



TaskCraft: Automated Generation of Agentic Tasks

OPPO AI Agent Team

Full author list in Contributions

Abstract

Agentic tasks, which require multi-step problem solving with autonomy, tool use, and adaptive reasoning, are becoming increasingly central to the advancement of NLP and AI. However, existing instruction data lacks tool interaction, and current agentic benchmarks rely on costly human annotation, limiting their scalability. We introduce **TASKCRAFT**, an automated workflow for generating difficulty-scalable, multi-tool, and verifiable agentic tasks with execution trajectories. TaskCraft expands atomic tasks using depth-based and width-based extensions to create structurally and hierarchically complex challenges. Empirical results show that these tasks improve prompt optimization in the generation workflow and enhance supervised fine-tuning of agentic foundation models. We present a large-scale synthetic dataset of approximately 36,000 tasks with varying difficulty to support future research on agent tuning and evaluation.

Date: June 13, 2025

Correspondence: zhouwangchunshu@oppo.com

Code & Data: <https://github.com/OPPO-PersonalAI/TaskCraft>

1 Introduction

Agentic tasks—autonomous, multi-step problem-solving requiring tool use and adaptive reasoning—are increasingly pivotal in AI and NLP. Advances in language agents [6, 15, 26, 38–40] have shifted AI from passive assistance to proactive agency, enabling complex workflow execution. This is exemplified by systems combining reasoning frameworks like ReAct [33] with dynamic orchestration, where solution trajectories critically improve inference quality. However, the inherent complexity of such tasks challenges conventional annotation paradigms, necessitating novel approaches to model training and evaluation.

To assess advanced agent capabilities, benchmarks such as GAIA [8], BrowseComp [25], and Humanity’s Last Exam (HLE) [9] have been introduced. GAIA evaluates reasoning, tool use, and web browsing through 466 real-world questions. BrowseComp comprises 1,266 tasks that test an agent’s ability to retrieve and integrate complex online information. HLE includes 2,500 multi-modal questions across over 100 disciplines to measure advanced reasoning and domain knowledge. While these datasets have significantly contributed to agent evaluation, they suffer from scalability limitations due to the labor-intensive nature of data annotation. For example, creating HLE required 1,000 experts to label just 2,500 data points, hindering its ability to scale.

Prior work has explored the automatic generation of instruction-following data using large language models to alleviate the scalability issues of human-annotated datasets. A representative example is the Self-Instruct framework [24], which demonstrated that LLMs can generate high-quality, diverse instruction data for multi-turn dialogues. This approach has proven effective for supervised fine-tuning (SFT). However, these methods

are primarily designed for static instruction-following scenarios and fall short in modeling agentic tasks, which require interaction with external tools and environments. Consequently, such data is insufficient for training or evaluating agents that operate in dynamic, real-world settings.

In this work, we introduce TASKCRAFT, an agentic workflow for the automated generation of agentic tasks. Our approach provides the following advantages:

- **Scalability.** The workflow supports adaptive difficulty, seamless multi-tool integration, and the generation of tasks beyond the capabilities of the task-generation agent, along with their corresponding trajectories.
- **Efficient Verification.** During each task extension, only incremental components undergo agentic validation, eliminating the need for full verification of the extended task.

The core approach involves initially generating multiple atomic tasks, each solvable with a single target tool invocation, and then expanding them using depth-based and width-based extension. For depth-based task extension, we iteratively transform specific textual elements of the original task (such as key terms) into a new atomic task to support progressive resolution. In contrast, the width-based extension formulates tasks that require resolving multiple sub-problems by integrating distinct problem instances.

To ensure high-quality agentic tasks, we employ a rejection sampling strategy during verification. For atomic tasks, we include cases where an agent using external tools can solve the task while an LLM cannot, ensuring that atomic tasks genuinely necessitate tool usage. For extension tasks, we leverage linguistic analysis with LLMs, enabling rapid validation and facilitating the creation of challenges beyond existing agent capabilities. This approach enhances efficiency and broadens problem-solving potential.

The controlled generation process ensures inherent access to ground-truth execution trajectories, enabling precise interpretability, reproducibility, and verifiability—critical for agent evaluation and reinforcement learning. To further validate task effectiveness, we implement a self-evolving prompt optimization strategy inspired by bootstrap few-shot learning [5]. This iterative refinement improves rejection sampling pass rates while minimizing generation time. Additionally, we leverage the generated task trajectories to train an agent foundation model [4]. Experimental results show that an independent LLM, trained on these trajectories, effectively plans and invokes tools, yielding performance gains on HotpotQA [32], Musique [21], and Bamboogle [10].

Based on this method, we generated a task dataset comprising approximately 36,000 tasks of varying difficulty, each requiring different tools for resolution, including search, web browsing, PDF reading, and image understanding.

Our key contributions are as follows:

- We introduce an automated agentic task generation workflow capable of producing scalable difficulty, efficient verification, and multi-tool supported tasks, along with their corresponding execution trajectories.
- We empirically evaluate task effectiveness through prompt learning, which facilitates the self-evolution of our workflow and holds potential for optimizing existing agent workflows. Additionally, SFT is applied to an agent foundation model, enabling it to substitute agent workflows where applicable.
- We release a synthetic dataset comprising about 32k agentic tasks of varying difficulty levels, complete with their execution trajectories, to facilitate further research.

2 Notations and Preliminary

Tool-Assisted Task Execution

As Figure 1 shown, given a task q , the agent extracts the input index i_T (e.g., document name, webpage title) for invoking a target tool T . We focus solely on steps that yield a valid tool context, omitting unrelated processes such as file location or search for simplicity. Executing tool T with i_T retrieves the associated context C . The LLM implicitly deduces the relationship R between C and the expected outcome, producing the final result a .

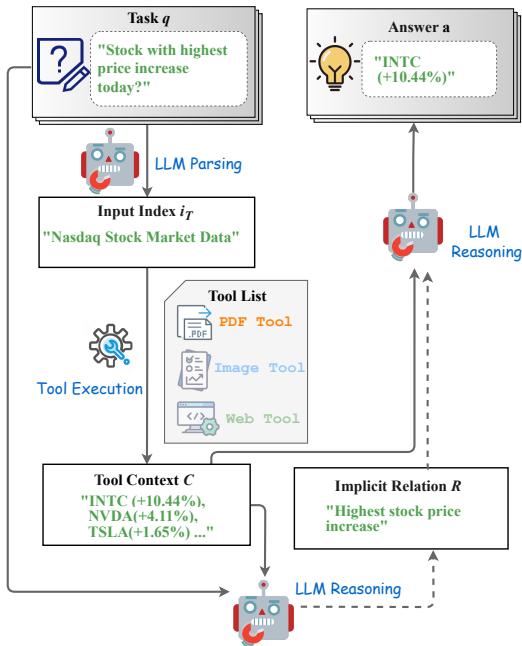


Figure 1 Execution flow of a single tool invocation. The agent extracts the input index i_T (e.g., document name, webpage title) for invoking tool T , focusing solely on steps that yield valid tool context. Executing T with i_T retrieves context C , enabling the LLM to infer the relationship R and produce the final result a .

Atomic Task

An atomic task is resolved with a single target tool invocation. To simplify, we disregard search and file system operations, assuming a detailed input index i_T enables retrieval through finite navigation.

Given an answer a , the most direct approach to construct an atomic task involves prompting an LLM to generate the corresponding question. However, questions produced in this manner often suffer from low tool invocation rates, unpredictable difficulty levels, unregulated tool requirements, and inconsistent verification complexity (see Section 4.5 for more details).

To mitigate these issues, we assume an ideal search engine capable of retrieving precise data based on i_T (e.g., paper titles, image paths, music names, etc.). Under this assumption, we can construct a task question $q = f(i_T, R) \rightarrow a$, where f represents a sampling function that enables the LLM to generate the corresponding natural language representation of the question q based on the provided information.

