

OctoTools: An Agentic Framework with Extensible Tools for Complex Reasoning

Pan Lu^{*}, Bowen Chen^{*}, Sheng Liu^{*}, Rahul Thapa, Joseph Boen, James Zou

^{*}Equal contribution Stanford University

Website: <https://octotools.github.io>

Code Demo Tool Cards Visualization

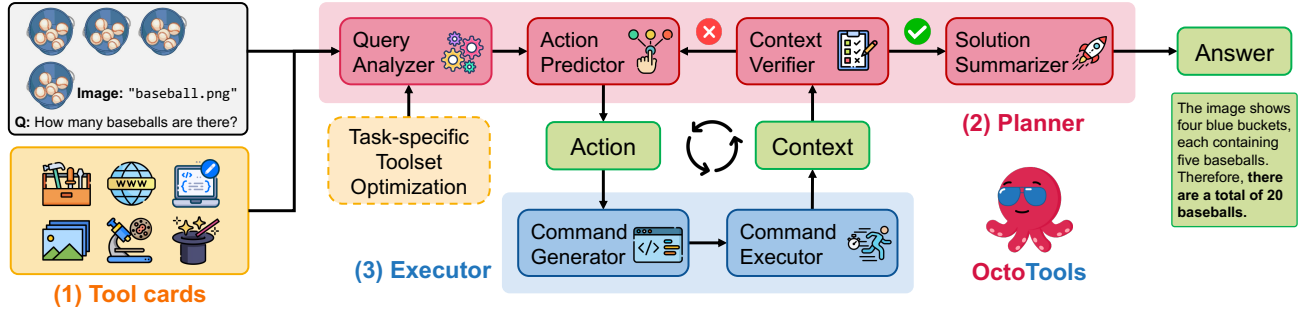


Figure 1. The framework of OctoTools. (1) *Tool cards* define tool-usage metadata and encapsulate tools, enabling training-free integration of new tools without additional training or framework refinement. (2) The *planner* governs both high-level and low-level planning to address the global objective and refine actions step by step. (3) The *executor* instantiates tool calls by generating executable commands and save structured results in the context. The final answer is summarized from the full trajectory in the context. Furthermore, the *task-specific toolset optimization* algorithm learns to select a beneficial subset of tools for downstream tasks. See Figure 3 for an example.

Abstract

Solving complex reasoning tasks may involve visual understanding, domain knowledge retrieval, numerical calculation, and multi-step reasoning. Existing methods augment large language models (LLMs) with external tools but are restricted to specialized domains, limited tool types, or require additional training data. In this paper, we introduce OctoTools, a training-free, user-friendly, and easily extensible open-source agentic framework designed to tackle complex reasoning across diverse domains. OctoTools introduces standardized tool cards to encapsulate tool functionality, a planner for both high-level and low-level planning, and an executor to carry out tool usage. We validate OctoTools’ generality across 16 diverse tasks (including MathVista, MMLU-Pro, MedQA, and GAIA-Text), achieving substantial average accuracy gains of 9.3% over GPT-4o. Furthermore, OctoTools outperforms AutoGen, GPT-Functions and LangChain by up to 10.6% when given the same set of tools. Through comprehensive analysis and ablations, OctoTools demonstrates advantages in task planning, effective tool usage, and multi-step problem solving.

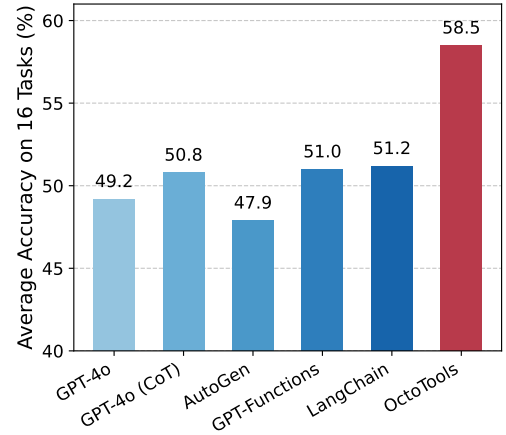


Figure 2. Performance comparison across 16 benchmarks. Our OctoTools framework achieves an average accuracy gain of 9.3% over GPT-4o without function plugins and 7.3% over LangChain, using the same tools under the same configuration.

^{*}Equal contribution. PL and RT started the project. PL completed the early framework. PL, BC refined the framework. PL, BC, and SL contributed to experiments and paper writing. Correspondence to: Pan Lu <panlu@stanford.edu>, James Zou <jamesz@stanford.edu>.

1. Introduction

Large language models (LLMs) (Brown et al., 2020; Chowdhery et al., 2022; OpenAI, 2023b) have made rapid progress on tasks such as summarization, translation (Thoppilan et al., 2022), code generation (Nakano et al., 2021), and math problem solving (Shuster et al., 2022). However, complex reasoning tasks that involve multiple steps, logical decomposition, or specialized domain knowledge remains challenging. For example, solving a visual riddle may require fine-grained image understanding and text-based reasoning, while a math or chemistry question can require thorough computations or domain expertise. Existing prompting methods often fail to orchestrate these varied processes into a coherent chain of reasoning (Yao et al., 2022).

A promising direction to address these challenges is to *augment* LLMs with *external tools*. By offloading specialized subtasks (e.g., web queries, Python-based calculations, and specialized scientific tools) to dedicated modules, LLMs can focus on higher-level planning and synthesis. Several frameworks have explored such tool usage, from those relying on extensive supervised data and fine-tuning (Schick et al., 2023; Liu et al., 2023), to static solutions without refinement (Lu et al., 2023), and those limited to one specialized domain of tools (Nakano et al., 2021; Tao et al., 2023; Hu et al., 2024). Although these methods perform well on specific tasks, they still face challenges that hinder general widespread use. Many require substantial training with curated data, which limits their adaptability to new domains. Others are designed for a particular domain (Bran et al., 2023; Kang & Kim, 2024; Li et al., 2024a; Schmidgall et al., 2024) or cannot easily support multi-step problem-solving (Lu et al., 2023), restricting their generality.

In this paper, we propose OctoTools, a *training-free* (i.e., it does not require updating model weights), *user-friendly*, and *extensible* agentic framework for tackling *complex reasoning* tasks across diverse domains (Figure 1). A key feature of OctoTools is the concept of *tool cards*, standardized wrappers that encapsulate heterogeneous tools (e.g., Python calculators, web search APIs, and domain-specific modules), along with metadata such as input-output formats, usage constraints, and best practices that delineate ideal use cases. This standardized design enables easy integration, replacement, or expansion of tools—unlike approaches requiring painstaking re-engineering for each new tool (Lu et al., 2023; Hu et al., 2024).

