > 0$ is a constant depending on the horizon and reward scale, and $\bar{\mathbb{D}}_{\text{KL}}$ denotes the average KL-divergence between the two policies.

By optimizing the right-hand side of the above inequality, we are guaranteed to improve the performance of π_θ. Therefore, for policies π_θ^t and π_θ^{t+1} obtained from iterations t and $t+1$, we have:

$$J(\pi^{t+1}) \geq J(\pi^t). \tag{19}$$

Conclusion. This analysis establishes that Flow-GRPO optimizes a valid surrogate objective and guarantees monotonic policy improvement, thereby converging reliably to a locally optimal policy.

D EXPERIMENTAL DETAILS

D.1 TRAINING DETAILS

We provide further details on the training setup for AGENTFLOW. Our Flow-GRPO implementation uses a learning rate of 1×10^{-6}. The Action Planner generates actions with a sampling temperature of 0.5 to balance exploration and exploitation. To prevent policy collapse and stabilize training, we incorporate a KL-divergence penalty against a reference policy with a coefficient $\beta = 0.001$. The maximum output length for the planner is set to 2048 tokens to ensure complete exploration during rollouts.

To accelerate the training speed, we limit the maximum number of turns per rollout to 3. The final-outcome reward signal (Eq. 4) is provided by an LLM-as-judge, for which we use *GPT-4o*. All tool calls are executed synchronously with a 500-second timeout to handle external service latency robustly. The LLM engines within the tools are set to a temperature of 0.0 to ensure deterministic and stable outputs. The full training process was conducted on 8 NVIDIA A100 GPUs. Further details on agent prompts and the memory update mechanism are provided in §F.1.

D.2 EVALUATION DETAILS

Here, we outline the specifics of our evaluation protocol. For evaluation, we increase the maximum number of turns per rollout to $T = 10$ to allow for more extensive and deeper reasoning. The planner's sampling temperature is set to 0.7 to encourage diverse solution paths. Unless otherwise specified, all tool LLM engines are initialized with *Qwen2.5-7B-Instruct*.

For fair and consistent evaluation, we adopt the previous work's methodology while standardizing tools: we replace search tools in search-enhanced models with our Google Search tool and code tools in code-enhanced models with our Python Coder tool. We use *GPT-4o* as an LLM-based judge to determine the correctness of final answers. This approach provides a robust measure of semantic and numerical equivalence, which is critical for complex reasoning tasks. The specific judging prompt is detailed in §F.3, and additional information on evaluation datasets can be found in §D.4. To mitigate randomness, we report the average accuracy with standard deviation across three trials for all experiments.

D.3 COMPARED BASELINES

Proprietary LLMs:

- **Qwen2.5 Series** (Yang et al., 2024a), created by Alibaba, comes in multiple configurations. These models undergo training on multilingual corpora covering 29 different languages, demonstrating superior performance in cross-lingual applications. Furthermore, Qwen2.5 showcases robust proficiency in programming and mathematical domains.
- **Llama-3 Series** (Dubey et al., 2024), created by Meta AI, encompasses various iterations. Each model configuration within the Llama family provides dual versions: foundational and instruction-following variants. Training incorporates diverse dataset combinations spanning multiple domains and linguistic varieties. The Llama model family demonstrates excellent results in logical reasoning, software development, and cross-lingual comprehension evaluations. Through progressive enhancements in fine-tuning methodologies and expanded sequence lengths, these models become more applicable to practical deployment scenarios.
- **GPT-4o Series** (Hurst et al., 2024), produced by OpenAI, includes several model variants such as GPT-4o and GPT-4o-mini, with training leveraging extensive multimodal datasets encompassing text, vision, and audio modalities. The series achieves outstanding performance in complex reasoning tasks, creative generation, and multimodal understanding benchmarks with continuous refinements in alignment techniques and enhanced processing capabilities.

Reasoning LLMs:

- **SFT** (Zeng et al., 2025b) serves as our basic baseline following Search-R1 (Jin et al., 2025). We fine-tune models using supervised fine-tuning on GPT-4o-generated reasoning chains.

- **SimpleRL-Zoo** (Zeng et al., 2025b) investigates zero reinforcement learning training across 10 diverse base models spanning different families and sizes using GRPO algorithm with simple rule-based rewards, achieving substantial improvements in reasoning accuracy.
- **Open-Reasoner-Zero** (Hu et al., 2025a) presents the first open-source implementation of large-scale reasoning-oriented RL training using PPO with GAE and straightforward rule-based rewards, without KL regularization. The framework demonstrates that minimalist design can successfully scale both response length and benchmark performance.
- **General-Reasoner** (Ma et al., 2025) extends LLM reasoning capabilities beyond mathematics to diverse domains using RLVR through a 230K verifiable reasoning questions dataset spanning physics, chemistry, and finance.
- **LUFFY** (Yan et al., 2025) addresses limitations in on-policy RLVR by introducing an off-policy framework that augments training with external reasoning demonstrations using Mixed Policy GRPO and regularized importance sampling.

Search-Integrated Reasoning LLMs:

- **Iter-RetGen** (Shao et al., 2023) addresses limitations in retrieval-augmented language models by introducing iterative retrieval-generation synergy, where a model's previous response serves as context for retrieving more relevant knowledge in subsequent iterations.
- **Search-R1** (Jin et al., 2025) represents a reinforcement learning approach that develops a model from the ground up to invoke search functionality throughout the reasoning process.
- **ZeroSearch** (Sun et al., 2025) addresses high API costs in RL-based search training by using an LLM to simulate search engines, employing lightweight supervised fine-tuning to transform an LLM into a retrieval module that generates both useful and noisy documents. The framework combines this with a curriculum-based rollout strategy that progressively degrades document quality, achieving better performance than real search engine-based methods while incurring zero API costs.
- **ReSearch** (Chen et al., 2025) proposes a reinforcement learning framework that trains LLMs to integrate search operations as components of the reasoning chain without supervised data on reasoning steps, treating search decisions as guided by text-based thinking.
- **StepSearch** (Wang et al., 2025d) addresses the sparse reward problem in multi-hop reasoning by training search LLMs using step-wise proximal policy optimization with intermediate rewards and token-level process supervision based on information gain and redundancy penalties.
- **VerlTool** (Jiang et al., 2025) addresses fragmentation and synchronization bottlenecks in Agentic Reinforcement Learning with Tool use by introducing a unified modular framework that extends beyond single-turn RLVR paradigms, providing upstream VeRL alignment and unified tool management with asynchronous rollout execution achieving near 2× speedup.

Code-Integrated Reasoning LLMs:

- **TIR** (Yang et al., 2024b) is a basic baseline that demonstrates the model's ability to generate code for tool utilization. In our implementation, we directly prompt the model to write code that calls the programming interpreter and processes the returned results to generate the final answer.
- **ToRL** (Li et al., 2025b) is a code-enhanced architecture developed via reinforcement learning that empowers models to independently activate code execution environments for mathematical reasoning tasks.

Training-free Agentic System

- **AutoGen** (Wu et al., 2024) introduces an agentic conversation framework that enables developers to build LLM applications through conversable agents that can operate using combinations of LLMs, human inputs, and tools.