3 Automated Task Generation Workflow

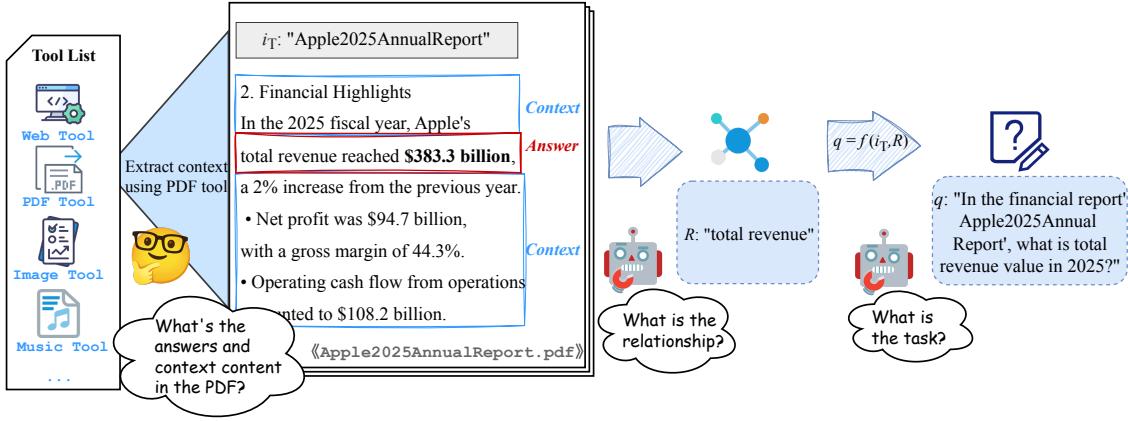


Figure 2 Atomic task generation: From an unlabeled corpus, we extract i_T and derive textual content C via tool execution. LLM identifies candidate answers a from C , infers their relationship R , and constructs question q conditioned on i_T and R .

3.1 Atomic Task Generation

As Figure 2 shown, we begin by compiling a corpus of unlabeled data aligned with the tool’s input requirements. From this corpus, we extract i_T and derive textual content C via tool execution. For example, browsing, PDF, and image comprehension tools yield webpage titles, PDF names, and image paths, from which we extract textual content C for answer sampling. We prompt an LLM to identify key candidate answers a from C and infer their relationship R with C , ultimately constructing question q conditioned on i_T and R .

3.2 Task Extension

In order to increase task difficulty in a scalable way, we adopted two extended task strategies: the depth-based extension and the width-based extension.

Depth-based extension. We aim to construct tasks requiring multiple sequential tool executions, where each step depends on the output of the previous one. To achieve this, a new subproblem must be derived from a known problem q^n . The tool input index i_T at each stage exhibits strong extensibility due to (1) its frequent association with proper nouns, which are less likely to be memorized by LLMs, and (2) its natural suitability for recursive definition. Specifically, a single atomic task follows the formulation:

$$q^n = f(i_T^n, R^n) \rightarrow a. \quad (1)$$

To extend a n-hot task q^n into a $(n+1)$ -hop dependency task q^{n+1} , we can define the recursive formulation:

$$q^{n+1} = f(\hat{q}^{n+1}, R^n) \rightarrow a, \quad (2)$$

where we ensure that

$$\hat{q}^{n+1} = f(i_T^{n+1}, R^{n+1}) \rightarrow i_T^n. \quad (3)$$

Here, i_T^{n+1} denotes a new tool input index derived from i_T^n through reversible operations (e.g., retrieving lyrics from a song name or vice versa). To obtain i_T^{n+1} and its corresponding relationship R^{n+1} , we employ a search agent that retrieves supersets of i_T^n to mitigate cyclic generation risks. Specifically, the agent extracts textual content C^{n+1} as superset candidates, expanding contextual coverage. An LLM then analyzes C^{n+1} to derive the superset index i_T^{n+1} and its relationship R^{n+1} with i_T^n . This process ensures progressive context expansion and effective information association. The resulting i_T^{n+1} and R^{n+1} are synthesized into an intermediate question candidate \hat{q}^{n+1} , which undergoes rigorous verification. Upon verification, the system generates the refined question q^{n+1} by integrating \hat{q}^{n+1} with all historical relationships $\{R^1, R^2, \dots, R^n\}$.

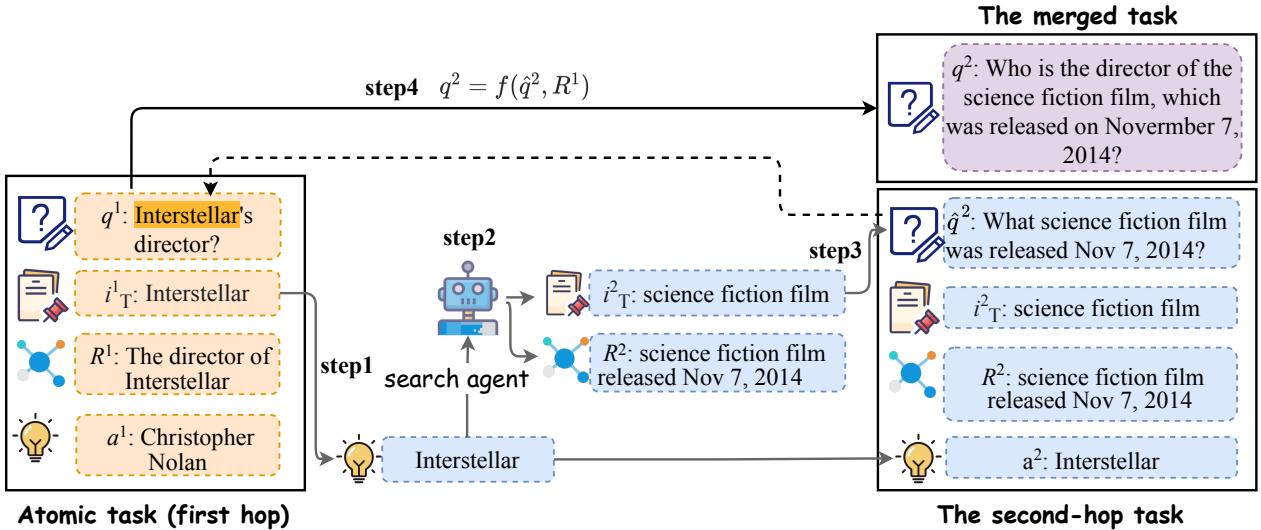


Figure 3 Depth-based extension. A 1-hop task q^1 is recursively extended to a 2-hop task q^2 . A search agent derives the new tool input index i^2_T by extracting superset candidates C^2 , which an LLM analyzes to determine i^2_T and its relationship R^2 with i^1_T . After verification, the refined question q^2 integrates \hat{q}^2 with historical relationships R^1 .

Width-based extension. The goal of the width-based extension is to generate a new task that needs to be decoupled into multiple subtasks to be completed. For simplicity, for two subtasks $q_1 \rightarrow a_1$ and $q_2 \rightarrow a_2$, the combined task q_{width} can be represented as

$$(q_{width} = q_1 + q_2) \rightarrow a_1 + a_2, \quad (4)$$

where the $+$ indicates using LLM to merge and rephrase two question strings.