Building on these tool cards, OctoTools employs a dedicated *planner* that governs both high-level and low-level planning. Given a user query, the planner proposes a tentative global plan for how various tools might be employed. At each step, it generates a text-based *action* (including sub-goals and tool selection) conditioned on the evolving *context*. A separate *executor* instantiates tool calls by converting this

textual action into an executable command, running the corresponding tool, and updating the context with the results. By separating strategic planning from command generation, OctoTools reduces errors and increases transparency, making the system more reliable and easier to maintain.

An additional challenge in agentic systems is determining which subset of tools to enable for a given domain. Although providing many tools can be beneficial, enabling them all may introduce noise or slow performance (Lumer, 2024; Fore et al., 2024; Paramanayakam et al., 2024). To address this, we propose a lightweight *toolset optimization* algorithm that identifies a more useful subset of tools for each task based on validation performance, ultimately improving both accuracy and efficiency.

While recent general agent frameworks also allow LLMs to use external tools autonomously, they often focus on high-level abstractions (LangChain, 2024), limited observability of intermediate decisions (OpenAI, 2023a), or multi-agent collaboration features (AutoGen, 2024), with less emphasis on enhancing complex reasoning and *quantitatively* benchmarking downstream task performance. In contrast, we systematically evaluate the entire agentic workflow of OctoTools across diverse tasks, providing in-depth analyses of when and how tool-based reasoning succeeds or fails in complex reasoning scenarios.

We conduct large-scale experiments across 16 diverse reasoning benchmarks, spanning general vision, mathematical, scientific, medical, and agentic domains. As summarized in Figure 2, OctoTools substantially outperforms other baselines, achieving an average accuracy gain of 9.3% over zero-shot prompting by GPT-4o and 7.7% over chain-of-thought (CoT) prompting, as well as up to 10.6% improvement compared to existing agentic frameworks when given the same tools (AutoGen, 2024; OpenAI, 2023a; LangChain, 2024). Detailed analyses show that OctoTools effectively combines multi-step planning and specialized tool usage, with each dimension providing distinct improvements. For tasks requiring intricate calculations or specialized knowledge, we found tool usage is particularly beneficial; for tasks requiring reasoning decomposition, we found multi-step planning offers significant gains.

Furthermore, our ablation studies offer insights into OctoTools’s performance under different conditions. Overall, the average accuracy tends to improve as the maximum number of steps increases. Without any toolset optimization, simply enabling all tools in the toolset yields 57.4% accuracy, which still surpasses the setup with only the base tool by 3.5%, suggesting a degree of generalization as the toolset expands. Learning the optimal toolset for specific tasks raises the overall performance to 58.9%, indicating the benefit of further optimization. Additionally, when using a weaker LLM (GPT-4o-mini) as the base engine, OctoTools

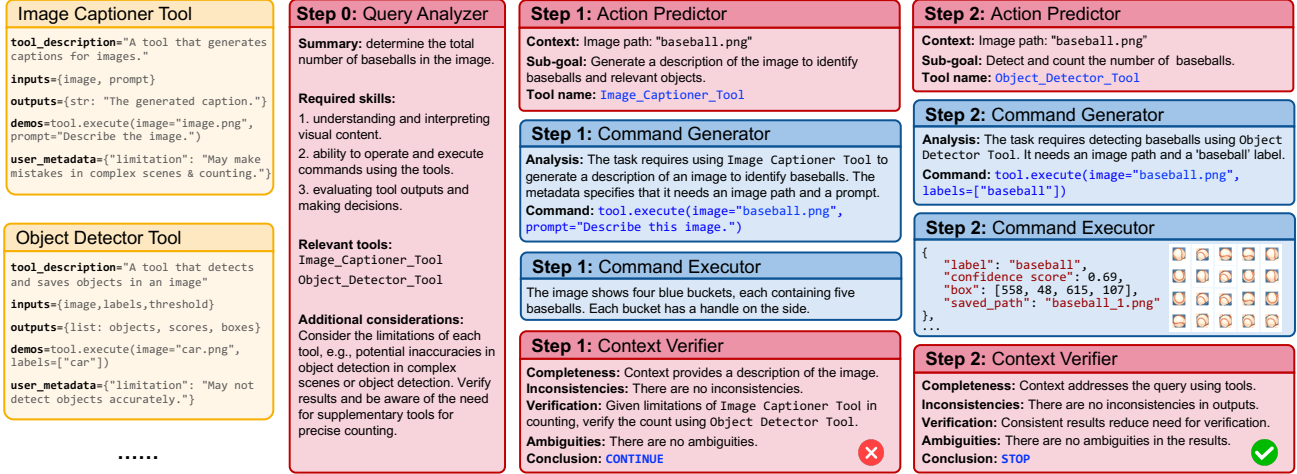


Figure 3. The demonstration of a self-contained example from Figure 1. We visualize the tool cards for selected tools, the initial plan generated by the planner, and two steps in which the planner and the executor orchestrate low-level planing and tool usage before arriving at the final answer. See §F.1 for details and §F for more examples. An interactive visualization of these examples is available at <https://octotools.github.io/#visualization>.

maintains a strong average gain of 7.1% across 16 tasks.

Our contributions are as follows: (1) We propose OctoTools, a training-free, extensible agentic framework that enables LLMs to call external tools in multiple steps, without the need for additional training or fine-tuning. (2) We introduce a comprehensive *planner-executor* paradigm with standardized *tool cards*, which can be easily customized or expanded for new domains. (3) We conduct large-scale experiments on 16 diverse benchmarks and show that OctoTools improves performance by a sizable margin compared to baseline prompting and other agentic frameworks. (4) We provide in-depth analyses and ablations on how multi-step reasoning and tool usage contribute to performance, offering practical guidance for future agent development.

2. Related Work

Tool-Augmented LLMs. A promising direction for enhancing large language models (LLMs) involves offloading specialized subtasks to external tools such as search engines (Komeili et al., 2022; Thoppilan et al., 2022; Lazaridou et al., 2022; Shuster et al., 2022; Yao et al., 2022), web browsers (Nakano et al., 2021), calculators (Cobbe et al., 2021; Thoppilan et al., 2022), translation systems (Thoppilan et al., 2022), or Python interpreters (Gao et al., 2023). Broadly, these methods either rely on large-scale fine-tuning or human supervision to teach LLMs how to invoke tools (Schick et al., 2023; Komeili et al., 2022; Nakano et al., 2021; Thoppilan et al., 2022) or use few-shot prompts for single tools in narrowly defined tasks (Yao et al., 2022; Lazaridou et al., 2022; Gao et al., 2023). In contrast, OctoTools is a *training-free* framework that integrates diverse

tools through standardized *tool cards* and employs a planner-executor paradigm to manage multi-step reasoning. Because new tools can be introduced without re-training, OctoTools offers a more *extensible* and *modular* approach to tool usage.