D.4 Evaluation Datasets

We provide a detailed introduction to the *search-intensive* and *agentic* benchmarks in our experiments as follows:

- **Bamboogle** (Press et al., 2023) presents a demanding multi-step reasoning dataset containing manually constructed questions requiring up to four inferential steps. The dataset evaluates models' capacity for intricate compositional reasoning across interconnected facts.
- **2Wiki (2WikiMultihopQA)** (Ho et al., 2020) constitutes a comprehensive multi-step QA corpus combining structured Wikidata knowledge with unstructured Wikipedia text. The dataset encompasses varied question formats and annotated reasoning chains to facilitate interpretable sequential inference. We randomly sample 100 examples as a test set for efficiency.
- **HotpotQA** (Yang et al., 2018) represents a widely-adopted question answering corpus featuring multi-step queries constructed from Wikipedia entries. We randomly sample 100 examples as a test set for efficiency.
- **Musique** (Trivedi et al., 2022) comprises a multi-step reasoning corpus requiring sequential inference where each reasoning stage depends on information derived from preceding steps. We conduct evaluations using the development partition of this particularly challenging dataset. We randomly sample 100 examples as a test set for efficiency.
- **GAIA** (Mialon et al., 2023) constitutes a benchmark engineered to assess general AI systems and agents, demanding capabilities including sequential reasoning, web navigation, and comprehensive tool utilization skills. We utilize the text-exclusive portion of this dataset, designed to challenge base language models in our experimental setup.

Furthermore, we also conduct a series of experiments on *math* and *scientific reasoning* benchmarks:

- **AIME24** (Art of Problem Solving, 2025) A collection of 30 demanding mathematical problems sourced from the 2024 American Invitational Mathematics Examination (AIME), encompassing algebra, geometry, number theory, and combinatorics. Each JSONL-formatted record contains the problem identifier, question text, comprehensive solution methodology, and the final numerical result. Created to assess large language models' sophisticated mathematical reasoning abilities, the dataset presents substantial difficulty, systematic multi-phase solutions, and distinctive answers—establishing it as a robust benchmark for evaluating advanced analytical capabilities.
- **AMC23** (MAA, 2023) contains mathematical problems derived from the 2023 American Mathematics Competition, emphasizing areas such as functional equations and complex analysis.
- **GameOf24** (Lile, 2024) derives from the traditional numerical puzzle known as 24 (alternatively called the 24 numbers game). The challenge requires utilizing four given numbers with fundamental arithmetic operations (addition, subtraction, multiplication, division) to create an expression yielding 24. For instance, with numbers 4, 9, 10, and 13, a correct solution would be "(10 - 4) × (13 - 9) = 24". Successfully solving requires computational proficiency along with iterative attempts to validate potential solutions. Each challenge is formatted as open-ended inquiries.
- **GPQA** or Graduate Level Google-Proof Q&A Benchmark (Rein et al., 2024) comprises a collection of demanding text-based multiple choice problems authored by subject specialists in biology, physics, and chemistry, intentionally crafted to be "exceptionally challenging". We randomly sample 100 examples as a test set for efficiency.
- **MedQA** (Jin et al., 2021) features text-based multiple choice problems assembled from professional medical licensing examinations. Problems encompass comprehensive medical knowledge and clinical reasoning skills.

E More Discussion about Experiment Results

E.1 Main Result Analysis

Our main results are presented in Tables 1 and 2. Overall, AgentFlow consistently outperforms all baseline models across diverse domains, including search-intensive tasks, agentic tasks, and mathematical and scientific reasoning tasks. These comprehensive results yield several key insights:

Monolithic LLMs are insufficient for complex reasoning. While scaling up model size (from 7B model to GPT-4o) improves average performance, their monolithic nature presents limitations when facing complex tasks that require multi-turn reasoning and sub-goal decomposition. In contrast, our proposed AgentFlow consistently outperforms these larger models. Specifically, it achieves an average improvement of 8.2% over GPT-4o on search-intensive tasks (57.3% vs. 49.1% in Table 1), and a remarkable 15.8% gain over GPT-4o on agentic tasks (33.1% vs. 17.3% in Table 1). For mathematical reasoning benchmarks, AgentFlow obtains a substantial improvement of 16.4% over GPT-4o (51.5% vs. 35.1% in Table 2). Furthermore, it surpasses the strong Llama-3.3-70B by 12.5% on scientific reasoning tasks (63.5% vs. 51.0% in Table 2). These results demonstrate that the carefully designed agentic system of AgentFlow, despite being built on a 7B-parameter backbone, can deliver superior and more efficient performance compared to substantially larger monolithic LLMs.

Specialized reasoning models exhibit strong in-domain focus but limited generalizability. While domain-specific fine-tuning and tailored tool integration provide clear benefits over base LLMs, they fail to deliver robust cross-domain performance due to fundamental scaling limitations. Our evaluation across three reasoning domains substantiates these limitations. On search-intensive tasks, specialized models such as Search-R1 (33.3%) and VerlTool (39.0%) perform well within their narrow scope yet fall substantially short of AgentFlow (57.3%) as shown in Table 1. Similarly, in mathematical reasoning, methods like SimpleRL-reason (36.6%) and ToRL (37.0%) trail significantly behind AgentFlow (51.5%) in Table 2. Even in scientific reasoning, where models such as Luffy (55.5%) offer competitive results, they are consistently surpassed by AgentFlow (63.5%) in Table 2. These findings demonstrate that while specialized reasoning models excel within narrow domains, their reliance on a single monolithic policy introduces poor generalization, making them brittle when confronted with diverse, cross-domain challenges.

AgentFlow demonstrates superior, versatile reasoning through its adaptive agentic system. AgentFlow establishes a new state-of-the-art agentic system by achieving an average accuracy of 57.3% on search-intensive tasks, 33.1% on agentic tasks, 51.5% on mathematical reasoning, and 63.5% on scientific reasoning. Our method's advantage stems from combining an agentic system with targeted planning policy refinement via on-policy reinforcement learning in an online fashion. When compared to AutoGen—a general agent framework with the same backbone model—AgentFlow demonstrates a massive improvement of 14.9% on search tasks and 19.9% on math tasks. This underscores that the core advantage comes from our dedicated trainable agentic system that integrates our novel Flow-GRPO for in-system on-policy optimization, enabling effective agent planning and tool utilization to solve complex, long-horizon problems across diverse domains.

E.2 In-depth Analysis of Optimized Planning

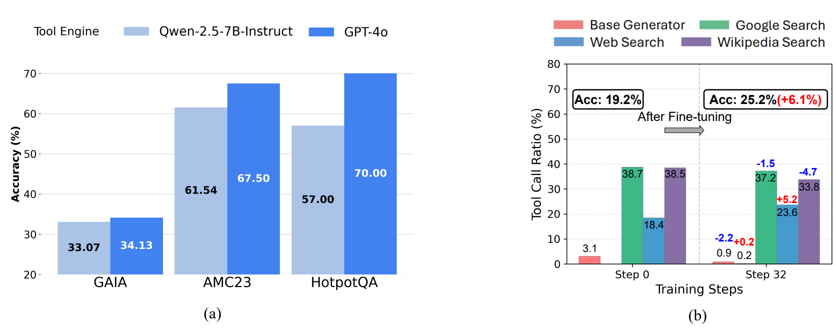


Figure 9: (a) Tool scaling: performance improves when base tools are upgraded from Qwen-2.5-7B-Instruct to GPT-4o on GAIA and AMC23. (b) Tool call ratio change by Flow-GRPO fine-tuning.