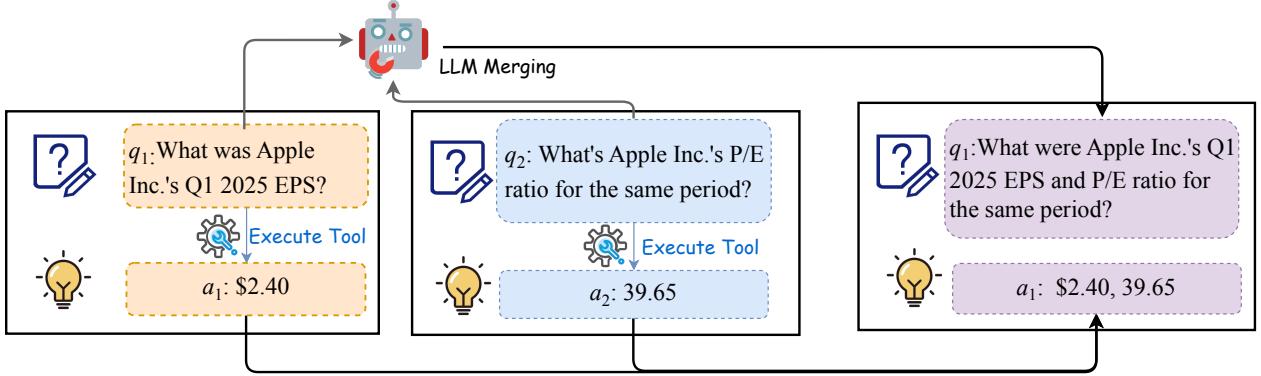


Figure 4 Width-based extension. A new task is formed by merging two subtasks q_1 and q_2 , creating $q_{width} = q_1 + q_2$, where $+$ denotes LLM-based rephrasing.

Trajectory generation. Two strategies exist for generating execution trajectories in this task: (1) For simple tasks, such as atomic tasks, existing agents can directly infer and capture the trajectory, including tool selection, parameters, return results, and plans. (2) For complex tasks, such as depth-wise extension tasks, the trajectory is recorded while iteratively expanding and validating new atomic tasks. At each step, the LLM refines the plan or reasoning based on generated intermediate questions.

3.3 Task Verification

Under this generation workflow, the verification of generated tasks can be easily performed in two distinct phases:

Atomic task verification: An atomic task is defined as a simple agent task solvable via a single tool call. During verification, we relax this definition slightly: for each candidate task, we evaluate the task agent’s output within a limited number of tool-use steps (e.g., three) and compare it with an infer-LLM separately. A judge-LLM verifies whether only the agent’s output contains the golden answer, retaining only validated tasks. (see Appendix C for more details)

Task extension verification: This process is conducted purely through linguistic analysis without agent involvement. During depth-wise extension, we first employ a judge-LLM to validate: (1) whether the obtained i_T^{n+1} and its relation R^{n+1} constitute a proper superset of i_T^n with logically sound relationships, and (2) whether the final input index i_T^n in q^n is appropriately replaced by \hat{q}^{n+1} in the expanded task q^{n+1} . Furthermore, an infer-LLM derives the merged task, while the judge-LLM filters out tasks where the correct result is easily inferred, preventing information leakage that could render the problem trivially solvable after merging. (see Appendix B for more details).

This framework ensures efficiency by applying agent reasoning only in atomic task verification at creation, while relying on LLM-based verification elsewhere for faster execution. It also enables complex task generation beyond agent capabilities, with reverse reasoning providing supervisory signals to enhance agent learning or reinforcement learning.

4 Experiments

4.1 Corpus Construction

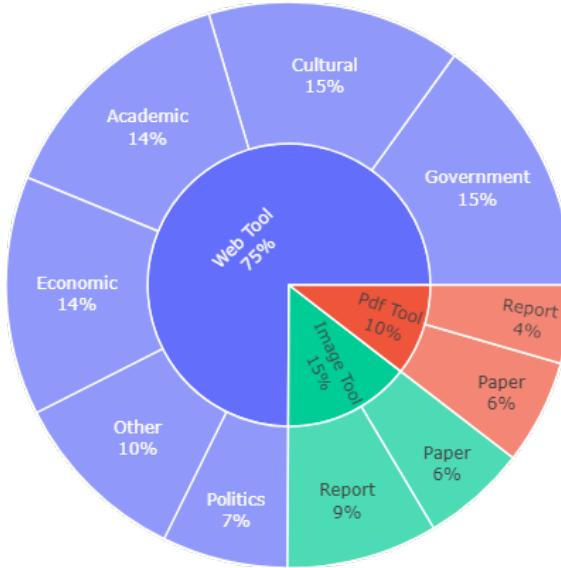


Figure 5 Corpus source distribution. Webpages, PDFs, and images are processed to construct tool-specific tasks.

We collect seed documents across multiple modalities to generate tool-specific atomic tasks, extracting key insights to ensure task relevance. For instance, our PDF processor constructs atomic tasks by combining document titles with core findings, thereby enhancing the necessity for agent-based PDF tool invocation. To support atomic task generation, we constructed a dataset comprising webpages, PDF files, and images. Webpage data constitutes the largest proportion (75%), sourced from up-to-date news across multiple domains. Image data accounts for 15%, primarily derived from financial reports and research papers, with filtering

to retain images containing information beyond text. PDF data makes up 10%, originating from English financial documents and academic publications.

4.2 Synthetic Tasks Analysis

Agent reasoning analysis . To practically assess task difficulty, we sample 1,000 tasks and deploy both Smolagents [14] and its enhanced variant, Smolagents+ (see Section E for more details), for execution and validation. While both agents performed identical tasks, Smolagents+ incorporated advanced tool capabilities for refined analysis.

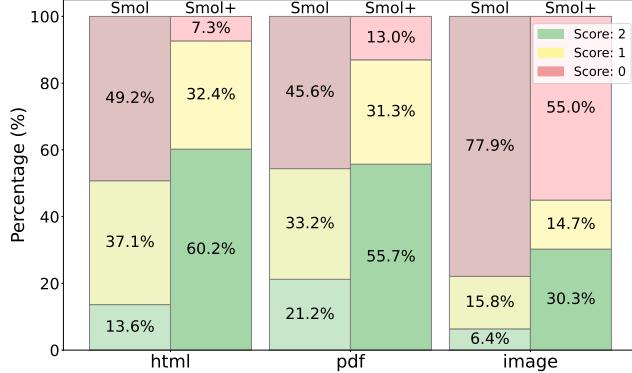


Figure 6 score distribution comparison

Responses were evaluated by comparing the agents’ outputs to the golden answer, following a three-point scoring scheme: 2 for fully correct responses, 1 for answers that included the golden answer but contained additional information, and 0 for incorrect responses.

In Figure 6, task failure rates increase from web pages to PDFs and then to images within PDFs, indicating that multi-hop web search tasks are more manageable for agents, while complex comprehension challenges, such as PDF extraction and image interpretation, remain difficult. Additionally, these results demonstrate that our generated tasks span varying difficulty levels, including those that pose significant challenges for current agent capabilities.

Comparison with the GAIA dataset. Table 1 presents the accuracy comparison of Smolagent on the GAIA dataset and our generated dataset. The results indicate that tasks derived from different tool corpora align with GAIA’s varying difficulty levels, with image understanding tasks posing the greatest challenge and achieving accuracy comparable to LEVEL3 data.

Table 1 Accuracy comparison of Smolagents on the GAIA dataset and our synthetic tasks.

GAIA	Level1	Level2	Level3	Avg.
	54.71	43.02	26.92	44.20
Synthetic Task	PDF	html	Image	Avg.
	54.4	50.7	22.1	42.4

Unlike GAIA, which requires extensive human annotation, our approach automates task generation, eliminating the need for labor-intensive data labeling while maintaining scalability and adaptability for agent self-evolution and optimization.