LLM Agents. A growing body of work leverages LLMs as autonomous agents that make decisions and invoke tools in multi-step workflows. Some agents use closed-source models with hand-engineered prompts (Chen et al., 2023; Wang et al., 2024a), while others fine-tune LLMs on curated data that learn when and how to call tools (Liu et al., 2023; Tao et al., 2023; Zhang et al., 2024). These frameworks often face limitations. For example, although specialized agent frameworks achieve strong performance in particular domains (e.g., chemistry (Bran et al., 2023), vision (Li et al., 2024a; Hu et al., 2024), materials science (Kang & Kim, 2024), or medical imaging (Schmidgall et al., 2024)), they typically lack generality across diverse tasks. Additionally, some systems are constrained by narrow capabilities with static planning (Lu et al., 2023) and multi-step reasoning (Hu et al., 2024). Recently, general-purpose agent platforms such as AutoGen (AutoGen, 2024), GPT-Functions (OpenAI, 2023a), and LangChain (LangChain, 2024) have emerged, but they have seen less emphasis on complex reasoning and rigorous benchmarking across diverse downstream tasks. In contrast, OctoTools combines the flexibility of such platforms with a dedicated planner and executor to handle multi-step decision-making.

Complex Task Reasoning. When faced with multi-step problems, a common strategy is to break down a question into simpler sub-questions and solve them step by step.

Early work approached decomposition with unsupervised or weakly supervised models (Perez et al., 2020; Khot et al., 2022), and more recent research has explored prompting techniques for step-by-step reasoning, including Chain-of-Thought (Wei et al., 2022), Least-to-Most (Zhou et al., 2022), ReAct (Yao et al., 2022), Pearl (Sun et al., 2023), Forest-of-Thought (Bi et al., 2024), and rStar-Math (Guan et al., 2025). While these methods significantly improve LLMs’ single-model reasoning capabilities, they primarily rely on the latent capacity of an LLM without external validation or targeted tool usage. In contrast, OctoTools systematically combines multi-step decomposition (via an iterative planner) with specialized tools (encapsulated by *tool cards*) and an executor for reliable, context-aware function calling. This design makes it easy to incorporate domain-specific functionalities and check intermediate steps with external modules, thus improving both correctness and versatility in tackling challenging tasks.

3. The OctoTools Framework

We propose OctoTools, an open-source, versatile, and user-friendly agent-toolbox framework for complex reasoning tasks, as illustrated in Figure 1. Given a user query $q \in \mathcal{Q}$ and a pretrained language model $\text{LLM}_\theta(\cdot)$, a naive approach would generate an output directly as $y \sim \text{LLM}_\theta(q)$, providing a single-step response. In contrast, our OctoTools framework introduces a structured, multi-step process that leverages external tools to tackle queries effectively.

Specifically, OctoTools contains a set of *tools* $\mathcal{D} = \{d_i\}_{i=1}^n$ and associated metadata $\mathcal{M} = \{m_i\}_{i=1}^n$, where n is the number of available tools. Given a query, a *planner* (based on a language model) first generates a *tentative plan* from a high-level perspective, indicating how these tools can be used to address the query, which forms the initial *context* s_0 . From this plan, the planner determines the initial *action* a_1 for tool usage, specifying which tool d_1 to use, the relevant context, and a sub-goal. An *executor* (also powered by a language model) then converts the planner’s text-based action a_1 into a machine-executable *command* o_1 , which is run to obtain intermediate results r_1 . These results, along with the original action, update the context to $s_1 := (a_1, o_1, r_1)$. This process constitutes one step in our framework.

This process repeats, with the planner iteratively refining its actions based on the evolving context until it either finds a complete solution or inference limits (e.g., time or steps) are reached. After T steps, the framework produces a full *trajectory* (s_0, s_1, \dots, s_T) , which is stored in a structured manner in the context. The planner then uses this trajectory to generate the final solution to the original query.

OctoTools provides a robust and effective framework for solving complex tasks through sub-goal decomposition and

systematical tool usage. Standardized *tool cards* encapsulate functionality (§3.1), the *planner* orchestrates both high-level and low-level task planning (§3.2), and the *executor* instantiates tool calls for each sub-goal (§3.3). The following sections detail the logic of each component, with implementation details provided in §C.

3.1. Tool Cards

To enable seamless interaction among tools, the planner, and the executor, the toolbox serves as a fundamental building block of OctoTools. In contrast to previous work (Lu et al., 2023; Hu et al., 2024) that rely on careful adaptations and tuning to support new tools, OctoTools leverages *tool cards*, which encapsulate tools in a modular manner.

Each *tool card* represents a tool d along its essential metadata m (as illustrated in Figure 3). This metadata includes the tool’s name, input and output types, and command demonstrations. It may also contain additional usage constraints (e.g., user-defined “*limitations*” and “*best practices*”), which provide developers’ insights to guide the planner and executor. For example, `Image_Captioner.Tool` includes “*it may make mistakes in complex scenes*” and “*consider using other tools for verification*” (§D.4), while `Object_Detector.Tool` notes the limitation in detecting objects (§E.1). See §D for details on featured tools.

To ensure consistent interactions, every tool card implements two standard functions. The function `execute()` encapsulates the tool’s primary functionality, e.g., generating code snippet or performing object detection. Executed results are stored in a structured format to support different output types, e.g., generated code, detected objects, stored files (see §D). The function `get_metadata()` allows the planner and executor to dynamically evaluate the tool’s capabilities and understand usage constraints.

The design of each tool card is modular relative to the framework, enabling users to integrate diverse tools without modifying the underlying framework or agent logic. New tool cards can be added, replaced, or updated with low effort. Consequently, OctoTools remains robust and extensible, even as tasks grow in complexity or scope.

3.2. Planner Module

Planner initialization. The planner inspects the *toolbox* and loads each *tool card* as defined in §3.1. Formally, it constructs a set $\{d_i\}_{i=1}^n$ of n available tools, each with metadata m that describes its input-output schema and usage constraints. Rather than enabling the full toolset, a subset $\mathcal{D}^* = \{d_i\}_{i=1}^k$ may be selected based on expert insights or optimized using a small set of examples (See §3.4 and §5.2 for details and experimental study).

Query analysis and action prediction. Given a query q , the planner formulates a *high-level*, tentative plan for tool usage based on its initialization. As shown in Figure 1, the high-level plan summarizes the query objective, analyzes the required skills, identifies relevant tools, and includes additional considerations to highlight the need for verification. The *high-level* task plan provides a global perspective on the final objective, ensuring that each subsequent sub-goal remains aligned with the original query.