AGENTFLOW adapts to inference-time tool scaling. We scale the tools—the Base Generator and Python Coder—to GPT-4o-powered versions. Empirical results on search and math datasets (Figure 9 (a)) show that AGENTFLOW, when using these GPT-4o-powered tools, substantially outperforms its performance with Qwen2.5-7B-Instruct-powered tools, achieving improvements of 1.0% on GAIA, 6.0% on AMC23, and a notable 13.0% on HotpotQA. This finding further supports a consistent trend: after in-the-flow RL training, the planner can adaptively leverage improvements in the underlying tools to enhance the agentic system's overall performance.

Flow-GRPO spontaneous tool usage preference change. We further compare tool usage distributions before and after in-the-flow RL training on Musique. Figure 9 (b) shows that due to Musique's need for a diverse source of information, Flow-GRPO optimizes the planner to increase Web Search to delve deeper into the URL provided by other search tools. This maneuver presents a steady performance improvement of 6.1%.

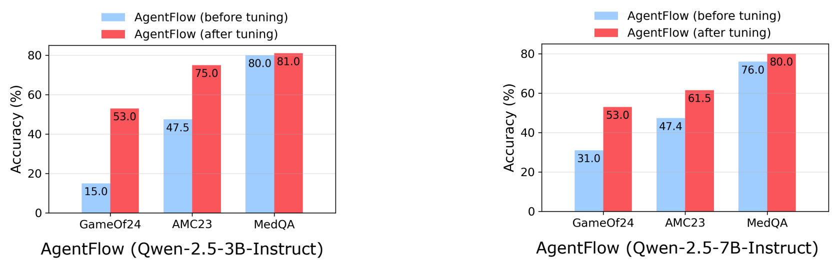


Figure 10: Flow-GRPO fine-tuning offers consistent gains on AGENTFLOW as the backbone model size scales from 3B to 7B.

More evidence of training scaling in backbone size. We further investigate how the backbone LLM scale affects AGENTFLOW's performance and the efficacy of Flow-GRPO. We construct two versions of the system: one using *Qwen2.5-3B-Instruct* and another using *Qwen2.5-7B-Instruct* for all four modules (planner, executor, verifier, and generator) as well as the associated tools. In both versions, only the planner is fine-tuned with Flow-GRPO. As shown in Figure 10, Flow-GRPO fine-tuning consistently improves performance across tasks for both backbones. These results demonstrate that our in-the-flow optimization is effective across model capacities, enhancing AGENTFLOW regardless of LLM size.

F Instruction Templates in AgentFlow

F.1 Modules and Memory

F.1.1 Action Planner

Tool Metadata can be found in § F.2.

Instruction for *Action Planner*

Task: Determine the optimal next step to address the query using available tools and previous context.

Context:
Query: {**Question**}
Available Tools: [*Base Generator, Python Coder, Google Search, Wikipedia Search, Web Search*]
Toolbox Metadata: [*Tool Metadata1, Tool Metadata2, ...*]
Previous Steps: {**Actions** *from Memory*}

Instructions:

1. Analyze the current objective, the history of executed steps, and the capabilities of the available tools.
2. Select the **single most appropriate tool** for the next action. Consider:
3. The *specificity* of the task (e.g., calculation vs. information retrieval).
4. The *source* of required information (e.g., general knowledge, mathematical computation, a specific URL).
5. The *limitations* of each tool as defined in the metadata.
6. Formulate a clear, concise, and achievable **sub-goal** that precisely defines what the selected tool should accomplish.
7. Provide all necessary **context** (e.g., relevant data, variable names, file paths, or URLs) so the tool can execute its task without ambiguity.

Response Format:

1. **Justification:** Explain why the chosen tool is optimal for the sub-goal, referencing its capabilities and the task requirements.
2. **Context:** Provide all prerequisite information for the tool.
3. **Sub-Goal:** State the exact objective for the tool.
4. **Tool Name:** State the exact name of the selected tool (e.g., *Wikipedia Search*).

Rules:
Select **only one** tool per step.
The *Sub-Goal* must be directly and solely achievable by the selected tool.
The *Context* section must contain **all** information the tool needs; do not assume implicit knowledge.
The final response must end with the *Context*, *Sub-Goal*, and *Tool Name* sections in that order. No additional text should follow.

F.1.2 TOOL EXECUTOR

Instruction for *Tool Executor*

Task: Generate a precise command to execute the selected tool.

Context:
Query: {**Question**}
Sub-Goal: {**Sub Goal** *from Next Step Plan*}
Tool Name: {**Selected Tool** *from Next Step Plan*}
Toolbox Metadata: {**Selected Tool Metadata** *from Next Step Plan*}
Relevant Data: {**Context** *from Next Step Plan*}

Instructions:

1. Analyze the tool's required parameters from its metadata.
2. Construct valid Python code that addresses the sub-goal using the provided context and data.
3. The command must include at least one call to `tool.execute()`.
4. Each `tool.execute()` call must be assigned to a variable named **`execution`**.
5. Use exact numbers, strings, and parameters in the `tool.execute()` call based on the context.

Output Format: Present your response in the following structured format. Do not include any extra text or explanations.

Generated Command:

```
execution = tool.execute(query="Summarize the following porblom:"Isaac has 100 toys,
masa gets ...., how much are their together?")
```

Example 1:
Generated Command:

```
execution = tool.execute(query="Summarize the following porblom:"Isaac has 100 toys,
masa gets ...., how much are their together?")
```

Example 2:
Generated Command:

```
execution = tool.execute(query=["Methanol", "function of hyperbola", "Fermat's Last
Theorem"])
```

F.1.3 Execution Verifier

Instruction for *Execution Verifier*

Task: Evaluate if the current memory is complete and accurate enough to answer the query, or if more tools are needed.

Context:
Query: {**Question**}
Available Tools: [*Base Generator, Python Coder, Google Search, Wikipedia Search, Web Search*]
Toolbox Metadata: [*Tool Metadata1, Tool Metadata2, ...*]
Memory (Tools Used & Results): {**Actions** *from Memory*}

Instructions:

1. Review the original query, the initial analysis, and the complete history of actions and results in the memory.
2. Assess the **completeness** of the memory:
3. Does the accumulated information fully address all aspects of the query?
4. Are there any unanswered sub-questions or missing pieces of information?
5. Check for potential **issues**:
6. Are there any inconsistencies or contradictions between different steps?
7. Is any information ambiguous, potentially hallucinated, or in need of verification?
8. Determine if any **unused tools** could provide critical missing information based on their metadata.

Final Determination:
If the memory is sufficient to form a complete and accurate answer, explain why and conclude with **"Conclusion: STOP"**.
If more information is needed, clearly state what is missing, suggest which tool(s) could help, and conclude with **"Conclusion: CONTINUE"**.

Rules:
The response must end with either **exactly** "Conclusion: STOP" or "Conclusion: CONTINUE".
Do not include any text after the conclusion statement.
Your justification must be concise and directly tied to the query and memory.

F.1.4 SOLUTION GENERATOR

Instruction for *Solution Generatorr*

Task: Generate a concise final answer to the query based on all provided context.

Context:
Query: {**Question**} **Initial Analysis:** {**Query Analysis**}
Actions Taken: {**Actions** *from Memory*}

Instructions:

1. Carefully review the original user query, the initial analysis, and the complete sequence of actions and their results.
2. Synthesize the key findings from the action history into a coherent narrative.
3. Construct a clear, step-by-step summary that explains how each action contributed to solving the query.
4. Provide a direct, precise, and standalone final answer to the original query.

Output Structure:

1. **Process Summary:** A clear, step-by-step breakdown of how the query was addressed. For each action, state its purpose (e.g., "To verify X") and summarize its key result or finding in one sentence.
2. **Answer:** A direct and concise final answer to the query. This should be a self-contained statement that fully resolves the user's question.