4.3 Enhancing Task Generation Efficiency via Prompt Learning

We employ rejection sampling in both atomic task generation and task extension. To reduce the rejection rate and enhance sampling efficiency, several key challenges must be addressed:

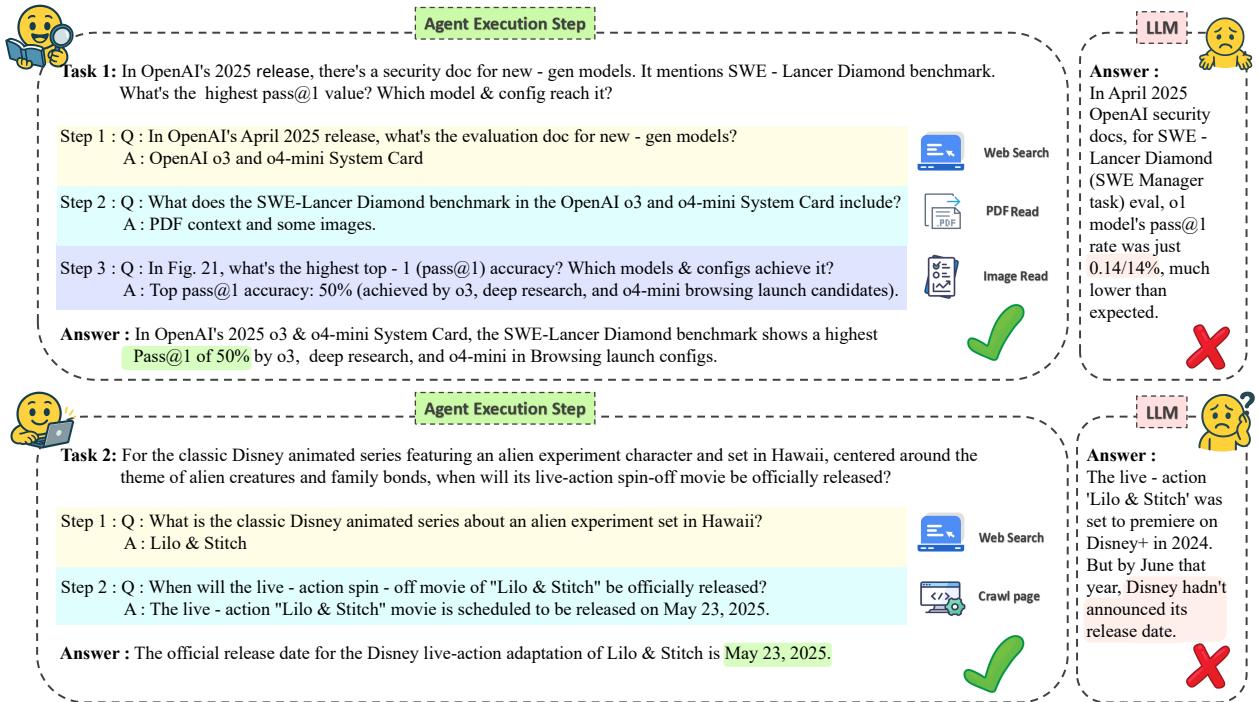


Figure 7 Generated case examples requiring multiple tool calls for completion.

- Efficiently extract candidate answers from the corpus to support atomic task formation and minimize rejections (Section 3.1).
- Guide the agent to find an input index i_T^{n+1} , ensuring coherent depth-wise extension.
- Prompt the LLM in depth-wise extension to articulate the relationship R^{n+1} between the previous input index i_T^n and observed content C^{n+1} , refining problem construction and mitigating incoherence-related rejections.
- Integrate tasks to ensure precise substitution, i.e., $q^{n+1} = f(\hat{q}^{n+1}, R^n)$, and clarity while maintaining logical coherence.

Evaluation. We assess atomic task generation and task extension separately. For atomic task generation, we evaluate three key metrics: (1) pass rate, representing the proportion of successfully validated atomic tasks relative to candidate tasks. (2) task density, quantifying the average number of validated atomic tasks per document. (3) sampling time, measuring the time required for processing each document.

For task extension, we evaluate three key metrics: (1) pass rate, the proportion of successful extensions across n_k attempts (set to 6 in our experiment). (2) sampling time, measuring the time required for extending each task.

Prompt Learning. Intuitively, providing the LLM with effective exemplars can further enhance its ability to identify intermediate objectives. To this end, we employ bootstrap few-shot learning [5] to systematically optimize the four prompts corresponding to the aforementioned challenges, thereby facilitating the generated workflow.

For atomic task generation, each prompt is optimized by appending 20 randomly sampled examples. Multiple prompt configurations are then generated by varying these samples, followed by an iterative evaluation process where pass rates determine the optimal selection of inserted examples. For task extension, we focus on depth-wise extension and adopt a similar strategy to optimize the prompts using 10 randomly sampled examples. These prompts are refined to maximize the number of hops.

Table 2 Effectiveness of generated task data in prompt learning and depth-wise extension across six extension attempts.

Method	Pass rate	Time
Atomic Task	54.9%	29.1s
+ Optimization	68.1%	23.5s
Depth-wise@6	41.0%	31.5s
+ Optimization	51.2%	30.2s

Results. Table 2 examines atomic task generation and depth-wise task extension before and after prompt learning, highlighting the role of generated task data in enabling self-evolution within both workflows. For atomic task generation, the data improves efficiency by reducing generation time by 19.2% (29.1 to 23.5 seconds) and increasing pass rate from 54.9% to 68.1%. Similarly, depth-wise extension benefits from the data, with pass rate rising by 10.2% (41.0% to 51.2%) across six extension attempts, and generation time decreasing by 1.3 seconds (31.5 to 30.2 seconds). These results validate the effectiveness of generated task data in enhancing sampling efficiency and supporting workflow adaptation. The optimized prompts are presented in Appendix C.2.

4.4 Fine-Tuning Agent Models Using Synthetic Trajectory

To validate the effectiveness of our synthetic multi-hop data method, we apply supervised fine-tuning (SFT) and reinforcement learning (RL) using the generated trajectory, refining an agent foundation model—an LLM with tool-integrated reasoning.

Evaluation. We evaluate our models on three multi-hop question answering benchmark datasets, as follows: HotpotQA [31], Musique [22], and Bamboogle [11]. These datasets encompass a diverse range of search with reasoning challenges, enabling a comprehensive evaluation.

Baselines. We conduct a comprehensive evaluation by comparing various baseline models before and after SFT with generated tasks to assess performance improvements: (1) Base workflow: We implement agent workflows (Search-R1 without training) across different LLM models. (2) Search-R1: An agentic workflow leveraging reinforcement learning for LLM model optimization.

Implementation setup. We evaluate two model variants: Qwen2.5-3B-Base and Qwen2.5-3B-Instruct. To facilitate multi-hop reasoning, we synthesize 3,202 multi-hop tasks and their trajectories for SFT. Following the Chain-of-Action framework [37], we apply content masking to search tool contexts during training. Our search method, RL training data, and reinforcement learning strategy follow the Search-R1 [4]. For further training details, refer to Appendix D.

Method	HotpotQA	Musique	Bamboogle	Avg.
Qwen2.5-3b-Base				
Base workflow	0.032	0.006	0.063	0.034
+ SFT	0.232	0.067	0.224	0.174
Search-R1	0.284	0.049	0.088	0.140
+ SFT	0.344	0.111	0.280	0.245
Qwen2.5-3b-Instruct				
Base workflow	0.190	0.037	0.112	0.113
+ SFT	0.221	0.049	0.248	0.173
Search-R1	0.324	0.103	0.264	0.230
+ SFT	0.340	0.104	0.264	0.236

Table 3 Performance across three datasets and two models. Avg. denotes average.

Results. As shown in Table 3, our method demonstrates significant performance improvements across three representative datasets and two model variants.

First, our synthetic data demonstrates significant value in standalone SFT training, achieving average performance improvements of +14.0% (Qwen2.5-3B-Base) and +6.0% (Qwen2.5-3B-Instruct) over the base

workflow for their respective models. These gains validate the quality and effectiveness of our synthetic data generation methodology.