Subsequently, at step t , an *action predictor* produces an action a_t that combines the planner-suggested sub-goal (e.g., “detect baseballs in the image”), the selected tool d (e.g., `Object_Detector`), and the relevant context. This *low-level* plan refines and executes each sub-goal in real time, adapting to new information or feedback at each step.

Context verification and solution summarization. After each command is executed, a *context verifier* checks whether the problem can be solved given the current context. It verifies completeness (e.g., whether all sub-goals are satisfied) and identifies any ambiguities. If the problem remains incomplete, the planner continues the next iteration of the cycle by predicting the next action a_{t+1} .

Once the verifier concludes that the query has been solved, a separate *solution summarizer* compiles the final answer from the trajectory (s_0, s_1, \dots, s_T) . This stage integrates intermediate tool outputs, traces reasoning steps, and presents a concise, user-friendly summary as the final solution.

3.3. Executor Module

Command prediction. Prior work (Lu et al., 2023; Hu et al., 2024) often expects a single language model both for planning each step (i.e., which tool to use) and for generating the corresponding executable command. This dual responsibility can overload the model and lead to errors, especially when dealing with complex or environment-specific code (Bran et al., 2023; Li et al., 2024b; Ji et al., 2024). To mitigate these issues, OctoTools introduce a *command generator* that interprets the planner’s text-based actions and produces executable code.

Given the action a_t predicted by the planner, the *command generator* (powered by a language model) creates a low-level command o_t in the form of an executable Python script, which calls the tool d_t with necessary inputs and performs any required data preparation. This step bridges the abstract action specified in a_t and the concrete tool call. By separating decision-making from code generation, each component of the system can concentrate on its specialized role.

Command execution. Once an executable command is generated, it must be run in an environment that may involve dependencies, external libraries, or resource access (e.g., file

systems). Directly coupling execution with planning poses security and maintainability challenges, especially if the planner is not capable of managing code execution details.

In OctoTools, an *command executor* runs the generated command o_t in a Python environment, obtaining a result r_t . This may include tool outputs, logs, or error messages. The executor then adds the current context of this step $s_t := (a_t, o_t, r_t)$ to the agent’s current trajectory $(s_0, s_1, \dots, s_{t-1})$. The trajectory preserves a clear history of the actions taken, the code produced, and the results obtained.

3.4. Task-specific Toolset Optimization

The OctoTools toolbox contains a diverse set of tools covering different modalities and skills. By leveraging structured tool cards and robust planning capabilities, OctoTools demonstrates strong generality when all available tools are enabled across different tasks (see §3.4). However, when a small set of validation examples are available for a task, configuring a *task-specific* subset of tools can further enhance efficiency and effectiveness.

To this end, we propose an automated algorithm to optimize the toolset configuration for each task. Given n available tools in the toolbox, the total number of possible subsets is $O(2^n)$, which is prohibitively large. To make this tractable, we employ a greedy search strategy that reduces the complexity to $O(n)$. Our approach proceeds in three stages.

Algorithm 1 Task-specific Toolset Optimization

```

1: Input: Toolbox  $\mathcal{D} = \{d_i\}_{i=1}^n$ , base toolset  $\mathcal{D}_{\text{base}}$ 
2: Output: Optimized toolset  $\mathcal{D}^*$ 
3: # Stage 1: Baseline setup
4:  $\text{Acc}_{\text{baseline}} \leftarrow \text{Acc}(\mathcal{D}_{\text{base}})$ 
5: # Stage 2: Individual tool evaluation
6: for each  $d_i$  in  $\mathcal{D}$  such that  $d_i \notin \mathcal{D}_{\text{base}}$  do
7:    $\mathcal{D}_i \leftarrow \mathcal{D}_{\text{base}} \cup \{d_i\}$ 
8:    $\text{Acc}_i \leftarrow \text{Acc}(\mathcal{D}_i)$ 
9:    $\Delta_{d_i} \leftarrow \text{Acc}_i - \text{Acc}_{\text{baseline}}$ 
10:  if  $\Delta_{d_i} > 0$  then
11:     $\mathcal{D}_{\text{beneficial}} \leftarrow \mathcal{D}_{\text{beneficial}} \cup \{d_i\}$ 
12:  end if
13: end for
14: # Stage 3: Select optimized toolset
15:  $\mathcal{D}^* \leftarrow \mathcal{D}_{\text{beneficial}} \cup \mathcal{D}_{\text{base}}$ 
16: Return  $\mathcal{D}^*$ 
    
```

Stage 1: Baseline setup. We first establish a baseline performance by enabling the base toolset in the toolbox, denoted $\mathcal{D}_{\text{base}}$. This base set represents a minimal starting toolset, which can be pre-defined by the user.

Stage 2: Individual tool evaluation. Next, we measure each candidate tool d_i by enabling it alongside the base toolset. For each d_i , we form an augmented subset $\mathcal{D}_i = \mathcal{D}_{\text{base}} \cup \{d_i\}$ and compute the accuracy difference on the set of validation examples

$$\Delta_{d_i} = \text{Acc}(\mathcal{D}_i) - \text{Acc}(\mathcal{D}_{\text{base}}). \quad (1)$$

If $\Delta_{d_i} > 0$, we consider the tool d_i beneficial for the target task.

Stage 3: Optimized toolset selection. Finally, we aggregate all tools that yield positive improvements and combine them with the default set to form the optimized toolset:

$$\mathcal{D}^* = \mathcal{D}_{\text{base}} \cup \{d_i \mid \Delta_{d_i} > 0\}. \quad (2)$$

This set contains all tools that individually demonstrate a performance gain over the baseline, ensuring a customized yet efficient configuration for the downstream task. While this selection does not guarantee global optima, we observe overall improvements over simply using all tools (see §5.2).

4. Experiments

4.1. Experimental Setups

To demonstrate the generality of our OctoTools framework, we conduct comprehensive evaluations on 16 diverse benchmarks spanning two modalities, five domains, and four reasoning types, as shown in Table 1. These benchmarks encompass a wide range of complex reasoning tasks, including visual understanding, numerical calculation, knowledge retrieval, and multi-step reasoning.

For general visual reasoning, we include AlgoPuzzleVQA (Ghosal et al., 2024), Hallusion-VD (Guan et al., 2024), PuzzleVQA (Chia et al., 2024), and VQA 2.0 (Goyal et al., 2017). For mathematical reasoning, we use Game of 24 (Lile, 2024), Omni-MATH (Gao et al., 2024), CLEVR-Math (Lindström & Abraham, 2022), and MathVista (Lu et al., 2024b). For scientific reasoning, we adopt GPQA (Rein et al., 2023), MMLU-Pro (Wang et al., 2024b), and SciFIBench (Roberts et al., 2024). To evaluate models in specialized medical domains, we test on MedQA (Jin et al., 2021), PathCLS (Sun et al., 2025), PathVQA (He et al., 2020), and SLAKE (Liu et al., 2021). Additionally, we incorporate GAIA-Text, a textual subset of the GAIA (Mialon et al., 2023) benchmark designed to evaluate agentic frameworks with tool-calling capabilities. See §A.1 for additional details of these benchmarks.