Rules:
The response must follow the exact two-part structure above.
The *Process Summary* should be informative but concise, focusing on the logical flow of the solution.
The *Answer* must be placed at the very end and be clearly identifiable.
Do not include any additional sections, explanations, or disclaimers beyond the specified structure.

F.1.5 Evolving Memory

Our shared evolving memory system creates a deterministic, structured record that captures the reasoning process across three integrated agents: the *Action Planner*, *Tool Executor*, and *Executive Verifier*. By sequentially stacking crucial information from each action step, the system enables transparent state tracking, controllable behavior, and bounded context growth.

Example *Memory* Entry

```
"Query": {Where is the largest shopping mall besides Tokyo's biggest metropolitan
station? }

"Action Turn 1": {
    "Tool Name": "Wikipedia Search",
    "Sub-Goal": "Retrieve detailed information about Tokyo"s metropolitan area from
Wikipedia.",
    "Command": "execution = tool.execute(query="Tokyo metropolitan area details")",
    "Result": "The Greater Tokyo Area is the largest metropolitan area in the
world...",
    "Verification Status": "
        ### Brief Review of the Query, Initial Analysis, and Previous Memory.
        ### Assessment of Completeness and Accuracy.
        ### Conclusion: The memory is not complete and accurate enough to answer the
query. Additional tools are needed to verify or generate more solutions.
         Final Determination: Conclusion: CONTINUE"
},

"Action Turn 2": {
    ...
},

...

"Action Turn t": {
    ...
    "Verification Status": "
        ### Brief Review of the Query, Initial Analysis, and Previous Memory.
        ### Assessment of Completeness and Accuracy. (Including Time Dilation
Calculation, Geographic Precise, Inconsistencies or Contradictions, Unit Conversion,
etc. )
        ### Conclusion: The memory is complete and accurate enough to answer the
query. No additional tools are needed to verify or generate more solutions.
         Final Determination: Conclusion: STOP" }
```

The memory reading and matching process employs regular expressions to parse outputs generated by different system components, adhering to standardized formats defined in their respective component instructions. For the *Action Planner*, we use a relatively permissive regular expression to extract key information. Specifically, it matches the content immediately following: *Sub-Goal* as the sub-goal and the content following; *Tool Name* as the selected tool. This extracted information is then used to populate the next memory entry. For the *Tool Executor*, the regular expression is designed to capture the entire *Command* line starting with `execution = tool.execute(...)`. Additionally, the value passed to the *Query* parameter within this command is parsed and saved into the memory for future reference. All results returned by the tools are directly stored in the *Result* field of the memory. The *Verification Status* is extracted from *Execution Verifier*, including a brief analysis of the current tool result and previous memory, and then it gives a conclusion whether the loop needs to be CONTINUE or STOP.

F.2 Toolset Metadata

This section details the implementation and metadata of the tools used in our main results. We employ a suite of specialized tools, each designed for distinct tasks. Below, we present the core metadata for each tool, including its functionality, input/output schema, limitations, and best practices.

F.2.1 Base Generator

Tool Metadata of *Base Generator*

Description: A generalized tool that takes query from the user, and answers the question step by step to the best of its ability. It can also accept an image.

Input: query: str - The query that includes query from the user to guide the agent to generate response.

Output: str - The generated response to the original query

Demo Commands:

Command:

```
execution = tool.execute(query="Summarize the following text in a few lines")
```

Description: Generate a short summary given the query from the user.

Limitation

The Base Generator may provide hallucinated or incorrect responses.

Best Practice

1. Use it for general queries or tasks that don't require specialized knowledge or specific tools in the toolbox.
2. Provide clear, specific query.
3. Use it to answer the original query through step by step reasoning for tasks without complex or multi-step reasoning.
4. For complex queries, break them down into subtasks and use the tool multiple times.
5. Use it as a starting point for complex tasks, then refine with specialized tools.
6. Verify important information from its responses.

LLM Engine Required: True

F.2.2 Python Coder

The Python coder leverages a large language model (LLM) engine to generate Python code snippets, which are formatted using the Markdown code block syntax: ```` ```python <code snippet> ``` ```` Each generated code snippet is executed in a secure local sandbox environment. To prevent excessive output—especially from infinite loops or verbose computations—the execution output is truncated if it exceeds 10,000 characters. This ensures system stability and responsiveness while maintaining visibility into the program's behavior.

Tool Metadata of *Python Coder*

Description: A tool that generates and executes simple Python code snippets for basic arithmetical calculations and math-related problems. The generated code runs in a highly restricted environment with only basic mathematical operations available.

Input: query: str - A clear, specific description of the arithmetic calculation or math problem to be solved, including any necessary numerical inputs.

Output: dict - A dictionary containing the generated code, calculation result, and any error messages.

Output prompt: Given a query, generate a Python code snippet that performs the specified operation on the provided data. Please think step by step. Ensure to break down the process into clear, logical steps. Make sure to print the final result in the generated code snippet with a descriptive message explaining what the output represents. The final output should be presented in the following format:

````
```python
<code snippet>
```
````

Demo Commands: Command:

```
execution = tool.execute(query="Find the sum of prime numbers up to 50")
```

Description: Generate a Python code snippet to find the sum of prime numbers up to 50.

Command:

```
query=" Given the list [1, 2, 3, 4, 5, 6, 7, 8, 9, 10], calculate the sum of squares of
odd numbers"

execution = tool.execute(query=query)
```

Description: Generate a Python function for a specific mathematical operation on a given list of numbers.

Limitation

1. Restricted to basic Python arithmetic operations and built-in mathematical functions.
2. Cannot use any external libraries or modules, including those in the Python standard library.
3. Limited to simple mathematical calculations and problems.
4. Cannot perform any string processing, data structure manipulation, or complex algorithms.
5. No access to any system resources, file operations, or network requests.
6. Cannot use 'import' statements.
7. All calculations must be self-contained within a single function or script.
8. Input must be provided directly in the query string.
9. Output is limited to numerical results or simple lists/tuples of numbers.
10. Output should be kept to a single numerical result or a simple list/tuple of numbers.
11. DO NOT generate loop output.

Tool Metadata of *Python Coder (Continue)*

Best Practice

1. Provide clear and specific queries that describe the desired mathematical calculation.
2. Include all necessary numerical inputs directly in the query string.
3. Keep tasks focused on basic arithmetic, algebraic calculations, or simple mathematical algorithms.
4. Ensure all required numerical data is included in the query.
5. Verify that the query only involves mathematical operations and does not require any data processing or complex algorithms.
6. Review generated code to ensure it only uses basic Python arithmetic operations and built-in math functions.

LLM Engine Required: True

F.2.3 GOOGLE SEARCH

Tool Metadata of *Google Search*

Description: A web search tool powered by Google Search that provides real-time information from the internet with citation support.

Input: query: str - The search query to find information on the web.

Input: add_citations: bool - Whether to add citations to the results. If True, the results will be formatted with citations. By default, it is True.

Output: str - The search results of the query.

Demo Commands:

Command:

```
execution = tool.execute(query="What is the capital of France?")
```

Description: Search for general information about the capital of France with default citations enabled.

Command:

```
execution = tool.execute(query="Who won the euro 2024?", add_citations=False)
```

Description: Search for information about the Euro 2024 winner without citations.

