FROM CHAINS TO ATOMS: A UNIFIED MATH AGENT WITH ATOMIC CAPABILITY-GROUNDED REASONING

Anonymous authors
Paper under double-blind review

ABSTRACT

Mathematical reasoning serves as a critical testbed for LLMs, offering both objectively verifiable outcomes and a wide spectrum of complexity. Despite recent progress, existing approaches remain fundamentally limited: (1) linear Chain-of-Thought scales by extending reasoning, often leading to overthinking and local optimality; (2) agent systems usually lack effective alignment of mathematical atomic capabilities between planning, execution, and feedback, preventing the specialized action of agent reasoning. We introduce ATOMMATH-AGENT, a Unified Math Agent based on *atomic capability-grounded reasoning unit*, which comprises one mathematical atom-goal paired with an atomic capability. Agent's planner forms these units a dependency graph shared by planning, execution, and memory. The Planner specifies what mathematical capability is required, while Executor determines a capability-conditioned atom skill for how to realize it. The atomic unit also defines a verification protocol based on it mathematical atomic capability, which is used for feedback of results and dynamic planning of subsequent steps. This shared representation turns atomic reasoning into the organizing principle of the agent. Across seven backbones, ATOMMATH-AGENT improves text-only and multimodal accuracy by 5.0 and 4.4 points on average. Tool-matched comparisons and controlled ablations further show gains and support the roles of atomic units and replanning. We further apply ATOMMATH-AGENT to data synthesis and improve agentic mid-training, with the clearest transfer from math to other agent tasks. These results provide insights and demonstrate atomic capability-grounded reasoning as a unified thought for LLM-based agent community.[1]

1 INTRODUCTION

Multimodal large language models (MLLMs) have substantially improved mathematical problem solving across textual expressions, diagrams, tables, and geometric relations Kuang et al. (2025b); Wang et al. (2024a; 2026); Yang et al. (2025). This progress has been driven by mathematical training Li et al. (2024); Yang et al. (2024), extended chain-of-thought (CoT) reasoning Guo et al. (2025); Zhong et al. (2024), and agent frameworks that interleave planning, tool use, and feedback Yang et al. (2023); Shinn et al. (2023); Yan et al. (2025). These approaches provide stronger performance, but their intermediate steps are commonly represented as natural language. Although formal mathematical languages exist for proof and verification, their adaptability in general mathematical fields is relatively low Lin et al. (2024). Consequently, a step's mathematical purpose, required capability, dependencies, admissible output, and validation criterion are only implicitly connected. A planner may propose a plausible subgoal while the executor selects an incompatible tool.

This mismatch is particularly consequential in mathematics. Early approaches to mathematical problem-solving in LLMs primarily rely on instruction-following capabilities of large-scale mathematical corporaLi et al. (2024); Yang et al. (2024) and extending Chain-of-Thought reasoning length Guo et al. (2025); Zhou et al. (2025); Sprague et al.. As task complexity increases, a solution often involves alternating between various mathematical atomic capabilities, such as symbol recognition, modeling, computation, and proof construction. Simply extending a single reasoning chain does not by itself better reveal which atomic capability a particular step requires, nor does it provide a targeted check plan for that step, resulting in reasoning more disordered and overthinking. Agents' generic

[1]Code is available at AtomMath-agent-Code; the project page is atommath-agent-page.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

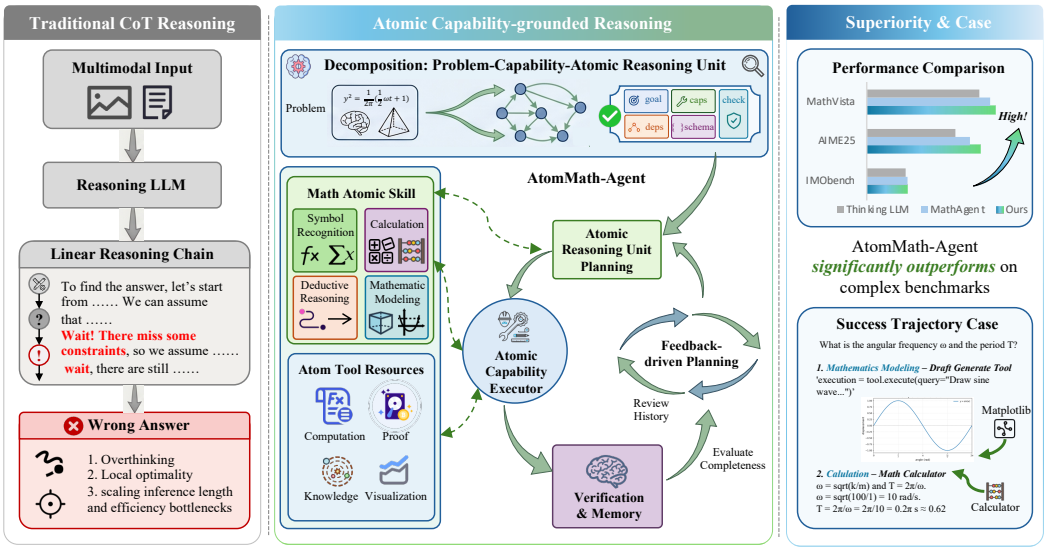

Figure 1: Free-form reasoning chains versus capability-grounded atomic reasoning in ATOMMATH-AGENT. Each atomic unit makes its subgoal, dependency, execution policy, and validation contract explicit.

task decomposition improves this control Gou et al.; Yan et al. (2025); Xiong et al. (2025); Liu et al., but its nodes still lack effective and unified alignment of mathematical atomic capabilities between planning, execution, and feedback, preventing the specialized action of agent reasoning. We therefore study the following hypothesis: **representing each intermediate step as a math atomic capability-grounded, dependency-aware, and checkable atomic reasoning unit improves mathematical agents beyond generic decomposition and tool access**. More importantly, although this atomic capability corresponds to mathematics, we also hope that this atomized reasoning organization can benefit more general agent reasoning and training than as a task-specific mathematical trick.

We introduce ATOMMATH-AGENT, an atomic capability-grounded reasoning agent framework. As illustrated in Figure 1, the Planner constructs a directed acyclic graph whose nodes jointly specify a subgoal, a math atomic capability, predecessor dependencies, an output schema, and a validation contract. The atomic capability then selects a math atom skill, a capability-conditioned execution policy containing semantic instructions, interface constraints, and candidate tool resources. Then the executor calls a concrete tool or sub-agent, obtains the evidence, and verifies it to store in a structured Memory, allowing the Planner to keep, modify, retry, or replan subsequent nodes. Therefore, we have implemented a unified atomic reasoning agent which unifies planning, execution, and verification through basic atomic unit representations, unifies agent reasoning behavior with diverse basic mathematical atomic capabilities, and unifies different mathematical problem-solving strategies in text and multimodal tasks through atomic unit combinations.

Our experiments evaluate both end-to-end utility and the proposed mechanism. Across seven backbones, ATOMMATH-AGENT improves the text-only and multimodal accuracy by 5.0 and 4.4 points on average. More importantly, under the same tool access, our atomic capability-grounded agent outperforms all evaluated agent frameworks. The ablation study further shows positive atomic-reasoning gains both with and without tools, while taxonomy and replanning ablations support the roles of capability semantics, granularity, and feedback-driven control. Finally, we use the same typed representation to synthesize agent-training trajectories and observe the clearest transfer to planning and feedback. Together, these experiments position capability-grounded atomic reasoning as a unified agent design principle rather than mathematical solution techniques. Our main contributions are:

- We propose ATOMMATH-AGENT, formulate mathematical agent reasoning through mathematical atomic capability units, toward a more systematic and specialized math reasoning logic representation and unified agent architecture.
- We introduce a shared interface atomic capability-grounded Planner–Executor–Feedback architecture in which mathematical capabilities target execution atomic skills, execution skills route optional tools, and long-short term atomic feedback drives dynamic planning.

- Significant performance improvements across multiple advanced LLM backbones and challenging benchmarks, and effective ablations demonstrate superior performance of ATOMMATH-AGENT while achieving a balance between accuracy and efficiency.
- We construct large-scale capability-grounded trajectories and validate their strongest effectiveness both in math agent reasoning and other agentic tasks such as code and tool call.

2 Related Work

2.1 Mathematical Reasoning in LLMs

Mathematical LLMs have progressed through several complementary directions. Data-centric methods improve domain knowledge through large mathematical corpora and specialized instruction tuning Li et al. (2024); Yang et al. (2024); Shao et al. (2024). Inference-time methods elicit intermediate derivations, search alternative branches, or allocate additional computation to difficult problems Guo et al. (2025); Zhou et al. (2025); Sprague et al.; Yao et al. (2023). Process supervision and verification methods instead assess intermediate states with reward models or verification-oriented objectives Lightman et al. (2023); Wang et al. (2024b); Zhang et al. (2025e); Du et al. (2025); Zhang et al. (2025a). These approaches improve mathematical accuracy, but their intermediate states are generally free-form derivations, where the mathematical capability required by a step and its execution and validation interfaces are not necessarily represented. Our work studies whether a typed intermediate atomic capability unit can coordinate planning, resource selection, and feedback within one agent's reasoning, which will align mathematical cognitive units.

2.2 Agent-based Mathematical Reasoning

Agent systems interleave reasoning with actions, external tools, memory, and iterative feedback Shinn et al. (2023); Gou et al.; Xiong et al. (2025); Liu et al.. Representative systems such as ToRA (Gou et al.), MathChat (Wu et al., 2023), and MathAgent-PRER (Liao et al., 2024) employ Planner–Reasoner–Executor-style decompositions, while MM-Agent (Liu et al.) and MathAgent (Yan et al., 2025) target real-world modeling or multimodal educational scenarios. PRM-based and debate-based methods further improve intermediate reasoning quality through reranking or multi-agent interaction (Wang et al., 2024b; Lightman et al., 2023; Ma et al., 2025; Zhang et al., 2025d;b). However, existing agents typically decompose problems by *roles*, *tools*, or *task stages*, not by the underlying mathematical capability being exercised. Recent work on atomic reasoning and skill decomposition Zhang et al. (2025c); Kuang et al. (2025a) suggests that mathematical competence is better viewed as a composition of specialized abilities. AtomMath-Agent inherits the Planner–Executor–Memory loop, and its distinguishing contribution is the *unifed reasoning units* shared by those components, where every execution node is guided by an atomic capability unit with explicit dependencies, output schema, and validation contract, rather than only a role, stage, or tool call.