For each benchmark, we sampled 100 examples to construct a validation set for toolset optimization (§3.4) and ablation studies (§5), and set aside a held-out test set of 200 examples for final evaluation. For benchmarks with fewer than 300 total samples, the test set consisted of all remaining samples

not used for validation. To mitigate randomness, we report the average accuracy with standard deviation across three trials for all experiments, including ablation studies. Details on experimental setups are provided in §A.

We created a diverse array of tools in the toolbox for our experiments. The base toolset consists of `Generalist_Solution_Generator`, a tool built on top of the base LLM that takes as input a specialized prompt generated by the executor, such as “Describe the image in detail” or “Generate stepwise solutions for the math problem”. This tool allows for general step-by-step reasoning without requiring external tools but lacks domain specificity. The toolbox also includes image perception tools such as `Image_Captioner`, web search APIs like `Google_Search`, a code generator `Python_Code_Generator`, and specialized tools like `Path_Generalist_Classifier` for pathology image classification. Among them, `Relevant_Patch_Zoomer` is a unique tool that takes a textual query and returns zoomed-in quarter patches, which provide localized details for fine-grained visual reasoning scenarios. See §A.2 and §D for more details.

4.2. Main Results

We compare the performance of our framework after toolset optimization (denoted as OctoTools) against three baselines: (1) zero-shot, where the base LLM directly answers queries without additional prompting, (2) chain-of-thought (CoT), where the base LLM is prompted to “Think step by step” to generate step-by-step reasoning, and (3) OctoTools_{base}, which uses only the base tool without external integrations. Unless otherwise specified, all results presented below use gpt-4o-2024-08-06 as the base model.

Table 1 and Figure 2 summarize performance across all 16 benchmarks. OctoTools achieves consistent gains, outperforming zero-shot and CoT baselines by 9.3% and 7.7% on average, respectively. OctoTools_{base} also demonstrates improvements over zero-shot (4.2%) and CoT (2.6%), indicating that our framework’s step-by-step reasoning contributes significantly to performance, independent of external tool integration. Figure 16 visualizes gains over zero-shot. Detailed analyses of these gains are presented in §4.4.

4.3. Comparisons with Other Agent Frameworks

In addition, we compare three commonly used general AI agent frameworks: GPT-Functions (OpenAI, 2023a), LangChain (LangChain, 2024), and AutoGen (AutoGen, 2024). GPT-Functions enables GPT-4o to call user-specified tools via function calling. LangChain is a framework providing multi-agent collaboration, long-term memory, and tool usage. AutoGen is a recent agentic framework that creates multiple autonomous agents with tool usage.

OctoTools: An Agentic Framework with Extensible Tools for Complex Reasoning









Datasets	Modality	Domain					0-shot	CoT	OctoTools _{base}	OctoTools	Δ (0-shot)	Δ (CoT)
AlgoPuzzleVQA	Vision	General	✓			✓	41.3 \pm 0.3	42.7 \pm 1.0	44.0 \pm 0.9	48.7 \pm 0.3	+7.4	+6.0
Hallusion-VD	Vision	General	✓				52.0 \pm 1.0	53.3 \pm 2.1	59.0 \pm 0.0	63.3 \pm 2.9	+11.3	+10.0
PuzzleVQA	Vision	General	✓			✓	52.2 \pm 1.0	54.0 \pm 1.3	59.3 \pm 0.8	61.0 \pm 0.5	+8.8	+7.0
VQA 2.0	Vision	General	✓			✓	50.3 \pm 1.0	48.7 \pm 0.3	47.2 \pm 0.8	54.5 \pm 0.0	+4.2	+5.8
Game of 24	Text	Mathematical		✓		✓	22.2 \pm 2.5	33.3 \pm 1.5	37.8 \pm 3.3	44.7 \pm 2.8	+22.5	+11.4
Omni-MATH	Text	Mathematical		✓	✓	✓	27.0 \pm 0.0	29.3 \pm 1.3	30.2 \pm 0.6	32.2 \pm 0.8	+5.2	+2.9
CLEVR-Math	Vision	Mathematical	✓	✓			64.5 \pm 3.0	75.2 \pm 1.5	68.8 \pm 0.8	79.0 \pm 0.9	+14.5	+3.8
MathVista	Vision	Mathematical	✓	✓	✓	✓	59.3 \pm 0.8	59.5 \pm 1.5	63.0 \pm 1.3	64.3 \pm 1.0	+5.0	+4.8
GPQA	Text	Scientific		✓	✓	✓	53.7 \pm 1.9	52.3 \pm 2.0	53.7 \pm 2.5	54.7 \pm 1.3	+1.0	+2.4
MMLU-Pro	Text	Scientific			✓	✓	71.7 \pm 0.3	70.3 \pm 0.6	71.5 \pm 1.3	73.7 \pm 1.3	+2.0	+3.4
SciFIBench	Vision	Scientific	✓		✓		72.5 \pm 0.0	75.0 \pm 0.9	77.3 \pm 0.8	78.3 \pm 0.6	+5.8	+3.3
MedQA	Text	Medical			✓	✓	84.5 \pm 1.0	84.8 \pm 0.6	92.8 \pm 0.6	91.5 \pm 1.8	+7.0	+6.7
PathCLS	Vision	Medical	✓		✓	✓	36.0 \pm 0.9	37.5 \pm 1.8	37.0 \pm 1.8	58.2 \pm 1.3	+22.2	+20.7
PathVQA	Vision	Medical	✓		✓	✓	32.0 \pm 1.8	27.8 \pm 1.8	43.5 \pm 2.6	49.2 \pm 1.2	+17.2	+21.4
SLAKE	Vision	Medical	✓		✓	✓	59.3 \pm 1.0	60.3 \pm 0.6	59.2 \pm 1.8	63.8 \pm 1.4	+4.5	+3.5
GAIA-Text	Text	Agentic		✓	✓	✓	8.7 \pm 0.8	8.4 \pm 0.5	9.7 \pm 0.9	18.4 \pm 1.2	+9.7	+10.0
Average (%)	-	-	-	-	-	-	49.2	50.8	53.4	58.5	+9.3	+7.7

Table 1. Main results across 16 benchmarks spanning different modalities, domains, and required reasoning skills: visual understanding () , numerical calculation () , knowledge retrieval () , and multi-step reasoning () . OctoTools_{base} uses only the base tool (Generalist_Solution_Generator), while OctoTools uses the optimal toolset. Performance gains (Δ) are computed for OctoTools relative to both zero-shot (0-shot) and chain-of-thought (CoT) baselines, with OctoTools achieving 58.5% average accuracy and improvements of 9.3% and 7.7% respectively. All results show average accuracy and standard deviation (gray) over three trials.