Command:

```
execution = tool.execute(query="Physics and Society article arXiv August 11, 2016",
add_citations=True)
```

Description: Search for specific academic articles with citations enabled.

Limitation

1. This tool is only suitable for general information search.
2. This tool contains less domain-specific information.
3. This tool is not suitable for searching and analyzing videos on YouTube or other video platforms.

Best Practice

1. Choose this tool when you want to search for general information about a topic.
2. Choose this tool for question types of query, such as “What is the capital of France?” or “Who invented the telephone?”.
3. The tool will return summarized information.
4. This tool is more suitable for definition, world knowledge, and general information search.

LLM Engine Required: False

F.2.4 WIKIPEDIA SEARCH

Wikipedia search will first call Wikipedia API to retrieve relevant URLs with snippets. Then the RAG (Retrieval-Augmented Generation) process begins by extracting raw text content from the given webpage URL, cleaning it to remove HTML elements and retain only meaningful text. This content is then split into overlapping chunks of approximately 200 words each, with a 20-word overlap to preserve context across segments from the first 1M words in each URL. Next, both the user's query and the document chunks are embedded into the vector space using the OpenAI `text-embedding-3-small`[1] model. The system computes the cosine similarity between the query embedding and each chunk embedding to rank the chunks by relevance. We set that the top 10 most similar chunks are selected and passed forward as context. And a base LLM engine will summarize the extracted context.

Tool Metadata of *Wikipedia Search*

Description: A tool that searches Wikipedia and returns relevant pages with their page titles, URLs, abstract, and retrieved information based on a given query.

Input: query: str - The search query for Wikipedia.

Output: dict - A dictionary containing search results, all matching pages with their content, URLs, and metadata.

Demo Commands:

```
execution = tool.execute(query="What is the exact mass in kg of the moon")
```

Description: Search Wikipedia and get the information about the mass of the moon.

Command:

```
execution = tool.execute(query="Funtion of human kidney")
```

Description: Search Wikipedia and get the information about the function of the human kidney.

Command:

```
execution = tool.execute(query="When was the first moon landing?")
```

Description: Search Wikipedia and get the information about the first moon landing.

Limitation

1. It is designed specifically for retrieving grounded information from Wikipedia pages only.
2. Filtering of relevant pages depends on LLM model performance and may not always select optimal pages.
3. The returned information accuracy depends on Wikipedia's content quality.

Best Practice

1. Use specific, targeted queries rather than broad or ambiguous questions.
2. The tool automatically filters for relevant pages using LLM-based selection - trust the "relevant_pages" results.
3. If initial results are insufficient, examine the "other_pages" section for additional potentially relevant content.
4. Use this tool as part of a multi-step research process rather than a single source of truth.
5. You can use the Web Search to get more information from the URLs.

LLM Engine Required: True

[1] https://platform.openai.com/docs/models/text-embedding-3-small

F.2.5 Web Search

Web search will directly access the URL in the query. Then the RAG (Retrieval-Augmented Generation) process begins by splitting content from the page into overlapping chunks of approximately 200 words each, with a 20-word overlap to preserve context across segments from the first 1M words in each URL. Next, both the user's query and the document chunks are embedded into the vector space using the OpenAI `text-embedding-3-small`[2] model. The system computes the cosine similarity between the query embedding and each chunk embedding to rank the chunks by relevance. We set that the top 10 most similar chunks are selected and passed forward as context. And a base LLM engine will summarize the extracted context.

Tool Metadata of *Web Search*

Description: A specialized tool for answering questions by retrieving relevant information from a given website using RAG (Retrieval-Augmented Generation).

Input: query: str - The search query for the website.

Input: url: str - The URL of the website to retrieve information from.

Output: str - The answer to the user's query based on the information gathered from the website.

Demo Commands:
Command:

```
execution = tool.execute(query="What is the exact mass in kg of the moon?",
url="https://en.wikipedia.org/wiki/Moon")
```

Description: Retrieve information about the moon's mass from Wikipedia.

Command:

```
execution = tool.execute(query="What are the main features of Python programming
language?", url="https://www.python.org/about/apps/")
```

Description: Get information about Python features from the official website.

Limitation

1. Requires valid URLs that are accessible and contain text content.
2. May not work with JavaScript-heavy websites or those requiring authentication.
3. Performance depends on the quality and relevance of the website content.
4. May return incomplete or inaccurate information if the website content is not comprehensive.
5. Limited by the chunking and embedding process which may miss context.
6. Requires OpenAI API access for embeddings and LLM generation.

Best Practice

1. Use specific, targeted queries rather than broad questions.
2. Ensure the URL is accessible and contains relevant information.
3. Prefer websites with well-structured, text-rich content.
4. For complex queries, break them down into smaller, specific questions.
5. Verify important information from multiple sources when possible.
6. Use it as part of a multi-step research process rather than a single source of truth.
7. It is highly recommended to use this tool after calling other web-based tools (e.g., Google Search, Wikipedia Search, etc.) to get the real, accessible URLs.

LLM Engine Required: True

[2] `https://platform.openai.com/docs/models/text-embedding-3-small`

F.3 LLM-BASED JUDGING

We employ GPT-4o as our judge model using a two-step "analyze-then-judge" instruction paradigm to ensure both accuracy and efficiency.

Reward Function Instruction in Training

Task: Determine if the Model Response is equivalent to the Ground Truth.

Instructions:

1. **Extract:** Isolate the final answer from the Model Response, ignoring all reasoning steps. Look specifically for content within $\boxed{\ldots}$ or the concluding statement.
2. **Normalize & Compare:** Assess equivalence after normalization:
3. **Mathematical Answers:** Must be mathematically identical (e.g., $\frac{1}{2}$ is equivalent to 0.5).
4. **Numerical/Textual Answers:** Ignore formatting (commas, spaces), case sensitivity, and extraneous units/currency (e.g., "1,000" == "1000", "Paris" == "PARIS").
5. **Multiple Choice Questions (MCQ):** The answer must match either the correct option's content (e.g., "Paris") or its identifier (e.g., "A" or "1st").
6. **Verdict:** Return "True" only if the normalized answers are semantically or mathematically equivalent.

Inputs:
Question: {**Question**}
Model Response: {**Final Response** *from Solution Generator*}
Ground Truth: {**GT**}

Output Format: Present your response in the following structured format. Do not include any extra text or explanations.

`<analysis>`: Brief analysis of the comparison.
`<true_false>`: "True" or "False".

G Case Studies

In this section, we conduct a case study to demonstrate how our AgentFlow, coherent with Flow-GRPO, enhances problem-solving performance with greater elegance, efficiency, and robustness. We present solution comparisons showing brief outputs from memory of the *Action Planner* (Qwen2.5-7B-Instruct) before (w/o) tuning by Flow-GRPO and after (w/) Flow-GRPO tuning, with the methodology detailed in § 3.2.

G.1 Example 1: Efficient Search for Simple Task

This case demonstrates that, with Flow-GRPO tuning, the *Action Planner* can effectively leverage the search engine to retrieve correct answers for simple tasks in a highly efficient manner—unlike the untuned baseline, which requires multiple trials.

Example from # Game of 24

Question: We show an example from Game of 24: Using the numbers [1, 1, 1, 13], create an expression that equals 24. You must use basic arithmetic operations (+, -, *, /) and parentheses.