3 Methods

AtomMath-Agent is an atomic capability-grounded agent framework for multimodal mathematical reasoning. Its core innovation is to model complex mathematical problems as multiple foundational atomic reasoning units, shared by planning, execution, feedback, and memory. This exposes the agent's task planning to possess greater mathematical interpretability and rigor, while allowing execution to specialize by mathematical capability.

3.1 Problem Formulation and Atomic Reasoning Units

Let $x = (x^{\text{text}}, x^{\text{img}})$ denote a mathematical problem, where x^{img} may be empty. Let $\mathcal{C} = \mathcal{C}^{\text{act}} \cup \mathcal{C}^{\text{fb}}$ be the mathematical capability taxonomy. Action capabilities $\mathcal{C}^{\text{act}}$ describe operations that produce intermediate mathematical results, such as recognition, calculation, spatial reasoning, formalization, and modeling. Feedback capabilities $\mathcal{C}^{\text{fb}}$ describe operations that assess, revise, or extend the current solution state, such as theorem checking, reflection, and knowledge acquisition. The full taxonomy and its primary module assignments are given in Appendix A. We represent the t-th reasoning unit as

$$u_t = (g_t, c_t, D_t, o_t, \phi_t), \tag{1}$$

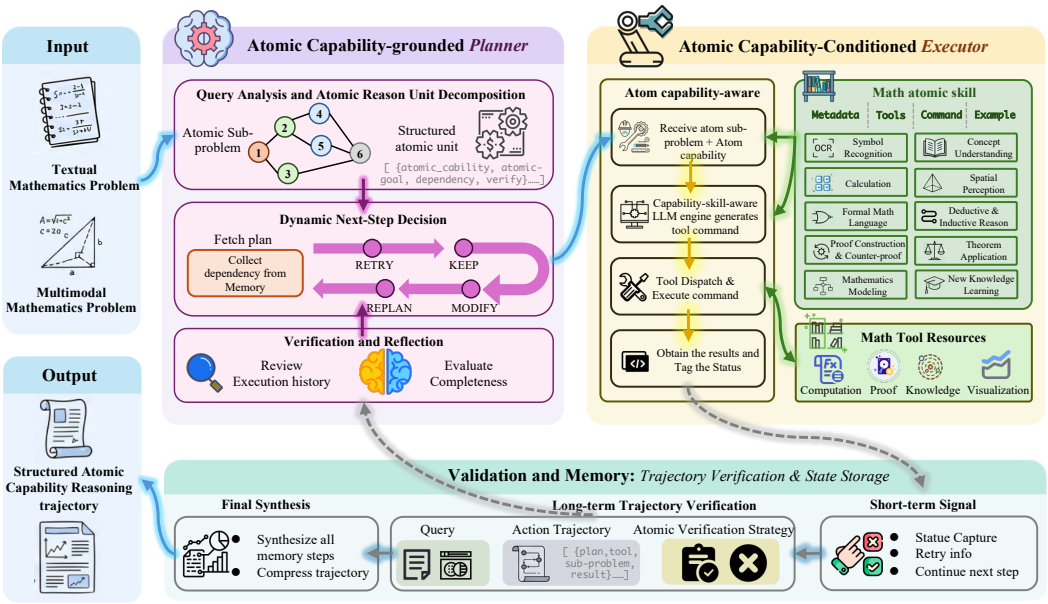

Figure 2: Overview of ATOMMATH-AGENT. A capability-typed atomic unit is the shared interface among planning, execution, feedback, and memory.

where g_t is one mathematical subgoal, $c_t \in \mathcal{C}$ is its primary mathematical atomic capability, D_t is the set of predecessor units whose verified outputs are required, o_t is an output schema, and ϕ_t is a validation contract. We call u_t *atomic* when it has one primary objective and its output can be independently (i) produced by the Executor, (ii) consumed by successor units through D_t, and (iii) checked against o_t and ϕ_t.

The units form a directed acyclic graph $G = (V, E)$, with $V = \{u_t\}$ and $(u_i, u_t) \in E$ iff $u_i \in D_t$. For capability c, an Atom Skill $s(c)$ is a capability-conditioned execution policy containing mathematical atomic capability semantic guidance, tool interface constraints, and candidate tool resources $\mathcal{T}(c)$. A resource may be an external tool, the backbone model, or a sub-agent. Executing these typed units produces a trajectory whose capability, resource, result, and feedback fields remain:

$$\pi = \{(a_t, u_t, s_t, \tau_t, r_t, \sigma_t)\}_{t=1}^{T}, \tag{2}$$

where a_t is the Planner's control action, $s_t = s(c_t)$, $\tau_t \in \mathcal{T}(c_t) \cup \{\varnothing\}$ is the selected resource, r_t is the execution result, and σ_t is structured feedback. The design objective is to find a trajectory that balances answer correctness and end-to-end inference cost:

$$\pi^* = \arg \max_{\pi \in \Pi(x, \mathcal{C})} \Pr(y = y^\star \mid x, \pi) - \lambda \, \mathrm{Cost}(\pi), \tag{3}$$

where $\mathrm{Cost}(\pi)$ includes model tokens, tool execution, and iterative control rounds.

3.2 OVERALL WORKFLOW

Figure 2 illustrates the complete inference and solution process of our ATOMMATH-AGENT. The planner first determines which basic mathematical atomic abilities are required to solve the problem, breaks it down into corresponding atomic reasoning units, and constructs a graph G. At the start of the solution process, it selects the next ready atomic reasoning unit to execute based on its dependencies and current memory state. During execution, the corresponding atomic reasoning unit passes the current atomic problem, historical solution information from memory, and related dependencies to its corresponding mathematical atomic skills, further deciding on specific solution strategies and invoking relevant tools and resources:

$$\text{Mathematical Capability } c_t \longrightarrow \text{Atom Skill } s(c_t) \longrightarrow \text{Optional Resource } \tau_t, \tag{4}$$

so the Planner specifies *what mathematical capability is required*, while the Executor decides *how to realize it*. After obtaining the execution result, it performs tool call verification at the atomic unit level and returns to the complete solution process to verify whether the current atomic unit's solution has made progress towards the complete trajectory, storing this information in memory and proceeding

with the next planning iteration. It may KEEP the planned unit, MODIFY its contract, RETRY a failed unit, or REPLAN the remaining graph. The result is checked against the unit's validation contract, appended to Memory. Algorithm 1 summarizes this process.

3.3 ATOMIC CAPABILITY-GROUNDED PLANNER

Global atomic reasoning unit planning. For a problem, the planner first determines which mathematical atomic capabilities are needed to solve it and organizes them into a DAG of atomic reasoning units. Each unit selects one mathematical atomic capability c_t with its subgoal, identifies predecessors D_t, defines the expected output format o_t, and a validation contract ϕ_t. This representation distinguishes mathematical dependencies from execution order: independent units can be scheduled separately, whereas a successor can only receive the outputs in D_t. The Planner also estimates a step budget from problem complexity to avoid breaking down atomic units into excessively small pieces. Appendix B.1 provides more details.

Algorithm 1 Capability-grounded inference loop.

Require: Problem x; capabilities $\mathcal{C}$; skill policies $\mathcal{S}$; budget B
Ensure: Answer y and trajectory memory M

$M_0 \leftarrow \emptyset$; $G \leftarrow \textsc{Decompose}(x, \mathcal{C}, B)$
for $t = 1$ to B **do**
  $(a_t, u_t) \leftarrow \textsc{Planner}(x, G, M_{t-1})$
  if $a_t = \textsc{Replan}$ **then**
    $G \leftarrow \textsc{Replan}(x, M_{t-1}, \mathcal{C})$
    continue
  else if $a_t = \textsc{Retry}$ **then**
    $u_t \leftarrow \textsc{LastFailed}(M_{t-1})$
  else if $a_t = \textsc{Modify}$ **then**
    $u_t \leftarrow \textsc{Modify}(u_t, M_{t-1}, \mathcal{C})$
  end if
  $c_t \leftarrow \textsc{Capability}(u_t)$; $s_t \leftarrow s(c_t)$
  $\tau_t \leftarrow \textsc{SelectResource}(s_t, u_t, M_{t-1})$
  $r_t \leftarrow \textsc{Execute}(s_t, \tau_t, u_t, M_{t-1})$
  $\sigma_t \leftarrow \textsc{Verify}(u_t, r_t, M_{t-1}, G)$
  $M_t \leftarrow M_{t-1} \cup \{(a_t, u_t, s_t, \tau_t, r_t, \sigma_t)\}$
  if $\sigma_t = \textsc{Stop}$ **then**
    return $\textsc{Summarize}(M_t), M_t$
  end if
end for
return $\textsc{Summarize}(M_B), M_B$

Verifier-driven dynamic planning. After global planning, the Planner conditions on G and M_{t-1} for dynamic planning. Based on the executed trajectory and its verification results, KEEP executes the next unit. MODIFY revises the next atomic unit, including its subgoal and capability when new evidence invalidates the original planning. RETRY re-executes the last failed unit. REPLAN reconstructs the unresolved portion of G when the current dependency structure is no longer appropriate. Verified predecessor outputs are injected as a structured dependency digest, which makes replanning local while retaining global revision as a fallback.

3.4 ATOMIC CAPABILITY-CONDITIONED EXECUTION

The Executor implements specialization through atomic skills corresponding to mathematical atomic capabilities and associated expert tool resources. Given atomic units to be solved, it loads $s(c_t)$, with three components: *semantic specialization* defines the mathematical atomic capability and its applicability; *interface specialization* specifies admissible inputs, outputs, and parameter schemas; and *resource specialization* ranks candidate tools or model-based procedures. The Executor converts g_t and the outputs in D_t into a valid request, selects τ_t, and returns both the result and an execution record. A tool failure is represented explicitly rather than silently replaced by an unrecorded answer. The Atom Skill Library originates from predefined hierarchical mathematical atomic capabilities and is the implementation layer of the routing hierarchy in Equation 4. An atomic skill can contain multiple tool resources with different applicable functions, and multiple atomic skills can share different applicability modes of a single tool resource. Detailed information on skill cards, runtime modes, tool interfaces, and specific mathematical tools is shown in the appendices C.1 and C.2. All of these tools are offline to avoid result hacking.