Second, compared to the Search-R1 baseline, the workflow with Qwen2.5-3b-Base achieves maximum gains of +19.2% on Bamboogle and +6.2% on Musique. The Qwen2.5-3B-Instruct maintains steady gains, with an average performance margin of +0.6%. The strong performance of our SFT-trained models underscores their suitability for subsequent reinforcement learning, suggesting that our synthetic data not only enhances immediate task execution but also provides a more effective initialization for RL optimization.

4.5 Effectiveness of Tool Context in Constructing Agentic Tasks.

In atomic task generation, we integrate the additional input index i_T along with the relational mapping R between the tool context and a given answer to systematically structure tasks.

To assess the efficiency of our atomic task generation approach, we perform an ablation study using an LLM to directly generate a task q that requires only one external tool to obtain the answer a , explicitly excluding the conditions i_T and R . Evaluation metrics include pass rate, task resolution time, average tool usage, and the variance in tool usage frequency.

Table 4 The effectiveness of tool context.

Method	Pass rate	Time	#Tool-use	σ^2
LLM only	18.5%	119.7s	2.8	1.2
Ours	43.0%	86.7s	2.1	0.4

Compared to atomic tasks generated via direct prompting of GPT-4.1, our approach significantly enhances atomic task generation efficiency. Specifically, our workflow achieves a 24.5% higher pass rate (43.0% vs. 18.5%) while reducing task generation time by 28 seconds (86.7s vs. 119.7s), underscoring the limitations of vanilla LLMs in constructing agentic tasks. Furthermore, our atomic tasks exhibit greater atomicity, as evidenced by a lower average tool invocation count (2.1 vs. 2.8 per query). Task complexity also remains more stable and controllable, with a reduced variance in tool usage (0.4 vs. 1.2). These findings underscore the robustness of our workflow, validating its efficacy in structured task generation.

5 Related Work

5.1 Instruction Data Generation

Synthetic data has emerged as a promising solution for enhancing performance and enabling new capabilities. STaR [36] augments learning with chain-of-thought (CoT) rationales but often requires a substantial number of task queries beforehand. Methods such as Self-Instruct [24], Self-Chat [28], NuminaMath [7], and OpenMathInstruct-2 [19] generate data from minimal seed examples using LLMs, yet they struggle to extend task generation for multiple tool invocations. WizardLM [27] employs Evol-Instruct to incrementally enhance instruction complexity. However, it relies primarily on rule-based modifications, making its generated instructions unsuitable for agentic task scenarios. MetaMath [34] generates mathematical data by rewriting questions, but adapting agent tasks to environmental feedback presents challenges beyond simple rephrasing. WebInstruct [35] extracts question-answer pairs from a pre-training corpus across multiple domains; however, the generated questions often fail to incorporate tool utilization in their solutions. AutoAct [12] uses a self-planning mechanism to generate planning trajectories for QA tasks.

5.2 Language Agent

Existing research on agentic task execution primarily advances along two core dimensions: role specialization and functional partitioning. Role-based paradigms structure collaborative networks by dynamically allocating differentiated tools, as demonstrated by AutoGPT [15], AutoGen [26], and Camel [6]. In contrast, functional partitioning frameworks, such as Barcelona2, Omne, and AgentIM¹, define distinct task execution roles,

¹These are closed-source frameworks.

optimizing modular efficiency. Smolagents [14] combines the ReAct [33] and CodeAct [23] architectures to build a multi-functional agents hierarchy to perform multiple rounds of interactions and actions in code to accomplish complex tasks. Magnetic-One [2] refines vision-language processing by decoupling perception [29, 30], planning [16, 18], and execution modules [13, 23], improving efficiency in multimodal environments. Dynamic orchestration mechanisms address real-time task reallocation and system resilience. Trase-Agent [20] adapts execution strategies based on real-time feedback, while TapeAgents [1] employs asynchronous communication to enhance robustness in agent coordination. Empirical findings suggest that stabilized sub-agent interactions yield higher task success rates than complex, centralized orchestration algorithms.

To further extend agentic autonomy, AutoAgent [17] facilitates intelligent execution and personalized agent customization without requiring manual coding. Its core components—natural language-driven coordination, customizable workflows, and self-managing file systems—streamline agent development. Hybrid architectures, such as h2oGPTe-Agent [3], explore multi-agent optimization strategies, achieving over 70% accuracy in code generation tasks. However, significant cross-modal processing bottlenecks remain an open challenge.

6 Conclusion

We present TASKCRAFT, an automated workflow for scalable, multi-tool, verifiable agentic task generation. Through width-based and depth-based extension, our framework constructs hierarchically complex challenges. Empirical results demonstrate its effectiveness in structured task generation, improving prompt optimization and supervised fine-tuning while reducing reliance on human annotation. Additionally, we release a large-scale synthetic dataset of approximately 36,000 tasks with varying difficulty to support future research on agent tuning and evaluation.

7 Limitation

This work currently focuses on constructing atomic tasks for common tools, including browsing, PDF processing, and image analysis. Future iterations will enable users to generate atomic tasks tailored to their agents' specific tool requirements.

Contributions

Core Contributors

- Dingfeng Shi
- Qianben Chen
- Jingyi Cao

Contributors

- Weichen Sun
- Hongxuan Lu
- Tianrui Qin
- Minghao Yang
- Ge Zhang
- Changwang Zhang
- Yuchen Eleanor Jiang
- Weizhen Li
- Fangchen Dong
- King Zhu
- Jian Yang
- Jiaheng Liu
- Jun Wang