Datasets	AutoGen	GPT-Functions	LangChain	OctoTools
AlgoPuzzleVQA	44.0 \pm 1.0	44.5 \pm 0.5	42.7 \pm 2.8	48.7 \pm 0.3
Hallusion-VD	52.7 \pm 4.7	57.0 \pm 1.7	53.7 \pm 3.1	63.3 \pm 2.9
Puzzle VQA	40.0 \pm 2.3	52.5 \pm 2.8	53.5 \pm 7.8	61.0 \pm 0.5
VQA 2.0	46.0 \pm 1.0	45.5 \pm 0.9	54.0 \pm 1.0	54.5 \pm 0.0
Game of 24	24.2 \pm 2.4	34.5 \pm 2.3	18.3 \pm 4.1	44.7 \pm 2.8
Omni-MATH	28.5 \pm 1.3	22.8 \pm 1.8	29.7 \pm 0.6	32.2 \pm 0.8
CLEVR-Math	69.5 \pm 3.9	71.2 \pm 1.0	69.2 \pm 4.6	79.0 \pm 0.9
MathVista	24.7 \pm 2.5	54.5 \pm 2.0	55.7 \pm 0.3	64.3 \pm 1.0
GPQA	48.7 \pm 2.9	45.8 \pm 2.6	52.2 \pm 1.2	54.7 \pm 1.3
MMLU-Pro	65.0 \pm 2.5	65.8 \pm 2.4	70.3 \pm 1.2	73.7 \pm 1.3
SciFIBench	70.0 \pm 2.2	68.8 \pm 3.2	77.0 \pm 0.5	78.3 \pm 0.6
MedQA	83.7 \pm 2.8	84.8 \pm 0.3	73.7 \pm 0.6	91.5 \pm 1.8
PathCLS	58.0 \pm 1.3	58.2 \pm 0.6	56.3 \pm 1.3	58.2 \pm 1.3
PathVQA	42.7 \pm 0.8	42.8 \pm 2.3	45.7 \pm 4.4	49.2 \pm 1.2
SLAKE	62.2 \pm 1.8	59.7 \pm 1.9	59.3 \pm 0.8	63.8 \pm 1.4
GAIA-Text	6.3 \pm 0.8	7.9 \pm 0.8	7.6 \pm 1.2	18.4 \pm 1.2
Average (%)	47.9	51.0	51.2	58.5

Table 2. Comparison with other agent frameworks using the same underlying toolbox. OctoTools achieves superior performance with an average accuracy of 58.5%, outperforming the next best baseline by 7.3%. Results are averaged over three trials.

For a standardized comparison of each system’s ability to plan and use tools over multiple steps, we configure all agent frameworks, including OctoTools, to use the same underlying model (GPT-4o) and hyperparameters. They share the same toolset (with the same implementations of tools), a maximum reasoning budget of 10 steps, and a time budget of 300 seconds. See §A for further details.

Table 2 and Figure 10 show the performance of OctoTools compared to other agent frameworks across all 16

benchmarks. Overall, OctoTools outperforms other agent frameworks, achieving an average accuracy gain of 10.6% over AutoGen, 7.5% over GPT-Functions, and 7.3% over LangChain. See §4.4 for detailed analysis.

4.4. Analysis of Performance Gains

Our agent framework includes three aspects for improvement in complex reasoning: *task planning*, *external tool calling*, and *multi-step problem solving*. We include an analysis to disentangle added benefits of each of these aspects.

Tool usage distribution. Figure 4 (a) shows the tool usage distribution for OctoTools and the agent baselines, averaged over 16 tasks. OctoTools takes advantages of *task planning* by calling both a base tool to decompose the query into subtasks (32.2%) and external specialized tools (67.8%) for reasoning skills such as fine-grained image understanding, domain knowledge access, precise calculation, and other domain-specific tasks. In contrast, the other agent baselines exhibit limitations in task planning for external tool usage, with external tool usage rates of 10.6% for AutoGen, 23.3% for GPT-Functions, and 10.7% for LangChain.

Figure 4 (b) illustrates the tool usage distribution of OctoTools across the 16 tasks. OctoTools adapts to each task by selecting the most suitable tools. For instance, OctoTools employs five different tools to address the diverse challenges in GAIA-Text. In MathVista, a benchmark featuring multiple diagram-based and mathematical reasoning problems, it uses Relevant_Patch_Zoomer for local vision perception, Google_Search for web queries, and

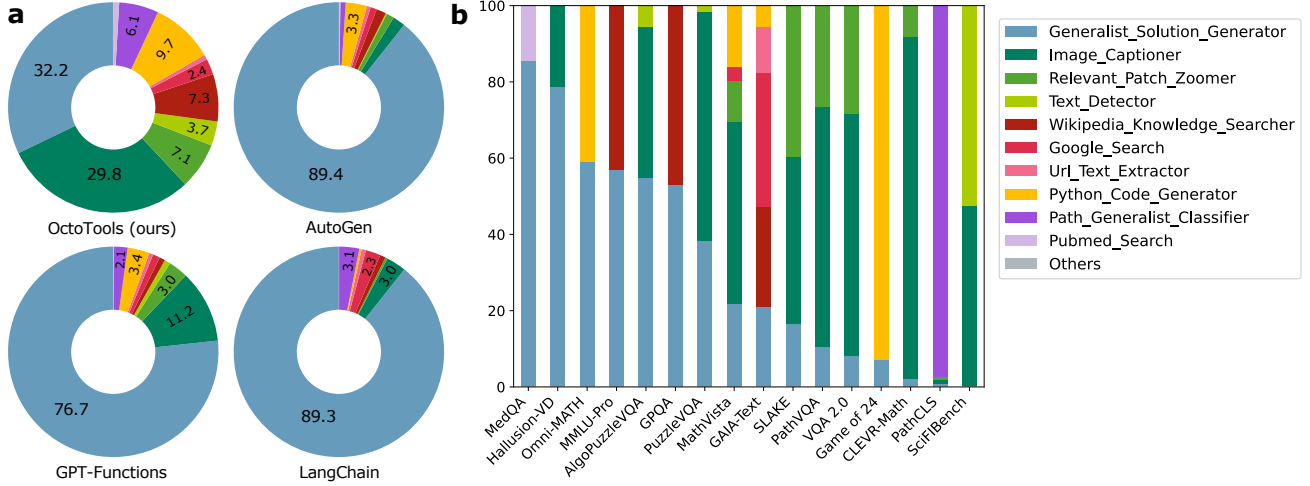


Figure 4. **a.** Tool usage distribution in our OctoTools framework and agent baselines by averaging results from 16 tasks. **b.** Tool usage distribution across 16 tasks in OctoTools. OctoTools takes advantage of different external tools to address task-specific challenges.