3.5 VALIDATION, FEEDBACK, AND MEMORY

After execution, each atomic reasoning unit organizes three types of checks according to its validation contract $\phi_t = (\phi_t^{\mathrm{op}}, \phi_t^{\mathrm{loc}}, \phi_t^{\mathrm{glob}})$. Different atomic reasoning units will be based on different mathematical atomic capabilities, define and implement different mathematical validation strate-

Table 1: Accuracy (%) over different backbones. CM, MV, MVa, and OB denote CounterMath, MathVision, MathVista, and OlympiadBench-Math. **Green bold** and underline mark the best and second results.

Backbone	Method	Text-only reasoning				Multimodal reasoning			
		AIME25	IMO	CM	Avg.	MV	MVa	OB	Avg.
Open-source backbones									
Qwen3-8B-VL	Base	20.0	3.0	52.4	25.1	53.9	59.7	15.8	43.1
	+Ours	24.4	12.5	54.1	**30.3**↑	55.9	62.2	19.1	**45.7**↑
Qwen3-30B	Base	43.3	20.8	69.2	44.4	60.2	70.1	17.6	49.3
	+Ours	51.1	23.5	73.5	**49.4**↑	64.4	73.8	22.5	**53.6**↑
Qwen3-235B	Base	50.0	24.1	72.2	48.8	66.5	78.2	34.7	59.8
	+Ours	56.7	26.3	78.9	**54.0**↑	68.5	82.3	38.0	**62.9**↑
Proprietary backbones									
GPT-5.4-mini	Base	63.3	25.4	76.8	55.2	68.0	70.0	49.2	62.4
	+Ours	70.0	29.1	81.2	**60.1**↑	71.2	83.5	59.4	**71.4**↑
GPT-5.4	Base	83.3	27.6	83.4	64.8	75.2	82.0	64.8	74.0
	+Ours	86.7	29.5	85.2	**67.1**↑	77.5	85.1	69.0	**77.2**↑
Gemini-3-flash	Base	60.0	26.5	78.2	54.9	69.9	76.3	53.4	66.5
	+Ours	76.7	30.4	85.4	**64.2**↑	72.5	85.4	57.9	**71.9**↑
Gemini-3.1-Pro	Base	93.3	30.0	84.6	69.3	80.3	89.4	69.6	79.8
	+Ours	**94.4**	**32.6**	**90.5**	**72.5**↑	**83.5**	**92.5**	**72.6**	**82.9**↑

gies, while maintaining the same operational interface. *Operational checks* determine whether the selected resource executed successfully and returned a parseable object. *Local mathematical checks* compare r_t with the unit's output schema and prescribed constraints, such as symbolic form, theorem preconditions, or consistency with predecessor values when these checks are specified. *Global checks* assess whether the accumulated trajectory is coherent and sufficient to answer the original problem. These checks are evaluated jointly and aggregated into one control signal $\sigma_t \in \{\text{RETRY}, \text{CONTINUE}, \text{STOP}\}$, which, along with the relevant check results and reasons, enters the memory. Memory stores the original input and, for every unit, its planner action, capability, skill and resource call, returned results, and verify feedback. A RETRY signal attaches failure metadata to the same unit; CONTINUE releases successor units whose dependencies are satisfied; and STOP triggers final answer synthesis. The complete reasoning process will be maintained in memory, so the trajectory retains explicit planning and feedback fields for mid-term training studies in Section 5.

4 EXPERIMENTS

4.1 EXPERIMENTAL SETUP

We evaluate ATOMMATH-AGENT with open-source backbones, **Qwen3-8B-VL**, **Qwen3-30B-A3B-VL**, and **Qwen3-235B-A22B-VL** Yang et al. (2025), and proprietary backbones, **GPT-5.4-mini**, **GPT-5.4**[2], **Gemini-3-flash**, and **Gemini-3.1-pro-preview**[3]. Framework baselines include **AutoGen** Wu et al. (2024), **OctoTools** Lu et al. (2025b), and **MathAgent** Yan et al. (2025). We evaluate benchmarks (AIME25, IMOBENCH, COUNTERMATH), (MATHVISION, MATHVISTA, and OLYMPIADBENCH-MATH) (Li et al.; Wang et al., 2024a; Lu et al., 2023; He et al., 2024). All reported scores are means over three independent runs. Table 2 retains the framework's native configuration, whereas Figure 3 matches the same tool access, with more implementation details in Appendix D.2.

4.2 MAIN RESULTS

Performance across different base models Table 1 shows consistent gains across all seven backbones. Averaged across backbones, ATOMMATH-AGENT improves the text-only and multimodal macro averages by 5.0 and 4.4 points, respectively. The largest text gain occurs on Gemini-3-flash (+9.3), while the largest multimodal gain occurs on GPT-5.4-mini (+9.0). The gains are smaller

[2]https://openai.com/index/introducing-gpt-5-4/

[3]https://deepmind.google/models/gemini/pro/

Table 2: Accuracy (%) under each framework.

Framework	Text-only reasoning				Multimodal reasoning			
	AIME25	IMO	CM	Avg.	MV	MVa	OB	Avg.
Qwen3-8B-VL								
Base model	20.0	3.0	52.4	25.1	53.9	59.7	15.8	43.1
AutoGen	20.0	10.2	50.4	26.9	**55.3**	60.3	17.1	44.2
OctoTools	16.7	10.0	52.6	26.4	52.4	61.1	16.9	43.5
MathAgent	20.0	10.7	53.1	27.9	54.2	59.8	**20.5**	44.8
Ours	**24.4**	**12.5**	**54.1**	**30.3**	55.9	**62.2**	19.1	**45.7**
Qwen3-235B								
Base model	50.0	24.1	72.2	48.8	66.5	78.2	34.7	59.8
AutoGen	53.3	25.2	76.4	51.6	67.2	81.0	27.2	58.5
OctoTools	53.3	24.8	75.8	51.3	67.9	73.3	35.2	58.8
MathAgent	50.0	25.0	77.2	50.7	67.3	73.6	35.9	58.9
Ours	**56.7**	**26.3**	**78.9**	**54.0**	**68.5**	**82.3**	**38.0**	**62.9**
GPT-5.4-mini								
Base model	63.3	25.4	76.8	55.2	68.0	70.0	49.2	62.4
AutoGen	60.0	24.3	78.9	54.4	69.5	80.2	55.8	68.5
OctoTools	66.7	25.6	79.1	57.1	68.3	72.2	55.3	65.3
MathAgent	**70.0**	26.1	79.8	58.6	69.2	74.6	56.4	66.7
Ours	**70.0**	**29.1**	**81.2**	**60.1**	**71.2**	**83.5**	**59.4**	**71.4**
Gemini-3-flash								
Base model	60.0	26.5	78.2	54.9	69.9	76.3	53.4	66.5
AutoGen	63.3	27.3	79.3	56.6	70.6	79.2	55.4	68.4
OctoTools	66.7	**28.0**	77.0	57.2	69.5	81.8	53.7	68.3
MathAgent	70.0	27.6	78.5	58.7	**72.0**	83.5	55.2	70.2
Ours	**76.7**	30.4	**85.4**	**64.2**	72.5	**85.4**	**57.9**	**71.9**

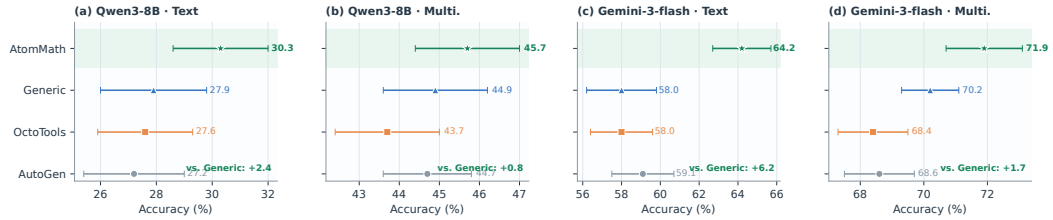

Figure 3: Tool-matched comparison. The four panels show text-only and multimodal results for both backbones.

but remain positive on Gemini-3.1-Pro ($+3.2/+3.1$ for text/multimodal). On stronger models, this improvement suggests relatively diminishing, such as Gemini-3.1-Pro ($+3.2/+3.1$ for text/multimodal), mainly because performance has already reached a certain saturation in easier tasks, while there is still a significant improvement in more difficult problems (like countermath 5.9).

Comparison with other agent frameworks Table 2 compares complete agent systems. ATOMMATH-AGENT achieves the highest text and multimodal averages. Relative to the strongest non-ATOMMATH-AGENT entry in each block, the gains are $+2.4/+0.9$ on Qwen3-8B-VL, $+2.4/+3.1$ on Qwen3-235B, $+1.5/+2.9$ on GPT-5.4-mini, and $+5.5/+1.7$ on Gemini-3-flash.

Tool-matched agent framework comparison. In Figure 3, we further assign every agent the same tool with tool descriptions, token limit, and maximum number of iterations. Under matched tool access, ATOMMATH-AGENT remains stronger than other frameworks on both backbones. The gains are $+2.4/+0.8$ points on Qwen3-8B-VL and $+6.2/+1.7$ on Gemini-3-flash for text/multimodal reasoning with a generic agent, respectively. The larger margins indicate that atomic capability-grounded decomposition contributes beyond merely exposing the agent to mathematical tools. This controlled gap is the key mechanism result that the improvement persists when tool access is held fixed, so we can conclude that this atomic reasoning structure better stimulates our agent framework's performance by how reasoning is planned, composed, and checked.

Table 3: Ablations of reasoning structure, tool access, atomic capability taxonomy, and dynamic replanning.