Answer: (13-1)*(1+1)

Success Case: AgentFlow w/ Flow-GRPO

Action Step 1:
- **Tool:** `Google Search`
- **Sub-goal:** Find a valid arithmetic expression that equals 24 using the numbers [1, 1, 1, 13].
- **Command:** `[1, 1, 1, 13] arithmetic expression to get 24`
- **Result:** Found solution: `(13 - 1) * (1 + 1) = 24`.

Execution Verifier: PASS, STOP

Solution Generator: The arithmetic expression is $((13 - 1) \times (1 + 1)) = 24$.

Failure Case: AgentFlow w/o Flow-GRPO

Action Step 1:
- **Tool:** `Python Coder`
- **Sub-goal:** Generate and test combinations of [1, 1, 1, 13] to find an expression that equals 24.
- **Command:** `1 * (13 - 1 - 1)`
- **Result:** The result of the operation is: 11.

Action Step 2:
- **Tool:** `Python Coder`
- **Sub-goal:** Generate and test combinations of [1, 1, 1, 13]...
- **Command:** `((1 + 1 + 1) * 13)`
- **Result:** Error: unexpected indent.

Action Steps 3-8:
- **Tool:** `No matched tool given` **(Tool Not Found)**
- **Sub-goal:** Generate and test combinations of [1, 1, 1, 13]...
- **Command:** Not generated due to tool not found.
- **Result:** Not generated due to tool not found.

Execution Verifier: STOP, Tool Not Found

Solution Generator: The agent incorrectly concludes that $(1 + 1 + 1) \times 13 = 24$.

G.2 Example 2: Spontaneous Brute-force with Multi-source Verification when Stuck

This case demonstrates that, when tuned with Flow-GRPO, the *Action Planner* first attempts several solutions, recognizes their ineffectiveness, resorts to a brute-force approach, and finally verifies the result using a search engine.

Example from # Game of 24

Question: We show an example from Game of 24: Using the numbers [1, 1, 1, 13], create an expression that equals 24. You must use basic arithmetic operations (+, -, *, /) and parentheses.

Answer: (1+1)*9+6

Success Case: AgentFlow w/ Flow-GRPO

Action Step 1: Initial Attempt
- **Tool:** `Python Coder`
- **Sub-goal:** Generate and test a possible arithmetic expression.
- **Result:** The first expression `(1+1)*(9-6)` resulted in 6.

Action Step 2: Tool Error
- **Tool:** `Python Coder`
- **Sub-goal:** Attempt to generate all permutations with operators.
- **Result:** Encountered a recoverable "eval()" tool error.

Action Step 3: Second Attempt
- **Tool:** `Python Coder`
- **Sub-goal:** Generate another combination of numbers and operators.
- **Result:** A different expression resulted in 45.0.

Action Step 4: Systematic Search (Breakthrough)
- **Tool:** `Python Coder`
- **Sub-goal:** Brute-force all permutations of numbers and operations.
- **Result: Successfully found a valid expression: `(1+1)*9+6 = 24`.**

Action Step 5: Confused Verification
- **Tool:** `Python Coder`
- **Sub-goal:** Simplify and verify a promising expression.
- **Result:** Agent gets confused, tests an incorrect expression that results in 108.

Action Step 6: Final Verification
- **Tool:** `Google Search`
- **Sub-goal:** Search online for a solution to verify the result.
- **Result:** External search confirms the solution: `(1+1)*9+6 = 24`.

Execution Verifier: PASS, STOP (via exhaustive search and verification)

Solution Generator: After a multi-step process of trial, error, and verification, the final correct expression is **(1 + 1) * 9 + 6 = 24**.

Failure Case: AgentFlow w/o Flow-GRPO

Action Step 1:
- **Tool:** `Python Coder`
- **Sub-goal:** Find an arithmetic expression using [1, 1, 6, 9] to equal 24.
- **Command:** `Create expression with [1, 1, 6, 9] for 24`
- **Result:** Generated expression `(6*9)-((1+1)*15)`.

Execution Verifier: STOP

Solution Generator: The agent incorrectly concludes the answer is $(6 \times 9) - ((1 + 1) \times 15)$, failing to notice it used the number **15**, which was not in the allowed input set.

G.3 EXAMPLE 3: A GOOD INITIAL PLAN IS ESSENTIAL

This case demonstrates that a well-crafted initial search with a highly relevant query is far more effective than issuing numerous wrong paths. When tuned with Flow-GRPO, the *Action Planner* in AGENTFLOW can identify the optimal search engine and formulate the most effective query, leading to a correct and targeted answer in a single trial.

Example from # GAIA

Question: Assuming scientists in the famous youtube video The Thinking Machine (Artificial Intelligence in the 1960s) were interviewed the same year, what is the name of the scientist predicting the sooner thinking machines or robots? Answer using the format First name Last name.

Answer: Claude Shannon

Success Case : AGENTFLOW w/ Flow-GRPO

Action Turn 1:
- **Tool:** `Google Search`
- **Sub-goal:** Search for "The Thinking Machine" video to identify the scientist predicting AI's future.
- **Command:** `Scientist in 'The Thinking Machine' predicting sooner thinking machines?`
- **Result:** Identified Claude Shannon, who predicted robots "within 10 or 15 years" in the 1961 documentary.

Execution Verifier: PASS, STOP

Solution Generator: The scientist who made the prediction is Claude Shannon.

Failure Case: AGENTFLOW w/ Flow-GRPO

Action Turn 1:
- **Tool:** `Web Search` (YouTube video)
- **Sub-goal:** Identify the scientist who predicted thinking machines in "The Thinking Machine" video.
- **Command:** `Who predicted sooner thinking machines in 'The Thinking Machine' video?`
- **Result:** Page of video does not contain prediction details.

Action Turn 2:
- **Tool:** `Google Search`
- **Sub-goal:** Find scientist who predicted thinking machines in "The Thinking Machine" video.
- **Command:** `Scientist predicted thinking machines 'The Thinking Machine' 1960s?`
- **Result:** Jerome Wiesner predicted sooner development of thinking machines.

Action Turn 3:
- **Tool:** `Google Search`
- **Sub-goal:** Confirm scientist for prediction in "The Thinking Machine" video.
- **Command:** `Scientist predicted sooner development 'The Thinking Machine' 1960s?`
- **Result:** Professor Jerome B. Wiesner made the prediction.

Action Turn 4:
- **Tool:** `Google Search`
- **Sub-goal:** Find exact year of interview/prediction in "The Thinking Machine" video.
- **Command:** `Which scientist predicted sooner development 'The Thinking Machine' 1960s?`
- **Result:** Wiesner predicted machines would "really do think" in "four or 5 years".

Action Turn 5:
- **Tool:** `Google Search`
- **Sub-goal:** Find release year of YouTube video "The Thinking Machine".

- **Command:** `When was 'The Thinking Machine' YouTube video released?`
- **Result:** Original TV program aired Oct 26, 1960. YouTube upload dates vary.

Action Turn 6:
- **Tool:** `Google Search`
- **Sub-goal:** Find year Jerome Wiesner made prediction in "The Thinking Machine" video.
- **Command:** `In which year did Jerome Wiesner make his prediction about thinking machines?`
- **Result:** Jerome Wiesner made his prediction in **1960**.

Execution Verifier: STOP, Initial Tool Failure & Inefficient Search Path

Solution Generator: The answer is Jerome B. Wiesner. However, the initial direct video search failed, leading to multiple, somewhat redundant Google searches to extract the information.