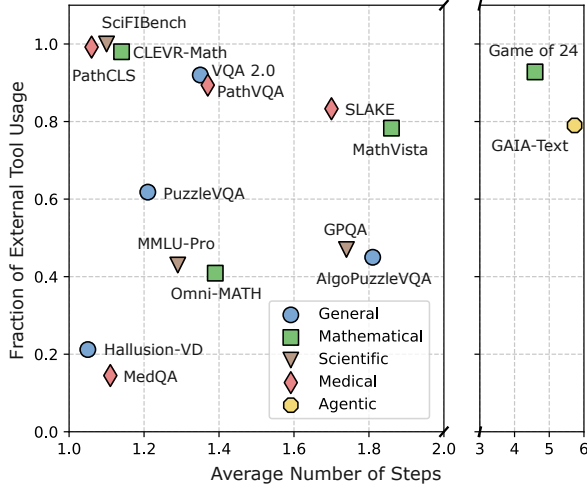


Figure 5. Benchmark distribution across average number of steps and fraction of external tool usage (tools that exclude the base tool Generalist_Solution_Generator) in OctoTools.

Python_Code_Generator for precise calculations.

External tool usage v.s. multiple steps. Figure 5 shows how tasks are distributed based on the average number of steps and the fraction of external tool usage in OctoTools. Tasks with a high average number of steps indicates that *multi-step problem solving* is valuable, while a high proportion of external tool usage highlight the benefits of *external tool calling*. Notably, two tasks score highly on both dimensions: Game of 24, which frequently calls Python_Code_Generator to explore and verify arithmetic expressions that evaluate to 24, and GAIA-Text, which requires multiple tools to address a variety of underlying skills.

Gains from decompositions v.s. tool usage. To assess the performance gains from *tool calling*, we compute for each benchmark the difference between OctoTools, *i.e.*, with external tools, and OctoTools_{base}, *i.e.*, without external tools: $\Delta_{\text{tools}} = \text{Acc}_{\text{OctoTools}} - \text{Acc}_{\text{OctoTools}_{\text{base}}}$. Similarly, to assess the performance gains from *decomposing* the problem into *multiple steps*, we compute the difference between OctoTools_{base} and the 0-shot baseline: $\Delta_{\text{decom}} = \text{Acc}_{\text{OctoTools}_{\text{base}}} - \text{Acc}_{0\text{-shot}}$.

For each benchmark, the performance gains from these two dimensions are visualized in Figure 6. We observe that the tasks can be broadly categorized into three groups: (1) Tasks far above the $y = x$ diagonal benefit more from step decomposition, and a specialized tool is either not needed or not available. (2) Tasks far below the diagonal benefit more from one or more specialized tools that are well-suited for the task, while multi-step reasoning is less critical. (3) Tasks along the diagonal benefit from both aspects. This analysis highlights that AI agent frameworks can improve performance in different ways depending on the specific skills demanded by each task, and that OctoTools is a versatile framework for achieving such improvements across a diverse set of tasks.

5. Ablation Study

This section explores several factors that affect OctoTools’s performance, using a validation set of 100 samples.

5.1. Number of Maximum Allowed Steps

We explore how the behavior and performance of OctoTools change under different maximum step limits. We report the average over three trials, with results summarized in

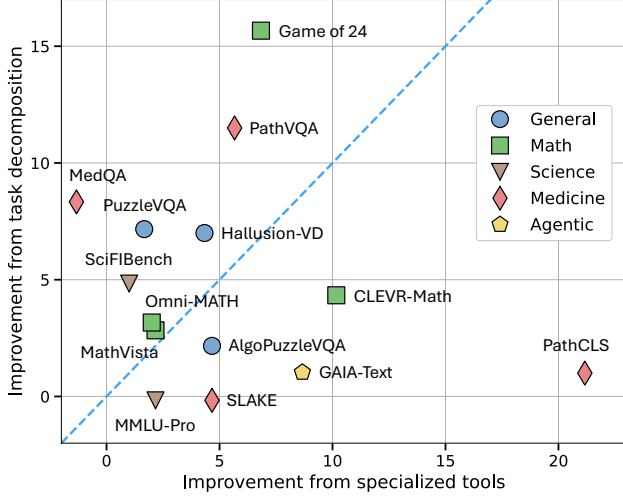


Figure 6. Benchmark distribution across two dimensions. Tasks that show high improvement from task decomposition likely require multi-step reasoning, while tasks that show high improvement from specialized tools likely require specialized skills.

Figure 7. Overall, performance tends to improve as the maximum number of steps increases, highlighting the benefit of longer chains of multi-step reasoning. Detailed results and analysis for individual tasks are provided in §B. Additionally, we found that averaged across 16 tasks, running OctoTools with GPT-4o typically takes less than \$5 for 100 queries with 10 maximum allowed steps.

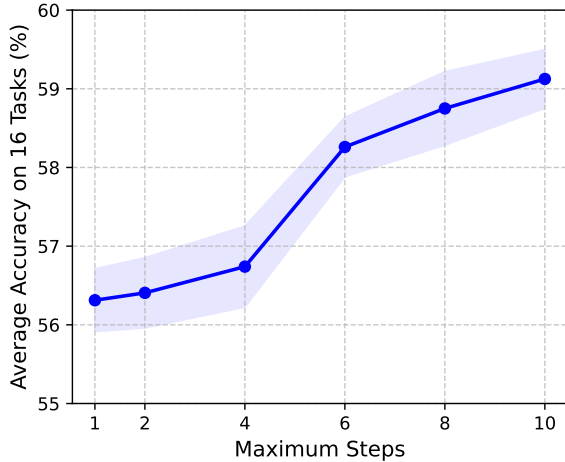


Figure 7. Average accuracy across 16 benchmarks with respect to maximum allowed reasoning steps in OctoTools.

5.2. Toolset Optimization

To investigate the benefits of our toolset optimization algorithm (presented in §3.4 and Algorithm 1), we evaluate OctoTools under three toolset strategies:

(1) OctoTools_{base}, i.e., the setup with only the base `Generalist_Solution_Generator` tool, (2) OctoTools with the *full* toolset, where all possible tools are enabled, and (3) OctoTools with the *optimized* toolset. Figure 8 presents results across all 16 tasks and different reasoning categories.

On average, the optimized toolset achieves 58.9% accuracy, outperforming the OctoTools_{base} (53.9%) by 5.0%. Meanwhile, simply enabling all tools yields 57.4% accuracy, which still outperforms OctoTools_{base} by 3.5%. Although toolset optimization provides the higher overall performance, our framework is sufficiently versatile that enabling all tools also offers a substantial improvement, helping ensure generalization as we scale up the toolset.