Reasoning structure	Atomic Capability	Tools	Replan	Qwen3-8B-VL		Gemini-3-flash	
				Text	Multi.	Text	Multi.
Reasoning structure × tool access							
Generic decomposition	None	×	✓	22.5	39.8	55.1	66.8
Generic decomposition	None	✓	✓	26.4	41.3	51.4	69.3
Atomic DAG	Full	×	✓	27.9	42.4	58.9	68.7
Atomic DAG (full)	**Full**	✓	✓	**30.3**	**45.7**	**64.2**	**71.9**
Taxonomy semantics and granularity (atomic DAG; tools on)							
Atomic DAG	Generic labels	✓	✓	25.7	42.9	58.8	67.6
Atomic DAG	Coarse Structure	✓	✓	27.5	43.2	58.5	70.3
Closed-loop control (full taxonomy; tools on)							
Atomic DAG	Full	✓	×	27.1	44.5	57.3	70.0

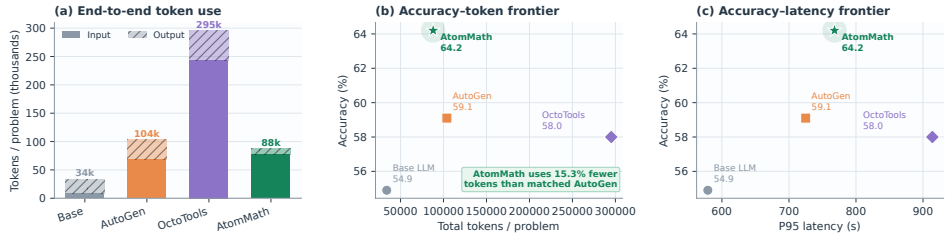

Figure 4: (a) Input/output token composition. (b) Accuracy versus total tokens. (c) Accuracy versus latency.

4.3 Ablation Study

We conduct ablation experiments to verify the effectiveness of the framework design. In Table 3, we compare different atomic planner structures, tool activation status, and dynamic next-step planning designs. With tools enabled, replacing generic decomposition with the atomic DAG improves Qwen3-8B-VL by $+3.9/+4.4$ and Gemini-3-flash by $+12.8/+2.6$. Atomic decomposition also helps without tools, showing its contribution. In Tools alone, they improve the generic agent on three settings but reduce Gemini text performance, whereas the atomic representation enables positive tool gains. The atomic capability taxonomy controls further identify the source of this advantage. Relative to generic labels (only provide atomic capability labels), the full capability taxonomy yields $+4.6/+2.8$ points on Qwen3-8B-VL and $+5.4/+4.3$ on Gemini-3-flash. Coarsening the capability structure also degrades performance, indicating that both capability semantics and granularity matter. Finally, disabling dynamic replanning reduces text accuracy by 3.2 and 6.9. Together, the ablations identify the sources of gain, where atomic structure organizes the problem solving, capability specializes each unit's execution and checks, and replanning prevents local failures.

5 Discussion

Efficiency Analysis Figure 4 summarizes the end-to-end efficiency comparison on Gemini-3-flash. AtomMath-Agent reaches 64.2% accuracy, outperforming AutoGen and OctoTools by 5.1 and 6.2 points while using 15.3% and 70.2% fewer total tokens. Its short output traces indicate that most context is used to maintain planning, evidence, and verification rather than to lengthen the final response. AtomMath-Agent reduces P95 latency from 914 to 768 seconds relative to OctoTools, but remains more expensive than the unwrapped backbone. Thus, the main efficiency benefit is a better accuracy–token frontier, not universally lower latency.

Effectiveness for Agentic Training We use AtomMath-Agent to synthesize structured mathematical agent trajectories for agentic mid-training. The construction, training details, and evaluation results are in Appendix D.3. In Figure 5(a), structured atomic trajectories improve the six-task average from 52.48 with generic trajectories to 56.11. The largest gains occur in math planning ($+10.78$) and math feedback ($+9.77$), directly matching the typed plans and verification signals

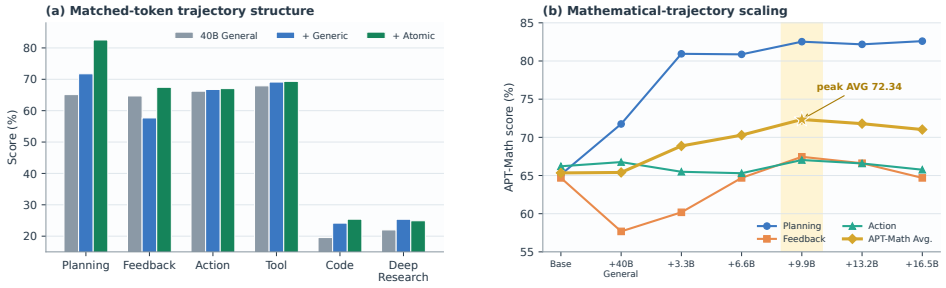

Figure 5: Agentic mid-training analysis. (a) Comparison between general data, generic trajectories, and structured atomic math trajectories. (b) Performance as the amount of mathematical trajectory data increases.

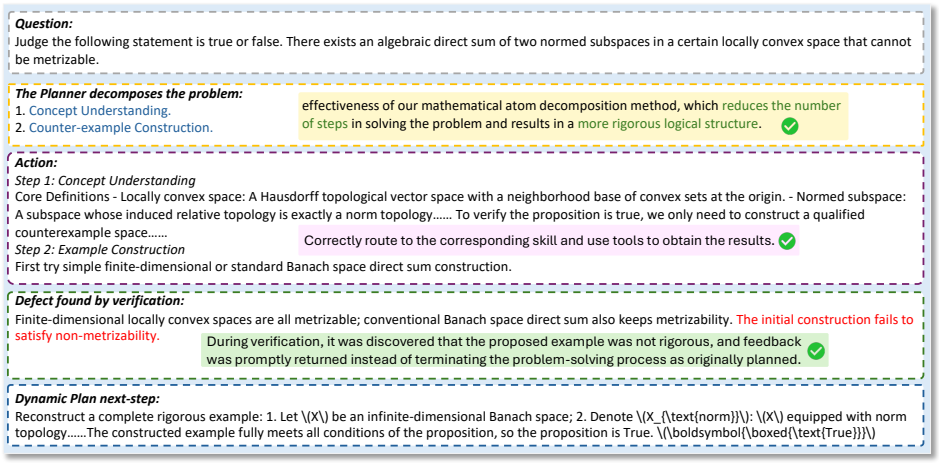

Figure 6: Representative trajectory of ATOMMATH-AGENT.

retained in the data. Beyond mathematics, this trajectory training also improves the fields of tool, code, and deep research, proving that structured atomic reasoning trajectories do not carry a single mathematical skill, but a unified agent reasoning paradigm. The scaling study in Figure 5(b) provides a complementary result. 40B general data alone leaves the APT-Math average nearly unchanged, whereas adding mathematical trajectories raises it to 68.87 at 3.3B tokens and peaks at 72.34 at 9.9B tokens, showing a useful scaling regime rather than monotonic scaling.

Case study Figure 6 shows a representative case. The agent decomposes a research-level proof into capability-typed atomic units, routes each unit through its execution policy, and revises the unresolved graph when verification detects a flaw. The case illustrates the intended advantage that an error is attached to a specific unit and contract, so correction can remain local instead of restarting an undifferentiated reasoning chain. More complete trajectories are provided in Appendix D.4.

6 CONCLUSION

We present ATOMMATH-AGENT, a Unified Math Agent to reshape the reasoning by capability-grounded atomic reasoning units. By binding one mathematical capability to its atomic goal, dependencies, schema, and validation, the unit becomes the interface for planning, execution, feedback, and memory. This design shifts the specialization from tools to the organization of reasoning, where the planner decides atomic capabilities and selects execution atomic skills, the executor realizes those policies, and capability-aware feedback supports planning. Results across backbones and ablations show that the gains are explained by atomic structure, capability semantics, and granularity. The agentic training further benefits planning and feedback, and other agentic areas during agentic mid-training. The results support capability-grounded atomic reasoning as a unified design principle, while providing insights into how the atomic capability taxonomy can be learned in future agents.

AI USE STATEMENT

LLM-based AI assistants are used for limited writing support, including language polishing, word choice correction, grammar checking, and chart creation. They are not used to originate scientific claims, experimental results, annotations, or citations. The authors verify all technical content, experiments, and manuscript decisions and take full responsibility for the paper.

REFERENCES

Lingxiao Du, Fanqing Meng, Zongkai Liu, Zhixiang Zhou, Ping Luo, Qiaosheng Zhang, and Wenqi Shao. Mm-prm: Enhancing multimodal mathematical reasoning with scalable step-level supervision. *arXiv preprint arXiv:2505.13427*, 2025.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yujiu Yang, Minlie Huang, Nan Duan, Weizhu Chen, et al. Tora: A tool-integrated reasoning agent for mathematical problem solving. In *The Twelfth International Conference on Learning Representations*.

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

Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu, Zhen Thai, Junhao Shen, Jinyi Hu, Xu Han, Yujie Huang, Yuxiang Zhang, et al. Olympiadbench: A challenging benchmark for promoting agi with olympiad-level bilingual multimodal scientific problems. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3828–3850, 2024.

Jiayi Kuang, Haojing Huang, Yinghui Li, Xinnian Liang, Zhikun Xu, Yangning Li, Xiaoyu Tan, Chao Qu, Meishan Zhang, Ying Shen, and Philip S. Yu. Atomic thinking of llms: Decoupling and exploring mathematical reasoning abilities, 2025a. URL https://arxiv.org/abs/2509.25725.

Jiayi Kuang, Ying Shen, Jingyou Xie, Haohao Luo, Zhe Xu, Ronghao Li, Yinghui Li, Xianfeng Cheng, Xika Lin, and Yu Han. Natural language understanding and inference with mllm in visual question answering: A survey. *ACM Computing Surveys*, 57(8):1–36, 2025b.

Jia Li, Edward Beeching, Lewis Tunstall, Ben Lipkin, Roman Soletskyi, Shengyi Huang, Kashif Rasul, Longhui Yu, Albert Q Jiang, Ziju Shen, et al. Numinamath: The largest public dataset in ai4maths with 860k pairs of competition math problems and solutions. *Hugging Face repository*, 13(9):9, 2024.

Yinghui Li, Jiayi Kuang, Haojing Huang, Zhikun Xu, Xinnian Liang, Yi Yu, Wenlian Lu, Yangning Li, Xiaoyu Tan, Chao Qu, et al. One example shown, many concepts known! counterexample-driven conceptual reasoning in mathematical llms. In *Forty-second International Conference on Machine Learning*.