Corresponding Authors

- Wangchunshu Zhou

References

- [1] Dzmitry Bahdanau, Nicolas Gontier, Gabriel Huang, Ehsan Kamalloo, Rafael Pardinas, Alex Piché, Torsten Scholak, Oleh Shliazhko, Jordan Prince Tremblay, Karam Ghanem, Soham Parikh, Mitul Tiwari, and Quaizar Vohra. Tapeagents: a holistic framework for agent development and optimization, 2024. URL <https://arxiv.org/abs/2412.08445>.
- [2] Adamour, Gagan Bansal, Hussein Mozannar, Cheng Tan, Eduardo Salinas, Friederike Niedtner, Grace Proebsting, Griffin Bassman, Jack Gerrits, Jacob Alber, et al. Magentic-one: A generalist multi-agent system for solving complex tasks. *arXiv preprint arXiv:2411.04468*, 2024.
- [3] H2O.ai. Autonomous agentic ai: execute multi-step workflows autonomously. [Online], 2024. <https://h2o.ai/platform/enterprise-h2ogpte/#AgenticAI>.
- [4] 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, 2025. URL <https://arxiv.org/abs/2503.09516>.
- [5] Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T. Joshi, Hanna Moazam, Heather Miller, Matei Zaharia, and Christopher Potts. Dspy: Compiling declarative language model calls into self-improving pipelines. 2024.
- [6] Guohao Li, Hasan Hammoud, Hani Itami, Dmitrii Khizbulin, and Bernard Ghanem. Camel: Communicative agents for "mind" exploration of large language model society. *Advances in Neural Information Processing Systems*, 36:51991–52008, 2023.
- [7] 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, 2024.
- [8] 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*, 2023.
- [9] Long Phan, Alice Gatti, Ziwen Han, Nathaniel Li, Josephina Hu, Hugh Zhang, Chen Bo Calvin Zhang, Mohamed Shaaban, John Ling, Sean Shi, Michael Choi, Anish Agrawal, Arnav Chopra, Adam Khoja, Ryan Kim, Richard Ren, Jason Hausenloy, Oliver Zhang, et al. Humanity's last exam, 2025. URL <https://arxiv.org/abs/2501.14249>.
- [10] Ofir Press, Muru Zhang, Sewon Min, Ludwig Schmidt, Noah A Smith, and Mike Lewis. Measuring and narrowing the compositionality gap in language models. *arXiv preprint arXiv:2210.03350*, 2022.
- [11] Ofir Press, Muru Zhang, Sewon Min, Ludwig Schmidt, Noah A. Smith, and Mike Lewis. Measuring and narrowing the compositionality gap in language models. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Findings of the Association for Computational Linguistics: EMNLP 2023, Singapore, December 6-10, 2023*, pages 5687–5711. Association for Computational Linguistics, 2023. doi: 10.18653/V1/2023.FINDINGS-EMNLP.378. URL <https://doi.org/10.18653/v1/2023.findings-emnlp.378>.
- [12] Shuofei Qiao, Ningyu Zhang, Runnan Fang, Yujie Luo, Wangchunshu Zhou, Yuchen Jiang, Chengfei Lv, and Huajun Chen. AutoAct: Automatic agent learning from scratch for QA via self-planning. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3003–3021, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.165. URL <https://aclanthology.org/2024.acl-long.165/>.
- [13] Yujia Qin, Shengding Hu, Yankai Lin, Weize Chen, Ning Ding, Ganqu Cui, Zheni Zeng, Xuanhe Zhou, Yufei Huang, Chaojun Xiao, et al. Tool learning with foundation models. *ACM Computing Surveys*, 57(4):1–40, 2024.
- [14] Aymeric Roucher, Albert Villanova del Moral, Thomas Wolf, Leandro von Werra, and Erik Kaunismäki. ‘smol-agents’: a smol library to build great agentic systems. <https://github.com/huggingface/smolagents>, 2025.
- [15] Significant-Gravitas. Autogpt. [Online], 2023. <https://github.com/Significant-Gravitas/AutoGPT>.
- [16] Chan Hee Song, Jiaman Wu, Clayton Washington, Brian M Sadler, Wei-Lun Chao, and Yu Su. Llm-planner: Few-shot grounded planning for embodied agents with large language models. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 2998–3009, 2023.
- [17] Jiabin Tang, Tianyu Fan, and Chao Huang. Autoagent: A fully-automated and zero-code framework for llm agents. *arXiv e-prints*, pages arXiv–2502, 2025.

- [18] Jesus Tordesillas and Jonathan P How. Mader: Trajectory planner in multiagent and dynamic environments. *IEEE Transactions on Robotics*, 38(1):463–476, 2021.
- [19] Shubham Toshniwal, Wei Du, Ivan Moshkov, Branislav Kisacanin, Alexan Ayrapetyan, and Igor Gitman. Openmathinstruct-2: Accelerating ai for math with massive open-source instruction data. *arXiv preprint arXiv:2410.01560*, 2024.
- [20] Trase. Meet trase systems. [Online], 2024. <https://www.trasesystems.com/>.
- [21] Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. Musique: Multihop questions via single-hop question composition. *Transactions of the Association for Computational Linguistics*, 10:539–554, 2022.
- [22] Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. Interleaving retrieval with chain-of-thought reasoning for knowledge-intensive multi-step questions. In Anna Rogers, Jordan L. Boyd-Graber, and Naoaki Okazaki, editors, *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, ACL 2023, Toronto, Canada, July 9-14, 2023, pages 10014–10037. Association for Computational Linguistics, 2023. doi: 10.18653/V1/2023.ACL-LONG.557. URL <https://doi.org/10.18653/v1/2023.acl-long.557>.
- [23] Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. Executable code actions elicit better llm agents. In *Forty-first International Conference on Machine Learning*, 2024.
- [24] Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A Smith, Daniel Khashabi, and Hannaneh Hajishirzi. Self-instruct: Aligning language models with self-generated instructions. *arXiv preprint arXiv:2212.10560*, 2022.
- [25] Jason Wei, Zhiqing Sun, Spencer Papay, Scott McKinney, Jeffrey Han, Isa Fulford, Hyung Won Chung, Alex Tachard Passos, William Fedus, and Amelia Glaese. Browsecomp: A simple yet challenging benchmark for browsing agents. *arXiv preprint arXiv:2504.12516*, 2025.
- [26] 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 conversation. *arXiv preprint arXiv:2308.08155*, 2023.
- [27] Can Xu, Qingfeng Sun, Kai Zheng, Xiubo Geng, Pu Zhao, Jiazhuan Feng, Chongyang Tao, and Dixin Jiang. Wizardlm: Empowering large language models to follow complex instructions. *arXiv preprint arXiv:2304.12244*, 2023.
- [28] Canwen Xu, Daya Guo, Nan Duan, and Julian McAuley. Baize: An open-source chat model with parameter-efficient tuning on self-chat data. *arXiv preprint arXiv:2304.01196*, 2023.
- [29] Dingkang Yang, Kun Yang, Yuzheng Wang, Jing Liu, Zhi Xu, Rongbin Yin, Peng Zhai, and Lihua Zhang. How2comm: Communication-efficient and collaboration-pragmatic multi-agent perception. *Advances in Neural Information Processing Systems*, 36:25151–25164, 2023.
- [30] Kun Yang, Dingkang Yang, Jingyu Zhang, Hanqi Wang, Peng Sun, and Liang Song. What2comm: Towards communication-efficient collaborative perception via feature decoupling. In *Proceedings of the 31st ACM international conference on multimedia*, pages 7686–7695, 2023.
- [31] Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun’ichi Tsujii, editors, *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, Brussels, Belgium, October 31 - November 4, 2018, pages 2369–2380. Association for Computational Linguistics, 2018. doi: 10.18653/V1/D18-1259. URL <https://doi.org/10.18653/v1/d18-1259>.
- [32] Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W Cohen, Ruslan Salakhutdinov, and Christopher D Manning. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. *arXiv preprint arXiv:1809.09600*, 2018.
- [33] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023.

- [34] Longhui Yu, Weisen Jiang, Han Shi, Jincheng Yu, Zhengying Liu, Yu Zhang, James T Kwok, Zhenguo Li, Adrian Weller, and Weiyang Liu. Metamath: Bootstrap your own mathematical questions for large language models. *arXiv preprint arXiv:2309.12284*, 2023.
- [35] Xiang Yue, Tianyu Zheng, Ge Zhang, and Wenhui Chen. Mammoth2: Scaling instructions from the web. *Advances in Neural Information Processing Systems*, 37:90629–90660, 2024.
- [36] Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah D Goodman. Star: Self-taught reasoner bootstrapping reasoning with reasoning. In *Proc. the 36th International Conference on Neural Information Processing Systems*, volume 1126, 2024.
- [37] Yuxiang Zhang, Yuqi Yang, Jiangming Shu, Xinyan Wen, and Jitao Sang. Agent models: Internalizing chain-of-action generation into reasoning models, 2025. URL <https://arxiv.org/abs/2503.06580>.
- [38] Wangchunshu Zhou, Yuchen Eleanor Jiang, Peng Cui, Tiannan Wang, Zhenxin Xiao, Yifan Hou, Ryan Cotterell, and Mrinmaya Sachan. Recurrentgpt: Interactive generation of (arbitrarily) long text, 2023. URL <https://arxiv.org/abs/2305.13304>.
- [39] Wangchunshu Zhou, Yuchen Eleanor Jiang, Long Li, Jialong Wu, Tiannan Wang, Shi Qiu, Jintian Zhang, Jing Chen, Ruipu Wu, Shuai Wang, Shiding Zhu, Jiyu Chen, Wentao Zhang, Xiangru Tang, Ningyu Zhang, Huajun Chen, Peng Cui, and Mrinmaya Sachan. Agents: An open-source framework for autonomous language agents. 2023. URL <https://arxiv.org/abs/2309.07870>.
- [40] Wangchunshu Zhou, Yixin Ou, Shengwei Ding, Long Li, Jialong Wu, Tiannan Wang, Jiamin Chen, Shuai Wang, Xiaohua Xu, Ningyu Zhang, Huajun Chen, and Yuchen Eleanor Jiang. Symbolic learning enables self-evolving agents. 2024. URL <https://arxiv.org/abs/2406.18532>.