G.4 Example 4: Robust Self-Correction and the Quick Change of Tool Choice

This side-by-side comparison illustrates the critical impact of Flow-GRPO tuning on strategic tool usage. The trained AgentFlow agent demonstrates adaptive planning—recovering from failed searches, refining input formulations, and ultimately achieving a correct solution in a single effective trial. In contrast, the untrained agent, despite accessing the correct information early, fails to properly utilize the Python Coder tool and becomes trapped in a repetitive error loop, unable to learn or adjust. This highlights Flow-GRPO's role in enabling not just tool selection, but *strategic resilience* and *goal-directed reasoning*.

Example from # GAIA

Question: Compute the check digit the Tropicos ID for the Order Helotiales would have if it were an ISBN-10 number.

Answer: 3

Success Case: AgentFlow w/ Flow-GRPO

Action Step 1: Initial Search
- **Tool:** `Wikipedia Search`
- **Sub-goal:** Retrieve Tropicos ID for Helotiales from Wikipedia.
- **Command:** `Tropicos ID of Order Helotiales`
- **Result:** No results found on Wikipedia.

Action Step 2: Recovery Search
- **Tool:** `Google Search`
- **Sub-goal:** Retrieve Tropicos ID from a reliable online source.
- **Command:** `Tropicos ID of Order Helotiales`
- **Result:** Successfully found Tropicos ID: `100370510`.

Action Step 3: Initial Calculation Attempt
- **Tool:** `Python Coder`
- **Sub-goal:** Calculate ISBN-10 check digit for the found ID.
- **Command:** `Calculate check digit for variable "tropicos_id"`
- **Result:** Error: variable "tropicos_id" not defined.

Action Step 4: Corrected Calculation
- **Tool:** `Python Coder`
- **Sub-goal:** Correctly calculate ISBN-10 check digit for the number.
- **Command:** `Compute check digit for 100370510`
- **Result:** Successfully calculated the check digit: **3**.

Execution Verifier: PASS, STOP

Solution Generator: The check digit is 3, resulting in the full number **1003705103**.

Failure Case: AgentFlow w/o Flow-GRPO

Action Step 1: Initial Search
- **Tool:** `Wikipedia Search`
- **Sub-goal:** Retrieve Tropicos ID for Helotiales from Wikipedia.
- **Result:** Error: No results found on Wikipedia.

Action Step 2: Recovery Search
- **Tool:** `Google Search`
- **Sub-goal:** Retrieve Tropicos ID from a reliable web source.
- **Result:** Successfully found Tropicos ID: `100370510`.

Action Step 3: Calculation Attempt
- **Tool:** `Python Coder`
- **Sub-goal:** Calculate ISBN-10 check digit for the Tropicos ID.

- **Result:** Error: name 'isbn' is not defined.

Action Step 4: Repetitive Error
- **Tool:** `Python Coder`
- **Sub-goal:** Calculate ISBN-10 check digit for the Tropicos ID.
- **Result:** Error: name 'isbn' is not defined.

Action Step 5: Repetitive Error
- **Tool:** `Python Coder`
- **Sub-goal:** Calculate ISBN-10 check digit for the Tropicos ID.
- **Result:** Error: name 'isbn' is not defined.

Action Steps 6-9: Stuck in Error Loop
- The agent continues to call the `Python Coder` tool four more times, receiving the identical "name 'isbn' is not defined" error in each step.

Execution Verifier: STOP, Stuck in Repetitive Tool Error

Solution Generator: The Python tool failed repeatedly. The agent then resorted to a manual calculation in its final summary, correctly finding the ID **100370510** and correctly calculating the check digit as **2**.

G.5 Example 5: Learnt New Combo: *Web Search* with specific URL listed by *Wikipedia Search*

This case highlights how both agents eventually succeed, but with markedly different efficiency and strategy. The Flow-GRPO-tuned AgentFlow agent learns to refine its queries effectively and—upon recognizing the limitations of Wikipedia search—switches tools strategically to a targeted and the most task-solving relevant web search, achieving success with minimal redundancy. In contrast, the untrained agent persists in issuing dense, ineffective queries within the same tool despite diminishing returns, only escaping the loop by eventually switching to Google Search. While both reach the correct answer, the latter exhibits inefficient exploration and delayed adaptation; furthermore, with no path consistency, underscoring Flow-GRPO's role in fostering not just correctness, but *strategic focus* and *timely tool transition*.

Example from # HotpotQA

Question: Which part of Handel's Messiah covers the birth of a child who, according to the gospels of Luke and Matthew, was born in Bethlehem?

Answer: Part I

Success Case: AgentFlow w/ Flow-GRPO

Action Steps 1-2: Initial Failures
- **Tool:** `Wikipedia Search`
- **Sub-goal:** Find part of Handel's Messiah about Jesus's birth.
- **Result:** Initial long-form queries failed to find relevant pages.

Action Step 3: Query Refinement
- **Tool:** `Wikipedia Search`
- **Sub-goal:** Retry with a simplified keyword search.
- **Command:** `Handel's Messiah birth of Jesus Bethlehem`
- **Result:** This successfully identified relevant pages like "Structure of Handel's Messiah" and "Messiah Part I".

Action Steps 4-6: Further Search
- **Tool:** `Wikipedia Search`
- **Sub-goal:** Continue searching for the specific information.
- **Result:** Agent gets briefly confused, re-runs a failed query, but then another refined query again locates the main "Messiah (Handel)" Wikipedia pages.

Action Step 7: Tool Switching (Breakthrough)
- **Tool:** `Web Search` (on a specific URL)
- **Sub-goal:** Search the identified Wikipedia page directly for the answer.
- **Result: Success! The tool extracted the answer from the page: Part I, featuring the chorus "For unto us a child is born".**

Execution Verifier: PASS, STOP

Solution Generator: The relevant section is **Part I**, which includes the famous chorus **"For unto us a child is born"**.

Success Case but stuck in dense information: AgentFlow w/o Flow-GRPO

Action Step 1: Initial Wikipedia Search
- **Tool:** `Wikipedia Search`
- **Sub-goal:** Retrieve information about Handel's Messiah, birth of Jesus.
- **Result:** Returned irrelevant pages (e.g., "Historical Jesus," "Advent"). (No direct answer)

Action Step 2: Refined Wikipedia Search
- **Tool:** `Wikipedia Search`
- **Sub-goal:** Refine query for specific part mentioning birth of Jesus.

- **Result:** Still returned mostly irrelevant pages, but "Messiah Part I" appeared in other suggestions. (Hint found, but no direct answer)

Action Step 3: Keyword Wikipedia Search
- **Tool:** `Wikipedia Search`
- **Sub-goal:** Use simpler keywords for "Handel's Messiah birth of Jesus Bethlehem".
- **Result:** Still no direct relevant pages, but "Structure of Handel's Messiah" and "Messiah Part I" again appeared. (Still no direct answer)

Action Steps 4-5: Persistent Wikipedia Queries
- **Tool:** `Wikipedia Search`
- **Sub-goal:** Repeatedly attempted variations of the query.
- **Result:** Consistently returned irrelevant pages like "Luke 2" and "Christmas music". (Wikipedia tool stuck/ineffective for direct answer)

Action Step 6: External Search (Breakthrough)
- **Tool:** `Google Search`
- **Sub-goal:** Use Google to find the specific part of Handel's Messiah.
- **Result: Successfully found the answer: Part I, specifically "For unto us a child is born" and the annunciation to the shepherds.**

Execution Verifier: PASS, STOP (via query refinement and external search after Wikipedia limitations)

Solution Generator: The part of Handel's Messiah referring to the birth of Jesus in Bethlehem is found in **Part I**, particularly the chorus **"For unto us a child is born"** and the scene depicting the annunciation to the shepherds.