Haoran Liao, Qinyi Du, Shaohua Hu, Hao He, Yanyan Xu, Jidong Tian, and Yaohui Jin. Modeling complex mathematical reasoning via large language model based mathagent. In *Proceedings of ICLR*, 2024. URL https://openreview.net/forum?id=Ge7ZqrKG9t.

Sam Lightman, Aitor Lewkowycz, Alex Slone, et al. Let's verify step by step: PRM800K, a process supervision dataset for math reasoning. *arXiv preprint arXiv:2305.20050*, 2023. URL https://arxiv.org/abs/2305.20050.

Haohan Lin, Zhiqing Sun, Yiming Yang, and Sean Welleck. Lean-star: Learning to interleave thinking and proving. *arXiv preprint arXiv:2407.10040*, 2024.

Fan Liu, Zhe-Rui Yang, Cancheng Liu, Tianrui SONG, Xiaofeng Gao, and Hao Liu. Mm-agent: Llm as agents for real-world mathematical modeling problem. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

Junru Lu, Jiarui Qin, Lingfeng Qiao, Yinghui Li, Xinyi Dai, Bo Ke, Jianfeng He, Ruizhi Qiao, Di Yin, Xing Sun, et al. Youtu-llm: Unlocking the native agentic potential for lightweight large language models. *arXiv preprint arXiv:2512.24618*, 2025a.

Pan Lu, Hritik Bansal, Tony Xia, Jiacheng Liu, Chunyuan Li, Hannaneh Hajishirzi, Hao Cheng, Kai-Wei Chang, Michel Galley, and Jianfeng Gao. Mathvista: Evaluating mathematical reasoning of foundation models in visual contexts. *arXiv preprint arXiv:2310.02255*, 2023.

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*, 2025b.

Yunsheng Ma, Beidi Chen, Fuchun Yang, et al. What are step-level reward models? a framework for structured reasoning supervision. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2025. URL https://ojs.aaai.org/index.php/AAAI/article/view/34663.

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

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652, 2023.

Zayne Rea Sprague, Fangcong Yin, Juan Diego Rodriguez, Dongwei Jiang, Manya Wadhwa, Prasann Singhal, Xinyu Zhao, Xi Ye, Kyle Mahowald, and Greg Durrett. To cot or not to cot? chain-of-thought helps mainly on math and symbolic reasoning. In *The Thirteenth International Conference on Learning Representations*.

Ke Wang, Junting Pan, Weikang Shi, Zimu Lu, Houxing Ren, Aojun Zhou, Mingjie Zhan, and Hongsheng Li. Measuring multimodal mathematical reasoning with math-vision dataset. *Advances in Neural Information Processing Systems*, 37:95095–95169, 2024a.

Peiyi Wang, Lei Li, Zhihong Shao, Runxin Xu, Damai Dai, Yifei Li, Deli Chen, Yixuan Wu, and Zhifang Sui. Math-shepherd: Verify and reinforce LLMs step-by-step without human annotations. In *Proceedings of ACL*, 2024b. URL https://aclanthology.org/2024.acl-long.510/.

Peng-Yuan Wang, Tian-Shuo Liu, Chenyang Wang, Ziniu Li, Yidi Wang, Shu Yan, Chengxing Jia, Xu-Hui Liu, Xinwei Chen, Jiacheng Xu, et al. A survey on large language models for mathematical reasoning. *ACM Computing Surveys*, 58(8):1–35, 2026.

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*, 2024.

Yiran Wu, Feiran Jia, Shaokun Zhang, Hangyu Li, Erkang Zhu, Yue Wang, Yin Tat Lee, Richard Peng, Qingyun Wu, and Chi Wang. Mathchat: Converse to tackle challenging math problems with LLM agents. *arXiv preprint arXiv:2306.01337*, 2023. URL https://arxiv.org/abs/2306.01337.

Wei Xiong, Chengshuai Shi, Jiaming Shen, Aviv Rosenberg, Zhen Qin, Daniele Calandriello, Misha Khalman, Rishabh Joshi, Bilal Piot, Mohammad Saleh, et al. Building math agents with multi-turn iterative preference learning. In *13th International Conference on Learning Representations, ICLR 2025*, pp. 92319–92350. International Conference on Learning Representations, ICLR, 2025.

Yibo Yan, Shen Wang, Jiahao Huo, Philip S Yu, Xuming Hu, and Qingsong Wen. Mathagent: Leveraging a mixture-of-math-agent framework for real-world multimodal mathematical error detection. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 6: Industry Track)*, pp. 69–82, 2025.

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*, 2024.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Zhengyuan Yang, Linjie Li, Jianfeng Wang, Kevin Lin, Ehsan Azarnasab, Faisal Ahmed, Zicheng Liu, Ce Liu, Michael Zeng, and Lijuan Wang. Mm-react: Prompting chatgpt for multimodal reasoning and action. *arXiv preprint arXiv:2303.11381*, 2023.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822, 2023.

Jianghangfan Zhang, Yibo Yan, Kening Zheng, Xin Zou, Song Dai, and Xuming Hu. Gm-prm: A generative multimodal process reward model for multimodal mathematical reasoning. *arXiv preprint arXiv:2508.04088*, 2025a.

Shuo Zhang, Bowen Li, Tianyu Sun, et al. Multi-agent debate for fine-grained reasoning in math. In *Findings of ACL*, 2025b. URL https://aclanthology.org/2025.findings-acl.862/.

Yue Zhang, Jiaxin Zhang, Qiuyu Ren, Tahsin Saffat, Xiaoxuan Liu, Zitong Yang, Banghua Zhu, and Yi Ma. GAUSS: Benchmarking structured mathematical skills for large language models. *arXiv preprint arXiv:2509.18122*, 2025c. URL https://arxiv.org/abs/2509.18122.

Zhen Zhang, Huajie Chen, et al. The lessons of developing process reward models in mathematical reasoning. *arXiv preprint arXiv:2501.07301*, 2025d. URL https://arxiv.org/abs/2501.07301.

Zhenru Zhang, Chujie Zheng, Yangzhen Wu, Beichen Zhang, Runji Lin, Bowen Yu, Dayiheng Liu, Jingren Zhou, and Junyang Lin. The lessons of developing process reward models in mathematical reasoning. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 10495–10516, 2025e.

Tianyang Zhong, Zhengliang Liu, Yi Pan, Yutong Zhang, Zeyu Zhang, Yifan Zhou, Shizhe Liang, Zihao Wu, Yanjun Lyu, Peng Shu, et al. Evaluation of openai o1: Opportunities and challenges of agi. *arXiv preprint arXiv:2409.18486*, 2024.

Kaiwen Zhou, Xuandong Zhao, Jayanth Srinivasa, Gaowen Liu, Aosong Feng, Dawn Song, and Xin Eric Wang. Safekey: Amplifying aha-moment insights for safety reasoning. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 25407–25423, 2025.

A Atomic Capability Taxonomy

Our taxonomy is organized by *reasoning function*, not by mathematical subject area. Subject labels such as algebra or geometry describe where a problem comes from, whereas the agent must decide what operation to perform at each step: recognize a symbol, formalize a statement, calculate an expression, construct a proof, or inspect a failure. We synthesized candidate capabilities from mathematical reasoning taxonomies, formal reasoning and modeling tasks, and prior work on atomic mathematical skills Lin et al. (2024); Liu et al.; Kuang et al. (2025a); Zhang et al. (2025c); Lu et al. (2025a).

Three researchers with PhDs in mathematics-related fields and two senior mathematics instructors reviewed the definitions and representative tasks. We iteratively refined the inventory for broad operational coverage, limited overlap, evaluability, and alignment with agent control. These criteria motivate a practical routing taxonomy; they do not establish that the inventory is theoretically complete or uniquely minimal. Table 4 reports each capability's primary role. Action capabilities produce or transform mathematical evidence, while feedback capabilities assess, revise, or extend the current solution state.

Table 4: Operational capability taxonomy used by ATOMMATH-AGENT. Colored bands group related reasoning functions; the role column indicates the capability's primary position in the control loop.

Atomic capability	Operational definition	Role
Perception, knowledge, and computation		
Symbol Recognition	Parse mathematical symbols and structural patterns from textual or visual inputs.	Action
Concept Understanding	Identify, distinguish, and relate mathematical concepts and their applicability conditions.	Action
Calculation	Perform arithmetic, algebraic, calculus, linear-algebra, and symbolic operations.	Action
Structured reasoning and application		
Spatial Perception	Infer geometric objects, spatial relations, and quantitative measurements from diagrams or descriptions.	Action
Formal Mathematical Language	Translate natural-language statements into machine-readable mathematical representations.	Action
Deduction and Induction Reasoning	Derive consequences from premises or infer reusable patterns from instances.	Action
Counter-proof and Construction	Construct counterexamples, contradictions, auxiliary objects, or proof witnesses.	Action
Modeling Transformation	Map textual or real-world scenarios to variables, constraints, objectives, and equations.	Action
Feedback and knowledge extension		
Theorem Application	Retrieve an applicable theorem, check its preconditions, and assess whether its use is valid.	Feedback
Self-Reflection	Diagnose trajectory errors and propose a correction or alternative reasoning strategy.	Feedback
New Knowledge Acquisition	Acquire an unfamiliar definition or theorem and transfer it to the current problem.	Feedback

B CAPABILITY-GROUNDED CONTROL AND PROMPT SCHEMAS

ATOMMATH-AGENT follows the Planner–Executor–Feedback–Memory loop illustrated in Figure 7. The solver first decomposes the input into capability-typed atomic units and then selects the next dependency-ready unit. The Executor receives a capability and its contract, loads the corresponding skill policy, and only then chooses a concrete resource or executes without an external tool. The result and execution metadata are stored in Memory, after which the verifier emits CONTINUE, RETRY, or STOP. Planning therefore specifies what mathematical operation is required, whereas resource selection remains an Executor responsibility.