While enabling all available tools might appear advantageous, prior work (Lumer, 2024; Fore et al., 2024; Paramanayakam et al., 2024) has shown that it can sometimes degrade performance. Our findings suggest that external feedback (e.g., performance on a validation set or expert insights) helps identify when a certain tool improves performance. Additionally, selectively enabling tools avoids unnecessary overhead and improves efficiency. A promising next step is to explore a query-based approach that dynamically refines the toolset for each query.

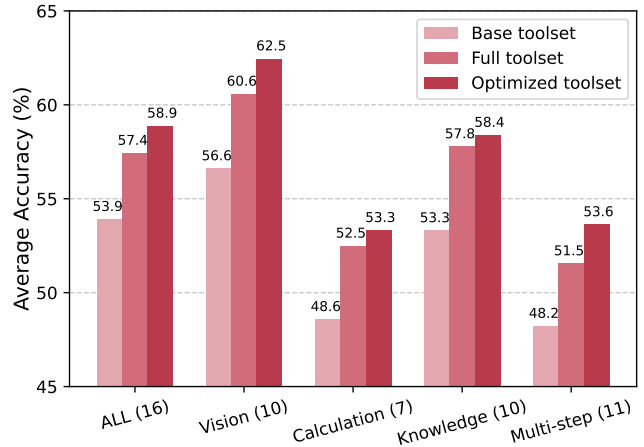


Figure 8. Performance under three toolset strategies in OctoTools across all 16 tasks and various categories (the number in parentheses indicates the number of tasks in each category).

5.3. Using a Weaker LLM

Additionally, we evaluate our framework using a weaker LLM (GPT-4o-mini) as the base engine. Similar to the analysis with GPT-4o, we compare OctoTools against the zero-shot and CoT baselines on the validation sets of 16 benchmarks. As shown in Figure 9, OctoTools generally outperforms the baselines across all 16 tasks and various categories. Task-specific analyses are provided in §B.

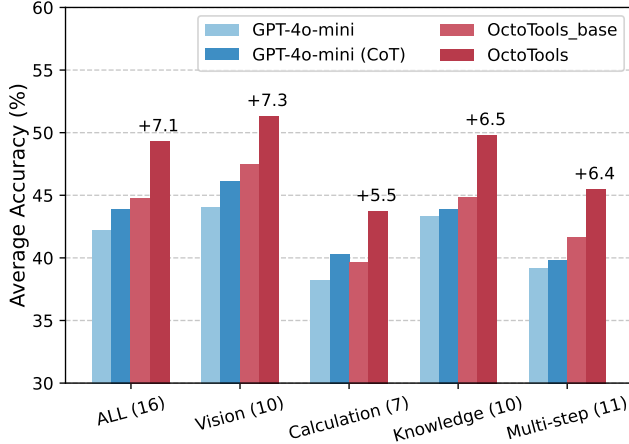


Figure 9. Performance of OctoTools on 16 tasks and various categories using a weaker LLM, GPT-4o-mini, as the base engine. OctoTools_{base} is the configuration in which only the base Generalist_Solution_Generator tool is enabled. The number in parentheses indicates # tasks in each category.

6. Conclusion

In this paper, we introduced OctoTools, a training-free, extensible agentic framework for complex reasoning. OctoTools employs standardized *tool cards* to facilitate seamless integration of diverse tools and a dedicated planner-executor workflow that separates high-level planning over multiple steps from low-level planning and command generation within each step. Through extensive experiments on 16 diverse benchmarks, OctoTools consistently outperforms baselines, achieving average accuracy gains of up to 9.3% over GPT-4o and up to 10.6% over strong agentic frameworks. Our in-depth analysis shows that OctoTools’ improvements stem from dynamic task planning, effective tool usage, and multi-step problem decomposition.

Ablation studies highlight the benefits of allowing more step, refining the toolset, and demonstrate the robustness when deployed with a weaker LLM. By streamlining the integration of new or specialized modules through tool cards, OctoTools readily adapts to a broad range of tasks. We believe our findings open new ecosystems for building next-generation AI agents that are more transparent, modular, and effective at solving real-world problems. Future work includes test-time inference at the query level, extending multi-agent collaboration, and exploring specialized domains.

Acknowledgments

This work is partially supported by the Hoffman-Yee Research Grants program at Stanford HAI and Chan Zuckerberg Initiative. We thank members of the Zou group for helpful discussions and insightful feedback.

References

- AutoGen. Autogen. <https://github.com/microsoft/autogen>, 2024.
- Bi, Z., Han, K., Liu, C., Tang, Y., and Wang, Y. Forest-of-thought: Scaling test-time compute for enhancing llm reasoning. *arXiv preprint arXiv:2412.09078*, 2024.
- Bran, A. M., Cox, S., Schilter, O., Baldassari, C., White, A. D., and Schwaller, P. Chemcrow: Augmenting large-language models with chemistry tools. *arXiv preprint arXiv:2304.05376*, 2023.
- Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners. *Advances in Neural Information Processing Systems (NeurIPS)*, 33:1877–1901, 2020.
- Caron, M., Touvron, H., Misra, I., Jégou, H., Mairal, J., Bojanowski, P., and Joulin, A. Emerging properties in self-supervised vision transformers. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 9650–9660, 2021.
- Chen, W.-G., Spiridonova, I., Yang, J., Gao, J., and Li, C. Llava-interactive: An all-in-one demo for image chat, segmentation, generation and editing. *arXiv preprint arXiv:2311.00571*, 2023.
- Chia, Y. K., Han, V. T. Y., Ghosal, D., Bing, L., and Poria, S. PuzzleVQA: Diagnosing multimodal reasoning challenges of language models with abstract visual patterns. *arXiv preprint arXiv:2403.13315*, 2024.
- Chowdhery, A., Narang, S., Devlin, J., Bosma, M., Mishra, G., Roberts, A., Barham, P., Chung, H. W., Sutton, C., Gehrmann, S., et al. PaLM: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*, 2022.
- Cobbe, K., Kosaraju, V., Bavarian, M., Chen, M., Jun, H., Kaiser, L., Plappert, M., Tworek, J., Hilton, J., Nakano, R., et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.
- Fore, M., Singh, S., and Stamoulis, D. Geckopt: Llm system efficiency via intent-based tool selection. In *Proceedings of the Great Lakes Symposium on VLSI 2024*, pp. 353–354, 2024.
- Gao, B., Song, F., Yang, Z., Cai, Z., Miao, Y., Dong, Q., Li, L., Ma, C., Chen, L., Xu, R., et al. Omni-MATH: A universal olympiad level mathematic benchmark for large language models. *arXiv preprint arXiv:2410.07985*, 2024.