Appendix

A Data Statistics

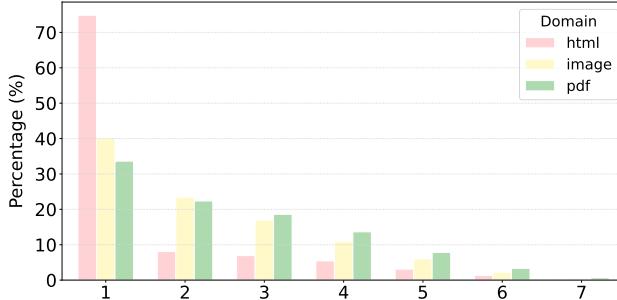


Figure 8 Analysis of all tasks.

As illustrated in Figure 8, task generation exhibits a hierarchical decay pattern across all domains as hop count increases, revealing distinct scalability trends:

- **PDF domain:** Shows gradual performance attenuation with hop depth, with 1-hop tasks accounting for 33.62% (2,737 tasks), decreasing to 22.36% (1,820 tasks) for 2-hop and 18.60% (1,514 tasks) for 3-hop. The sharp drop in 5-7 hop tasks (11.80% combined) indicates limited deep-extension capability, yet still surpasses other domains in depth scalability.
- **Image domain:** Presents the most pronounced performance decay, with 1-3 hops comprising 80.45% (4,342/5,397 tasks) but only 8.64% (467 tasks) for 5-7 hops, highlighting fundamental constraints in deep hierarchical task generation.
- **HTML domain:** In the HTML domain, 1-hop tasks dominate, constituting 74.84% (17,154 tasks) of the total. However, this domain also has the highest absolute number of deep extensions, with 5-7 hop tasks accounting for 4.75% (1,089 tasks).

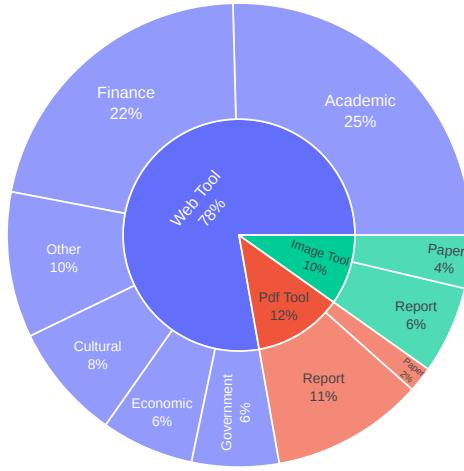


Figure 9 Distribution of atomic data.

Atomic task analysis. We collect data from webpages, PDF files, and images to support the generation of atomic tasks, which form the basis of the dataset, totaling 22,053 instances as shown in Figure 8.

Among them, atomic conclusions extracted by web-based tools account for the largest proportion, reaching 77.78%, with sources spanning multiple domains: academic (25.42%), financial (21.58%), cultural (8.09%),

economic (6.45%), and governmental (6.08%) resources. These conclusions are derived from up-to-date news and curated online materials to ensure relevance.

Image-based tools contribute 9.80% of the data, primarily extracting structured insights (e.g., key trends, comparisons) from charts and tables in financial reports and research papers. To avoid redundancy, we implement strict verification to exclude conclusions that directly replicate source text.

PDF-based extraction accounts for 12.41%, supplementing the dataset with findings from financial reports and academic publications. This multi-source approach enhances diversity while maintaining consistency in atomic fact representation.

By systematically integrating these extraction methods, we ensure high-quality task generation, providing a robust foundation for downstream model training and optimization.

B Verification Requirements for Depth-Based Extension

Effective n-hop task extension requires rigorous verification to ensure valid multi-hop reasoning. The transformation must preserve superset validity:

$$\hat{q}^{n+1} = f(i_T^{n+1}, R^{n+1}) \rightarrow i_T^n \quad (5)$$

$$q^{n+1} = f(\hat{q}^{n+1}, R^n) \rightarrow a \quad (6)$$

Current depth-based extension methods often introduce two critical flaws when replacing tool inputs i_T without proper verification:

- **Pseudo-Superset Problem:** Superficial substitutions that preserve semantic equivalence but lack genuine superset relationships
- **Information Leakage:** Premature disclosure of information that should only emerge through proper multi-step reasoning

These issues undermine the intended multi-hop reasoning process.

B.1 Pseudo-Superset Problem

A fundamental limitation arises when replacing i_T with a semantically equivalent but non-superset index i_T^{n+1} . Consider the following task extension example:

Original task

Query (q^n): How many travel trends for 2022 does 'Travel Trends 2025 | Our Annual Report' present?
Answer: 5

Substituting i_T ("Travel Trends 2025 | Our Annual Report") with the synonymous i_T^{n+1} ("2025 Annual Travel Trends Report") yields an intermediate task:

Intermediate task

Query (\hat{q}^{n+1}): What is the title of 2025 Annual Travel Trends Report?
Answer : Travel Trends 2025

Despite valid hop annotations, the intermediate question does not constitute an effective extension: it does not represent a necessary tool-use step. The core issue lies in the absence of a genuine superset relationship between i_T^n and i_T^{n+1} , leading to superficial expansion.

Extended task

Query (q^{n+1}): How many travel trends for 2022 does '2025 Annual Travel Trends Report' present?

Answer: 5

B.2 Information Leakage

A second failure mode occurs when expanded tasks inadvertently expose original answers, enabling large language models (LLMs) to bypass tool retrieval. For instance, consider the extended task:

Extended task

Query (q^{n+1}): In the AP Sports daily summary, Charter and Cox's proposed merger is valued at approximately \$34.5 billion. What is the exact amount?

Answer : 34.5B USD

While this query appropriately conceals the previous i_T^n ("Sports In Brief"), it directly reveals the answer "34.5B USD", allowing the LLM to bypass the intended retrieval process. This compromises the essential tool dependency required for multi-hop task answering.

B.3 Verification for Task Extension

To address these challenges, we propose a rigorous verification framework to ensure the validity of i_T^{n+1} , \hat{q}^{n+1} and q^{n+1} in task extension.

B.3.1 Strict Superset Verification

i_T^{n+1} must be the index of a strict superset of i_T^n , and the relationship can be formalized as:

$$\hat{q}^{n+1} = f(i_T^{n+1}, R^{n+1}) \rightarrow i_T^n \quad (7)$$

where R^{n+1} denotes hierarchical relations (e.g., *contains*, *part_of*). Valid extensions must introduce genuine depth, such as "*Sports In Brief*" \rightarrow "*AP News's Sports Section*" (relation: *contains*), while rejecting synonymous substitutions. Additionally, invalid extensions that allow the LLM to derive i_T^n directly should be excluded.

B.3.2 Information Leakage Verification

$$q^{n+1} = f(\hat{q}^{n+1}, R^n) \rightarrow a \quad (8)$$

The extended query q^{n+1} must adhere to the information-sealing principle to ensure proper tool-use reasoning. This requires that the query does not directly expose the original answer, and any query from which the LLM can directly obtain the answer should be filtered out.

B.4 Advantages of the Verification Framework

Our approach provides three key advantages:

- **Superset Integrity:** Guarantees valid hierarchical progression (e.g., *column* \rightarrow *page* \rightarrow *website*) without logical gaps.
- **Strict Tool Dependency:** Enforces authentic multi-hop reasoning by eliminating solution shortcuts, ensuring mandatory tool-use.
- **Transparent Reasoning:** Offers full explainability through explicit relation paths (R^n).