B.1 ATOMIC UNIT CONSTRUCTION AND PLANNER CONTROL

The Planner contains two levels of control. The first level is `Global decomposition`, which turns a question into dependency-aware atomic units and a JSON plan with step identifiers, capability assignments, dependency links, output schemas, and validation contracts. The second level is `Dynamic review and next-step planning`, which re-reads Memory, constructs an enriched dependency digest, and chooses one of four actions: KEEP, MODIFY, REPLAN, or RETRY. The Planner may revise what operation is needed, but concrete resource selection is deferred to the Executor.

Global Planner construct capability-typed atomic units

ROLE Build the smallest dependency-aware plan sufficient to solve the problem. Do not select concrete tools.

INPUT Problem: `{question}` Visual context: `{image_info}`
Available capabilities: `{capability_metadata}`

ATOMICITY RULES Each unit must have one primary subgoal and one capability, depend only on named predecessor units, and expose an independently checkable output. Merge units only when all three

conditions remain true.

CANONICAL OUTPUT SCHEMA

```
{{"problem_analysis": "...",
"atomic_units": [{{
"unit_id": "u1",
"subgoal": "single mathematical objective",
"capability_id": "one inventory identifier",
"dependency_ids": ["verified predecessor ids"],
"output_schema": {{"type": "...", "required_fields": [...]}},
"validation_contract": {{
"operational_checks": [...],
"local_checks": [...],
"global_checks": [...]
}}
}}]}}
```

CONSTRAINTS Return valid JSON only. Use capability identifiers exactly as provided. The graph must be acyclic, and every dependency must reference an available predecessor.

Planner Controller review one atomic unit

INPUT Problem: `{question}`
Current unit: `{current_atomic_unit}`
Verified dependency digest: `{dependency_digest}`
Latest feedback: `{feedback_record}`

CONTROL ACTIONS KEEP: execute the unit unchanged. MODIFY: minimally revise its subgoal, capability, dependencies, schema, or contract.
RETRY: re-execute the failed unit using its failure metadata. REPLAN: rebuild only the unresolved subgraph.

OUTPUT Return JSON with `control_action`, `reason`, and an optional `revised_unit` in the canonical atomic-unit schema. Never select a concrete tool.

Dynamic Planner instantiate the next dependency-ready unit

INPUT Unresolved graph: `{remaining_graph}`
Verified units: `{verified_memory}` Budget: `{remaining_budget}`
Available capabilities: `{capability_metadata}`

TASK Select a unit whose dependencies are satisfied. If no planned unit remains valid, return REPLAN; otherwise instantiate exactly one unit using the canonical schema.

OUTPUT `unit_id`, `subgoal`, `capability_id`, `dependency_ids`, `output_schema`, and `validation_contract`. Include only verified predecessor evidence; resource selection is deferred to the Executor.

This design exposes two important implementation choices. First, the Planner explicitly encodes data dependencies between steps instead of assuming a purely linear chain, which makes it possible to condition each step on a structured digest of previous results. Second, the Planner already contains step-minimization rules in the prompt itself, which biases it toward short, capability-aligned plans rather than verbose reasoning traces.

B.2 CONTRACT-AWARE FEEDBACK AND STATE TRANSITIONS

The Feedback module is implemented by `verificate_context`. After every execution, it receives the atomic unit and its validation contract together with the original question, image metadata, initial analysis, and a serialization of Memory. It jointly evaluates operational success, local contract satisfaction, and global trajectory completion, then aggregates these checks into one of three conclusions: RETRY, CONTINUE, or STOP. These are three validation roles within one feedback procedure, not separately trained verifier models. A RETRY record contains the failure status and reason so that the Planner can revise or re-execute the unit.

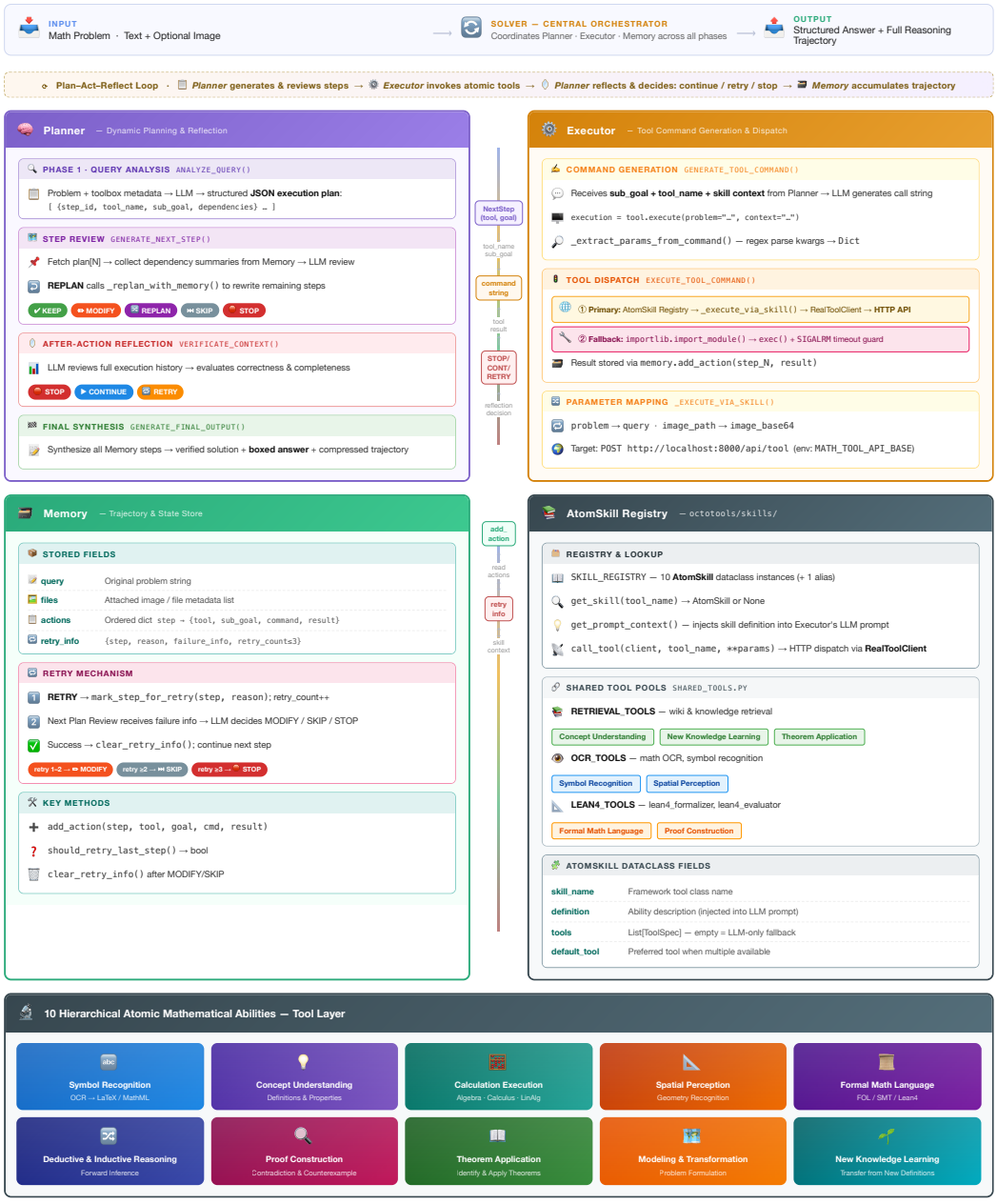

Figure 7: Implementation workflow of ATOMMATH-AGENT. The canonical atomic-unit record is created by the Planner, instantiated by an Atom Skill and optional resource, checked against its validation contract, and committed to Memory before the next state transition.

Contract-Aware Verifier check one unit and the global state

INPUT Problem: `{question}`
Current unit: `{current_atomic_unit}`
Validation contract: `{validation_contract}`
Execution result: `{execution_result}`
Verified memory and unresolved graph: `{memory_and_graph}`

CHECKS **Operational**: did the resource execute and return a parseable object?
Local: does the result satisfy the output schema, mathematical constraints, and dependency values?
Global: is the original problem solved, and if not, which goals remain unresolved?

DECISION RULE Return RETRY when operational or local checks fail; STOP only when the global objective is supported by verified evidence; otherwise return CONTINUE. Do not stop or continue solely from the number of executed steps.

Verifier Output structured feedback record

```
{{"operational": {{"pass": true, "evidence": "..."}},
"local": {{"pass": true, "violations": []}},
"global": {{"complete": false, "missing_goals": [...]}},
"decision": "RETRY | CONTINUE | STOP",
"retry_reason": null,
"released_successors": ["unit ids"]}}
```

Unlike a generic self-reflection prompt, the verifier consumes the same atomic-unit schema used by the Planner. Its output records which contract failed and which successor units become dependency-ready, making feedback an explicit state transition rather than an unstructured critique.

C EXECUTION POLICIES AND EXTERNAL RESOURCES

The Atom Skill Library operationalizes capability-conditioned execution. Each skill has two synchronized representations: a Python dataclass used by the runtime router and a Markdown skill card used as Executor context. Together they specify the capability definition, applicability conditions, admissible resources, input/output schemas, timeout constraints, and examples. The registry receives a capability identifier from the Planner, loads the corresponding policy, and exposes candidate resources to the Executor. It does not determine the decomposition or require a one-to-one mapping between capabilities and tools.

C.1 ATOM SKILLS AS EXECUTION POLICIES

A capability may admit several resources, and one resource may support multiple capabilities. The Executor therefore selects a resource from the current unit's subgoal, dependency evidence, output schema, and validation contract rather than from a fixed one-to-one mapping. Table 5 summarizes representative policies and candidates.

Atom Skill schema capability-conditioned execution policy

```
CALCULATION_POLICY = AtomSkill(
capability_id="Calculation",
applicability="exact or numerical mathematical computation",
output_schema=NumericOrSymbolicResult,
candidate_resources=[
ToolSpec(id="math_calculatorv2", params=["query"]),
ToolSpec(id="math_calculator", params=["query"])
]
)
```

Executor contract. The router receives `capability_id` from the atomic unit, loads the policy, checks applicability and I/O constraints, and then selects a resource. The policy never changes the unit's mathematical objective.

Example policy card Calculation

Capability and applicability. Perform arithmetic, algebra, calculus, linear algebra, number theory, combinatorics, statistics, or symbolic manipulation. Prefer exact symbolic results when the unit contract requires exactness.