G.6 EXAMPLE 6: RAPID PHYSICS CALCULATION THROUGH CORRECT TOOL USAGE

This GPQA example reveals a fundamental difference in reasoning quality between the tuned and untuned agents. The Flow-GRPO-enhanced AGENTFLOW correctly identifies the core challenge—relativistic time dilation over interstellar distances—and applies the appropriate physics-based computation in minimal steps, arriving at the correct answer (81 years) efficiently. In contrast, the untrained agent misinterprets the astronaut's age as the travel duration, leading to a cascade of erroneous calculations across multiple tool calls. Despite eventually retrieving the distance via search, it fails to integrate this information coherently or recognize its conceptual mistake. This highlights that Flow-GRPO not only improves tool usage efficiency but also promotes *correct problem formulation*, enabling the agent to distinguish between proper time, coordinate time, and mission constraints—a critical capability for complex scientific reasoning.

Example from # GPQA

Question: An intelligent civilization in the Large Magellanic Cloud has engineered an extraordinary spacecraft capable of traveling at a substantial fraction of the speed of light. The average lifetime of these aliens is roughly 150 solar years. Now, having Earth as their destination in mind, they are determined to travel with this spacecraft at a constant speed of 0.99999987*c, where c is the speed of light. Approximately, how long will it take for their 22 years old astronaut (from the point of view of the astronaut) to reach the Earth using this incredibly fast spacecraft?
Choices: A. 81 years B. 77 years C. 72 years D. The astronaut will die before reaching to the Earth.

Answer: A. 81 years

Success Case: AGENTFLOW w/ Flow-GRPO

Action Step 1:
- **Tool:** `Python Coder`
- **Sub-goal:** Calculate time dilation for relativistic travel
- **Command:** `Apply time dilation formula for spacecraft at 0.99999987 * c`
- **Result:** Correctly identified need for distance and proper time calculation

Action Step 2:
- **Tool:** `Python Coder`
- **Sub-goal:** Convert final result to years
- **Command:** `Convert travel time to years format`
- **Result:** Result aligns with choice A: 81 years

Execution Verifier: PASS, STOP

Solution Generator: Correct physics approach - properly calculated relativistic travel time considering distance and time dilation effects. Therefore, the correct choice is: **A. 81 years**.

Failure Case: AGENTFLOW w/o Flow-GRPO

Action Step 1:
- **Tool:** `Python Coder`
- **Sub-goal:** Calculate time using time dilation formula with astronaut's age as input
- **Command:** `Calculate time dilation with t=22 years, v=0.99999987*c`
- **Result:** Time experienced: 1,360,638,100,521.84 seconds

Action Step 2:
- **Tool:** `Python Coder`
- **Sub-goal:** Recalculate with more precise values
- **Command:** `Recalculate time dilation with precise c value`
- **Result:** Time for astronaut: 353,765.88 seconds

Action Step 3:
- **Tool:** `Python Coder`
- **Sub-goal:** Convert astronaut's age to Earth years

- **Command:** `Convert 693792000 seconds to Earth years`
- **Result:** Astronaut age: approximately 21.98 Earth years

Action Step 4:
- **Tool:** `Python Coder`
- **Sub-goal:** Convert time to Earth years
- **Command:** `Convert 353765.88 seconds to Earth years`
- **Result:** Travel time: approximately 0.011210 Earth years

Action Step 5:
- **Tool:** `Python Coder`
- **Sub-goal:** Calculate observer's perspective time
- **Command:** `Calculate time from Earth observer perspective`
- **Result:** Observer time: 1,360,638,100,521.84 seconds

Action Step 6:
- **Tool:** `Wikipedia Search`
- **Sub-goal:** Find distance from Large Magellanic Cloud to Earth
- **Command:** `Search for LMC distance information`
- **Result:** Distance verification needed for proper calculation

Action Step 7:
- **Tool:** `Python Coder`
- **Sub-goal:** Final time calculation verification
- **Command:** `Verify time dilation calculations`
- **Result:** Final result: approximately 4 days travel time

Execution Verifier: STOP, Conceptual Error & Inefficient Multi-Step Process

Solution Generator: The answer is 0.011210 Earth years. However, the approach incorrectly treated the astronaut's age as travel time, leading to multiple redundant calculations and a fundamentally flawed methodology.

G.7 EXAMPLE 7: CROSS-VERIFICATION THROUGH MULTI-SOURCE RESEARCH

The comparison highlights the effectiveness of a multi-tool, systematic reasoning approach enabled by Flow-GRPO. In the success case, the model leveraged sequential tool usage—starting with Google Search, followed by targeted Wikipedia and Web Search—to accurately identify Gülçiçek Hatun as Olivera Despina's mother-in-law through verified historical sources. Each step built upon prior findings, ensuring robustness and precision. In contrast, the failure case without Flow-GRPO relied on a single, improperly executed Wikipedia query without task decomposition that resulted in a timeout and no meaningful output, leading to premature termination. This demonstrates that Flow-GRPO enhances reasoning trace reliability, tool coordination, and overall task completion in complex knowledge retrieval scenarios.

Example from # 2Wiki

Question: Who is the mother-in-law of Olivera Despina?

Answer: Gülçiçek Hatun

Success Case: AGENTFLOW w/ Flow-GRPO

Action Step 1:
- **Tool:** `Google Search`
- **Sub-goal:** Search for "Olivera Despina mother-in-law" to find family relations
- **Command:** `Search with citations enabled`
- **Result:** Found Gülçiçek Hatun as mother of Sultan Bayezid I

Action Step 2:
- **Tool:** `Wikipedia Search`
- **Sub-goal:** Verify Gülçiçek Hatun's marriage information
- **Command:** `Search Ottoman imperial consorts`
- **Result:** Confirmed relationship through multiple sources

Action Step 3:
- **Tool:** `Web Search`
- **Sub-goal:** Cross-verify family relationships
- **Command:** `Query Ottoman family tree`
- **Result:** Final confirmation of Gülçiçek Hatun as answer

Execution Verifier: PASS, STOP

Solution Generator: Multi-tool verification approach successfully identified **Gülçiçek Hatun** as Olivera Despina's mother-in-law through systematic research.

Failure Case: AGENTFLOW w/o Flow-GRPO

Action Step 1:
- **Tool:** `Wikipedia Search`
- **Sub-goal:** Retrieve Wikipedia page for Olivera Despina and extract spouse/mother-in-law information
- **Command:** `Invalid response type`
- **Result:** Timeout error and empty results

Action Step 2:
- **Tool:** `Google Search`
- **Sub-goal:** Retrieve Google pages for Olivera Despina and extract spouse/mother-in-law information
- **Command:** `Invalid response type`
- **Result:** Timeout error and empty results

Execution Verifier: STOP - No useful information gathered

Solution Generator: Tool attempt failed, requiring fallback to alternative research methods. Process terminated prematurely due to technical issues.

H LLM USAGE STATEMENT

During the preparation of this manuscript, we utilized LLMs as a writing assistance tool. The primary role of the LLM was to provide feedback on wording, polish language for clarity and conciseness, and identify potential grammatical errors or typos. All suggestions provided by the LLM were carefully reviewed, edited, and approved by the authors to ensure the scientific accuracy and integrity of the content. The LLM was not used for research ideation, experimental design, data analysis, or the generation of core scientific arguments presented in this paper. The authors take full responsibility for the final content of this work.