A properly expanded task under this framework would appear as follows:

Qualified Extended task

Query (q^{n+1}): According to the recurring AP News's sports section feature that regularly provides concise summaries of top sports events and highlights, what is the merger value currently being pursued by US cable giants Charter and Cox as they face increasing competition from streaming services?

Answer : 34.5B USD

C Core Prompts

This section presents key components of the verification prompts used in our framework.

C.1 Atomic task verification

The following prompt is used in atomic task verification (Section 3.3):

Atomic task verification

Task: Evaluate the *consistency* between the golden answer (GA) and another answer (AA, either agent or LLM-generated) as follows:

- **2 points (Fully Consistent):** AA and GA are semantically equivalent, even if phrased differently.

Example:

- GA: “Interest rates should be raised and inflation monitored.”
 - AA: “It is necessary to raise interest rates and monitor inflation.”

- **1 point (Partially Consistent):** AA includes all GA information but adds valid extra details.

Example:

- GA: “The interest rates should be raised.”
 - AA: “The interest rates should be raised, and inflation monitored.”

- **0 points (Inconsistent):** AA omits key GA information or contradicts it.

Examples:

- *Omission:* GA: “Raise rates and monitor inflation.”
 - AA: “Raise rates.”
- *Contradiction:* GA: “Raise rates by 50bps.”
 - AA: “Raise rates by 25bps.”

The criteria prioritize semantic equivalence while accommodating informative expansions or reductions.

Output Format: ...

A task is retained as an atomic task if and only if: (1) the *AgentScore* strictly exceeds the *LLMScore*, and (2) the *AgentAnswer* is non-zero.

C.2 optimized prompts

The following prompts is optimized prompt mentioned in (Section 4.3):

Atomic Conclusion Extraction

Task: Extract standalone conclusions from document chunks meeting these criteria:

1. **Atomicity:** Extract only indivisible basic facts (no combined conclusions, e.g., split “A increased by 5% and B decreased by 2%” into two separate conclusions)
2. **Verifiability:** Include at least one definite identifier (numeric value, time, unique name) and reject vague expressions (e.g., “Performance has improved”)
3. **Timeliness Handling:** Explicitly mark time ranges for time-sensitive information (e.g., “Global GDP grew by 3.0% in 2023” instead of “Recent GDP growth of 3.0%”)
4. **Citation Integrity:** Embed complete content of cited references (e.g., expand “as stated in (2)” to include the full text of (2) in the conclusion)

Valid Examples:

- **Example 1:** 3D deconvolution microscopy illumination optimization for refractive index tomography (Optics Express 29, 6293-6301, 2021)
- **Example 2:** Azimuthal energy Φ parameters: ($\theta_0 = 0.5$, $\theta_a = 2\pi/7$, $\theta_w = \pi/9$, $\theta_f = 0.06$, $p = 1.0004$, $q = 100$)
... (more examples omitted) ...

Output Format: ...

Depth-wise Extension: Index i_T^{n+1} Guidance and R^{n+1} Articulation

Task: Identify a minimal unique superset for an input element based on its attributes, ensuring the superset+relationship uniquely points to the element.

Examples:

1. Paragraph/sentence: Its belonging text content
2. Specific term: Corresponding discipline/category
3. Specific date: Date range it's in (e.g., its week/month)
4. Short event: Complete specific event it's part of
5. Page: Referencing pages or parent page
6. Generate only one relationship, avoiding strongly specific proper nouns

Relationship expression guidelines:

1. Clearly show hierarchical/ownership. Indicate position for series sub-items; clarify ownership for parts of a superset
2. Specify input content's positioning (e.g., time range, publication field, role in superset)
3. Use research/industry standard wording
4. Provide only necessary associations

Notes:

1. Return the superset's unique identifier (e.g., attribute name, page title, paper title)
2. Obtain superset content via tool (web, PDF, image)
3. Concisely describe the relationship, listing unique qualification conditions
4. Use ≤ 3 search keywords per search; do multiple searches if needed
5. Derive the identifier from search results, excluding the input content
6. Prioritize reading PDF content with tools if the input is a PDF

Valid Examples:

- **Example 1:**
 - **Input:** Avatar 3: Fire and Ash
 - **Superset Identifier:** Avatar film series
 - **Relation:** The third film
- **Example 2:**
 - **Input:** V3LMA: Visual 3D-enhanced Language Model for Autonomous Driving
 - **Superset Index:** cs.CV
 - **Relation:** A paper on visual 3D-enhanced language models for autonomous driving
... (more examples omitted)...

Output Format: ...

Logical Substitution: q^{n+1} as $f(\hat{q}^{n+1}, R^n)$

Task: Substitute elements in core queries using auxiliary queries while preserving:

1. **Complexity Balance:** The new query should be slightly more complex than the original core Query and require more steps to solve. But do not make too many changes to the core query.
2. **Answer Uniqueness:** The new query should point to the unique answer: golden answer, and should not point to other answers.
3. **Answer Concealment:** The new query must not reveal information about the golden answer.
4. **Natural Language Polish:** After merging, polish the question to make it conform to human expression habits without changing the original meaning. Do not modify the proper nouns appearing in it.

Valid Examples (20 in total):

- **Example 1:**
 - **Core Query:** What is the 2nd positive integer?
 - **Auxiliary Query:** Numbers except 0 in natural numbers
 - **New Query:** What is the 2nd natural number except 0?
- **Example 2:**
 - **Core Query:** Ne Zha 2 attendance ranking
 - **Auxiliary Query:** 2025 May Day box office summary
 - **New Query:** Given 2025 May Day box office data, what is Ne Zha 2's attendance ranking?
... (18 more examples omitted)

Output Format: ...

C.3 Strict Superset Verification

The following prompt is used in Appendix B.3.1:

Strict Superset Verification

Task: Verify if index i_T^{n+1} uniquely determines subset i_T^n under relation R^n in given queries.

Criteria:

1. **SupersetSubset Relationship:**
 - i_T^{n+1} must be the index of a superset that properly contains i_T^n
 - $i_T^{n+1} \not\propto i_T^n$ (excluding synonym pairs like CAR/AUTOMOBILE)
2. **Relationship Validity:**
 - The relationship R^n must explicitly and uniquely link the superset to the subset (no many-to-one mappings)

Output Format: ...

D Further Training Detail

For SFT training, we synthesize 3,202 multi-hop tasks and their trajectories and apply content masking to search tool contexts in these trajectories.

For RL training, we follow the Search-R1 [4] and use the 2018 Wikipedia dump as a knowledge source and the E5 embedding model as a retriever. For fair evaluation, we fix the retrieval depth to 3 passages for all methods. We merge the training sets of NQ and HotpotQA to form a unified dataset. Evaluation is conducted on the test or validation sets of three datasets to assess both in-domain and out-of-domain performance. Exact Match is used as the evaluation metric. In the PPO settings, we set the learning rate of the policy LLM to 1e-6 and that of the value LLM to 1e-5. Training is conducted for 500 steps, with warm-up ratios of 0.285 and 0.015 for the policy and value models, respectively. We use Generalized Advantage Estimation with parameters $\lambda = 1$ and $\gamma = 1$. We employ vLLM for efficient LLM rollouts, configured with a tensor parallelism degree of 1 and a GPU memory allocation ratio of 0.6. Our sampling strategy utilizes a temperature parameter of 1.0 and top-p threshold of 1.0. For policy optimization, we apply KL divergence regularization with coefficient

$\pi=0.001$ and implement a clip ratio $\epsilon=0.2$. The action budget is constrained to 4, with a default retrieval depth of 3 passages per query.

E Smolagents+

We developed Smolagents+, enhancing its web search capabilities, integrating multiple information sources, streamlining search results, and implementing a query rewriting strategy to optimize search performance.