Candidate resources. `math_calculatorv2` for exact symbolic operations; `math_calculator` for fast numerical evaluation; `math_calculator_wa` for non-standard queries. The Executor chooses among them from the subgoal, output schema, and validation contract.

Unified request / response.

```
POST /api/resource
{ "resource_id": "math_calculatorv2",
"unit_id": "u3", "query": "<expression>" }
{ "status": "success", "result": "<exact result>",
"evidence": "<execution metadata>" }
```

Table 5: Representative Atom Skills and candidate resources. A skill is an execution policy associated with a capability; resources are selected at runtime and are not in one-to-one correspondence with skills.

Capability-conditioned policy	Candidate resources	Execution specialization
Perception and knowledge		
Symbol_Recognition	math_ocr_hy	Parse formula images and normalize symbols into structured text.
Concept_Understanding	knowledge_retrieval / backbone	Retrieve or explain definitions and applicability conditions.
Spatial_Perception	OCR / geometry resources	Align visual objects with symbolic quantities and spatial relations.
New_Knowledge_Learn	retrieval resources	Acquire an unfamiliar definition or theorem for the current context.
Computation and modeling		
Calculation_Execution	calculatorv2 / calculator / WA	Select exact symbolic, fast numeric, or non-standard computation.
Modeling_Transformation	math_modeling / backbone	Produce variables, constraints, objectives, and equations.
Formal and proof-oriented reasoning		
Formal_Math_Language	lean4_formalizer	Translate statements into machine-checkable formal syntax.
Deductive_Inductive	backbone / math_proverv3	Carry out forward deduction or infer a reusable pattern.
Proof_Construction	backbone / math_proverv3	Construct a contradiction, counterexample, witness, or proof object.
Theorem_Application	retrieval / proof resources	Retrieve a theorem, verify its preconditions, and apply it.

The dual-format design is useful in practice. The Python dataclass gives the Executor a strict runtime schema, while the Markdown skill cards provide human-readable documentation and richer prompt material. This separation keeps tool routing explicit without hard-coding all reasoning into a single monolithic system prompt.

Table 6: Representative external resources. Capability policies may share resources, and the Executor selects a resource only after the Planner has fixed the atomic unit.

Resource	Capability policies	Purpose
Perception and retrieval		
math_ocr_hy	Symbol Recognition; Spatial Perception	Formula OCR, symbol normalization, and extraction of visual relations.
knowledge_retrieval	Concept; Theorem; New Knowledge	Retrieve definitions, theorem statements, applicability conditions, and background knowledge.
Computation and modeling		
math_calculatorv2	Calculation	Exact SymPy-based computation for equations, integrals, matrices, and symbolic expressions.
math_calculator	Calculation	Fast numerical evaluation through code execution.
math_calculator_wa	Calculation	Non-standard mathematical queries through a Wolfram-style backend.
math_modeling	Modeling Transformation	Generate variables, constraints, objectives, and structured equations.
Formalization and proof		
lean4_formalizer	Formal Mathematical Language	Translate natural-language statements into Lean 4 syntax.
lean4_evaluator	Formal Language; Proof Construction	Check whether generated Lean 4 code is accepted.
math_proverv3	Proof Construction	Formalize, generate, and verify an end-to-end proof.
math_proverv2	Proof Construction	Combine natural-language and formal-statement theorem proving.

C.2 Resource Families and Unified Interfaces

We integrate representative resources for computation, proof, retrieval, OCR, modeling, and visualization, as summarized in Table 6. Because these resources have heterogeneous environments and dependencies, each external resource is isolated behind a common HTTP-style interface. This normalization is an engineering choice that lets the Executor apply a shared invocation protocol; it is separate from the capability taxonomy and atomic decomposition mechanism.

D Experimental Protocol and Supplementary Results

D.1 Benchmarks and Evaluation Metrics

We evaluate AtomMath-Agent on both text-only and multimodal mathematical benchmarks, each of which stresses a different aspect of agentic reasoning. AIME25 focuses on short-answer competition mathematics with exact numeric answers; it is the cleanest setting for testing decomposition and calculation. CounterMath is a university-level and research-level mathematical proof benchmark that focuses on counterexamples in mathematics, using examples and proof by contradiction to prove propositions. IMOBench contains Olympiad-style advanced reasoning problems whose answers are often symbolic or LaTeX-formatted, making surface-form matching unreliable. MathVision and MathVista test visual-symbolic alignment, while OlympiadBench-Math mixes long-form mathematical reasoning with multimodal context and, in some cases, answer tolerances.

The evaluation script first extracts the final boxed answer from the model response. For pure numeric tasks, it applies exact matching or tolerance-aware numeric comparison. For CounterMath, it converts both prediction and reference into a binary true/false label and additionally reports F1. For symbolic benchmarks such as IMOBench, the script falls back to an LLM judge when normalization and numeric parsing are insufficient to decide mathematical equivalence.

D.2 Implementation and Evaluation Details

Inference protocol. We use the official API for each proprietary backbone and the same released checkpoint for each open-source comparison. The maximum output length is 20,000 tokens and

Table 7: Benchmark coverage and evaluation protocol. LLM judging is used only when deterministic normalization cannot establish symbolic equivalence.

Benchmark	Primary stress	Answer form	Evaluation
Text-only reasoning			
AIME25	Closed-form competition solving	Integer or boxed numeric answer	Exact or tolerance-aware numeric matching.
CounterMath	Proof versus counterexample judgment	True/False label with rationale	Automatic label extraction; accuracy and F1.
IMOBench	Olympiad proof and symbolic reasoning	LaTeX or symbolic expression	Boxed-answer extraction; LLM equivalence fallback.
Multimodal reasoning			
MathVision	Visual–symbolic alignment and diagrams	Numeric, choice, or text	Numeric/choice matching; LLM fallback when needed.
MathVista	Chart and diagram understanding	Numeric, choice, or text	Automatic numeric or exact matching.
OlympiadBench-Math	Long-horizon Olympiad reasoning with visual context	Numeric or symbolic expression	Tolerance-aware matching; LLM fallback when needed.

the maximum agent horizon is 10 iterations. Unless a model requires a provider-specific setting, temperature and top-k use the corresponding default values and remain fixed across methods using that backbone. Every reported inference result is the arithmetic mean of three independent runs over the same benchmark instances and ordering. Text-only and multimodal macro averages are computed by first evaluating each benchmark and then averaging the three benchmark scores, so datasets of different sizes receive equal weight.

Native and controlled comparisons. Table 2 follows each framework's native orchestration policy and therefore measures end-to-end system quality. For Figure 3, all agents receive identical tool APIs and descriptions, a 20,000-token output limit, and at most 10 iterations. The Generic Agent retains dependency-aware decomposition, memory, verification, and replanning, but removes the mathematical capability type from each node and uses generic execution instructions. In the 2×2 ablation, Tools Off prohibits all external calls, while Tools On exposes the same tool set to both decomposition strategies. Taxonomy variants keep the graph, tools, verifier, and budgets fixed and alter only the capability labels and granularity.

Uncertainty reporting. The 95% confidence intervals in Figure 3 are computed from the three run-level macro averages using the Student-t interval $\bar{x} \pm t_{0.975,2}s/\sqrt{3}$. Tables 1, 2, and 3 report the corresponding three-run means. We avoid significance claims when paired per-instance predictions are unavailable.

D.3 Agentic Mid-Training and Trajectory Data

We train a Dense MLA-style long-context model from scratch rather than using an opaque post-trained checkpoint. The base model is pre-trained on approximately 8T general tokens spanning web, encyclopedic, STEM, and code corpora. Agentic mid-training starts from the same base checkpoint and uses the same optimizer, sequence length, batch construction, and evaluation protocol across conditions. Every reported training result is the mean of three independent runs with different data orders.

For the matched-token comparison, both trajectory conditions contain 9.9B mathematical tokens drawn from the same source problems and filtered by the same final-answer correctness rule. Generic trajectories retain untyped plan–action–feedback text, whereas structured atomic trajectories preserve the canonical unit fields—subgoal, capability, dependencies, output schema, and validation contract—together with selected resources, validation outcomes, and retry/replanning signals. This construction controls mathematical content and token volume while changing the trajectory representation. For the scaling experiment, we keep the 40B general corpus and all optimization settings fixed while varying only the number of structured mathematical-trajectory tokens from 3.3B to 16.5B.

Table 8: Matched-token trajectory controls on APT-Bench, averaged over three training runs. The Overall column averages all six reported dimensions.

Mid-training data	Math tokens	Planning	Feedback	Action	Tool	Code	Deep Research	Overall
40B General	0	65.12	64.69	66.21	67.91	19.56	21.99	50.91
+ Generic trajectories	9.9B	71.75	57.68	66.76	69.12	24.16	**25.39**	52.48
+ Structured atomic trajectories	**9.9B**	**82.53**	**67.45**	**67.03**	**69.34**	**25.41**	24.91	**56.11**

Table 9: Scaling structured mathematical trajectories during mid-training. All values are three-run means on APT-Math.

Training condition	Math tokens	Planning	Feedback	Action	APT-Math Avg.
Baseline	0	65.12	64.69	66.21	65.34
+40B General	0	71.75	57.68	66.76	65.40
+40B General + Math	3.3B	80.94	60.18	65.49	68.87
+40B General + Math	6.6B	80.87	64.69	65.30	70.29
+40B General + Math	**9.9B**	**82.53**	**67.45**	**67.03**	**72.34**
+40B General + Math	13.2B	82.18	66.61	66.58	71.79
+40B General + Math	16.5B	82.60	64.69	65.76	71.02

The retained mathematical trajectories contain the original question, the global execution plan, capability assignments, tool calls and outputs, validation results, retry/replanning events, and the final response. The scaling curve is non-monotonic: 9.9B mathematical tokens yield the best average, so we use that amount for the matched-token study rather than assuming that more trajectory data is always beneficial.

Trajectory construction and retention pipeline

1. Sample. Draw competitive and research-level mathematical problems.
2. Execute. Run AtomMath-Agent and retain the canonical atomic units, control actions, resource calls, evidence, and feedback records.
3. Filter. Keep only trajectories whose final answer passes the task evaluator; apply the same source-problem and correctness filters to matched Generic trajectories.
4. Match. Hold mathematical token count, base checkpoint, and optimization recipe fixed when comparing trajectory representations.
5. Evaluate. Measure APT-Math and transfer to planning, feedback, tool use, code, and Deep Research.

D.4 Trajectory Case Studies

Beyond the examples in the main text, Figure 8 provides a fuller visual summary of the trajectory analysis.

E Limitations, and Broader Impact

E.1 Limitations

AtomMath-Agent depends on a manually curated capability taxonomy and on contracts that can be expressed and checked by the current verifier. The taxonomy has broad operational coverage on the evaluated tasks, but it is neither proven complete nor automatically adapted to unseen capabilities. Likewise, local validation is strongest when outputs have explicit symbolic, numerical, or formal constraints; open-ended proof quality may still require model-based judgment. Tool availability, network latency, and context growth can affect efficiency, and the current experiments do not include every possible taxonomy randomization or difficulty-stratified strong-model control. Future work should learn capability boundaries and validation contracts from data, improve memory compression, and study automatic expansion without collapsing the interpretability of the atomic interface.

Input query

Determine the following values from a standard radio dial. \n\nSubproblem 0: What is the minimum wavelength in m for broadcasts on the AM band? Format your answer as an integer. \nSubproblem 1: What is the maximum wavelength in m for broadcasts on the AM band? Format your answer as an integer.

Step 1: Planning

"step_id": 1,
"tool_name": "Concept_Understanding_Tool",
"task": "Identify the correct frequency range for the AM band and understand the formula for wavelength calculation.",
"input": {
"primary_focus": "Understand the formula and frequency range",
"context_summary": "The formula for wavelength is λ = c / v, where c is the speed of light and v is the frequency. The AM band frequency range is needed.",
"previous_results": "None",
"original_question": "What is the maximum wavelength in m for broadcasts on the AM band?"
},
"dependencies": [],
"context": "The AM band frequency range is typically 530 kHz to 1600 kHz.",
"verification_strategy": {"checkpoints": ["Verify the AM band frequency range is correctly identified","Ensure the formula λ = c / v is correctly understood"],"final_validation": "Cross-check with known values or references",

Step 1: Action

"Action Step 1": {
"tool_name": "Concept_Understanding_Tool",
"sub_goal": "TASK: Identify the correct frequency range for the AM band and understand the formula for wavelength calculation.\nEXPECTED_OUTCOME: Understanding that the maximum wavelength corresponds to the minimum frequency in the AM band (530 kHz).\nDEPENDENCIES: []\nPREVIOUS_RESULTS (planned): None",
“Atom_tool”: “math_knowledge_retrieval”,
“query”: “Understand the formula and frequency range”,
"result": [
{
"concept": "Maximum Wavelength in AM Band Broadcasts. The maximum wavelength (λ) in meters for broadcasts on the AM band is calculated using the formula λ = c / v, where c is the speed of light (3 × 10^8 m/s) and v is the minimum frequency in the AM band (530 kHz). The wavelength is the distance between consecutive peaks of a wave. In the AM band, the maximum wavelength corresponds to the lowest frequency because wavelength and frequency are inversely related. Lower frequencies mean longer wavelengths. key_properties": ["Inverse relationship between wavelength and frequency","Dependent on the speed of light in a vacuum","Determined by the minimum frequency in the AM band"],
"examples": [
"Calculating the maximum wavelength for 530 kHz: λ = (3 × 10^8 m/s) / (530 × 10^3 Hz) ≈ 566 meters",
"Calculating the wavelength for 1600 kHz: λ = (3 × 10^8 m/s) / (1600 × 10^3 Hz) ≈ 187.5 meters"
],
"common_misconceptions": [
"Assuming higher frequencies have longer wavelengths",
"Ignoring the inverse relationship between wavelength and frequency",
"Confusing wavelength with wave amplitude"
]
}

Step 1: Verify and Feedback

Task: Verify the completeness and accuracy of the answer to the question.\n\nQuestion: Preamble: Determine the following values from a standard radio dial. \n\nSubproblem 0: What is the minimum wavelength in m for broadcasts on the AM band? ……"response": "Analysis: The execution log shows that more steps have been completed than initially planned, and the steps do not directly address the subproblem of finding the maximum wavelength for broadcasts on the AM band. The initial analysis and execution plan were not followed correctly, leading to irrelevant calculations and results. The final answer for the maximum wavelength has not been obtained, and the steps executed do not form a coherent solution to the problem.\n\nSignal: false\nConclusion: CONTINUE

Figure 8: Visual summary of the trajectory case study; the continuation appears in Figure 9.

E.2 Impact Statement

Our work may improve access to mathematical reasoning support in education, scientific research, and technical problem solving, particularly when users benefit from inspectable multimodal steps. It also carries risks: plausible but incorrect solutions may be over-trusted, the system may be misused in educational assessment, and uploaded images may contain personal information. Deployment should therefore preserve uncertainty and execution records, support human oversight, evaluate reliability across user groups and problem types, and apply privacy-aware data handling.

Step 2: Dynamic Planning for Next Step

"step_id": 2,
"tool_name": "Calculation_Execution_Tool",
"task": "Calculate the maximum wavelength using the formula λ = c / v with the minimum frequency of the AM band.",
"input": {
"primary_focus": "Perform the calculation",
"context_summary": "Using c = 3 × 10^8 m/s and v = 530 × 10^3 Hz, calculate λ.",
"previous_results": "From step 1: AM band minimum frequency is 530 kHz.",
"original_question": "What is the maximum wavelength in m for broadcasts on the AM band?"
},
"expected_outcome": "λ = m",
"dependencies": [1],
"context": "Ensure units are consistent (Hz for frequency, m/s for speed of light).",
"verification_strategy": "checkpoints": ["Check the arithmetic calculation","Verify the units are correctly handled"],
"final_validation": "Re-calculate to ensure accuracy"

Step 2: Action

"Action Step 2": {
"tool_name": "Calculation_Execution_Tool",
"sub_goal": "TASK: Calculate the maximum wavelength using the formula λ = c / v with the minimum frequency of the AM band.\nEXPECTED_OUTCOME: λ = (3 × 10^8) / (530 × 10^3) ≈ 566 m\nDEPENDENCIES: [1]\nPREVIOUS_RESULTS (planned): From step 1: AM band minimum frequency is 530 kHz.",
"command": "execution = tool.execute(problem=\"What is the maximum wavelength in m for broadcasts on the AM band? Format your answer as an integer.\", context=\"The maximum wavelength (λ) in meters for broadcasts on the AM band is calculated using the formula λ = c / v, where c is the speed of light (3 × 10^8 m/s) and v is the minimum frequency in the AM band (530 kHz). Ensure units are consistent (Hz for frequency, m/s for speed of light).\", image_path=\"None\")",
"Atom_tool": "math_calculatorv2",
"query": "What is the maximum wavelength in m for broadcasts on the AM band",
"result": [
"steps": [
"Step 1: Convert the frequency from kHz to Hz: 530 kHz = 530 × 10^3 Hz",
"Step 2: Use the formula λ = c / v, where c = 3 × 10^8 m/s and v = 530 × 10^3 Hz",
"Step 3: Perform the division: λ = (3 × 10^8) / (530 × 10^3)",
"Step 4: Simplify the expression: λ = 3 × 10^8 / 5.3 × 10^5",
"Step 5: Calculate the result: λ ≈ 566.0377 meters"
],
"result": 566,
],

Step 2: Verify

Task: Verify the completeness and accuracy of the answer to the question.
Quality Assessment Framework:\n1. Step Completion: Current Step 5 / 2\n2. Step Execution Quality: Check whether each step produces a valid result.\n3. Logical Coherence: Whether a complete chain of solutions is formed between the steps.\n4. Final Answer: Whether a clear answer to the question has been obtained.\n\nValidation Rules:\n- If the current step number is less than the predefined total number of steps and the previous step has a valid result, CONTINUE.\n- If the current step number is greater than or equal to the predefined total number of steps and a complete solution is available, STOP.\n- If a step fails or the result is empty, but there are still steps left, CONTINUE.\n- STOP only when all necessary steps have been completed and the final answer has been obtained.\n\nCurrent Status Analysis:\n- Completed Steps: 5\n- Total Planned Steps: 2\n- Step Completion Rate: 5 / 2\n\nBased on the above evaluation, make a judgment:\n\nIf the answer is complete, accurate, and reliable, or all predefined steps have been completed:\n\nAnalysis: [Details on the completeness, accuracy, and reliability of the answer]\n\nSignal: true\nConclusion: STOP\n\nIf you need to continue with the remaining steps:\n\nAnalysis: [State the reason for continuing and the next step plan]\n\nSignal: false\nConclusion: CONTINUE\n\nPlease output strictly in the above format.\n", "response": "Analysis: The answer provided for the maximum wavelength in the AM band is complete and accurate. The calculation was correctly performed using the minimum frequency of the AM band (530 kHz) and the speed of light (3 × 10^8 m/s), resulting in a wavelength of approximately 566 meters. The steps executed beyond the initial plan do not affect the correctness of the final answer for the given problem.\n\nSignal: true\nConclusion: STOP"

Output Summary

1. Problem Restatement and Understanding\n\n**Core Requirements:**\n- Determine the **maximum wavelength……
2. Problem-Solving Approach and Method\n\n**Strategy:**\n1. **Identify the minimum frequency** ……
3. Verification and Validation\n\n**Rationality Check:**\n- The AM band is known for its long wavelengths, typically ranging from **187 m to 566 m**. Our result (566 m) falls within this expected range.\n- The inverse relationship between frequency and wavelength holds: lower frequency (530 kHz) gives longer wavelength (566 m).\n\n**Calculation Verification:**\n- Recalculating:\n \\[\n \\frac{3 \\times 10^8}{530 \\times 10^3} = \\frac{300,000,000}{530,000} \\approx 566.0377 \\, \\text{m}\n \\]\n Rounding to the nearest integer confirms **566 m**.
4.Conclusion:\nThe solution is correct and meets all problem requirements. The final answer is \\(\\boxed{566}\\)."

Figure 9: Extended visual summary of the trajectory case study.