arXiv:2506.12508v3 [cs.AI] 13 Aug 2025

AgentOrchestra: A Hierarchical Multi-Agent Framework for General-Purpose Task Solving

Wentao Zhang[1,†], **Liang Zeng**[1,†], **Yuzhen Xiao**[1], **Yongcong Li**[1], **Ce Cui**[1], **Yilei Zhao**[2], **Rui Hu**[1], **Yang Liu**[1], **Yahui Zhou**[1,‡], **Bo An**[2,1,‡]

[1]Skywork AI
[2]Nanyang Technological University

Abstract

Recent advances in large language model (LLM)-based agent systems have demonstrated remarkable capabilities in solving both general-purpose and highly complex tasks. However, most current models lack mechanisms for coordinating specialized agents and have limited ability to generalize to new or diverse domains. To this end, we introduce `AgentOrchestra`, a hierarchical multi-agent framework for general-purpose task solving that integrates high-level planning with modular agent collaboration. Drawing inspiration from a conductor orchestrating a symphony, and grounded in the principles of *extensibility*, *multimodality*, *modularity*, and *coordination*, `AgentOrchestra` features a central planning agent that decomposes complex objectives and delegates sub-tasks to a team of specialized agents. Each sub-agent is equipped with general programming and analytical tools, as well as abilities to tackle a wide range of real-world specific tasks, including data analysis, file operations, web navigation, and interactive reasoning in dynamic multimodal environments. Notably, `AgentOrchestra` introduces an MCP Manager Agent that enables intelligent evolution through dynamic tool creation, retrieval, and reuse mechanisms, significantly enhancing the system's adaptability and scalability. `AgentOrchestra` supports flexible orchestration through explicit sub-goal formulation, inter-agent communication, and adaptive role allocation. We evaluate the framework on three widely used benchmarks for assessing LLM-based agent systems. Experimental results show that `AgentOrchestra` consistently outperforms flat-agent and monolithic baselines in terms of task success rate and adaptability. On the GAIA benchmark testing dataset, `AgentOrchestra` achieves an average score of 83.39%, ranking among the top general-purpose agents. These results highlight the effectiveness of hierarchical organization and role specialization in building scalable and general-purpose LLM-based agent systems.

1 Introduction

Recent advances in Large Language Models (LLMs) and Large Multimodal Models (LMMs) have driven a shift from simple dialogue models [6, 11, 1, 32, 34, 5] to systems capable of performing sophisticated reasoning [12, 23, 4, 8, 42, 33], enabling progress from answering straightforward questions to addressing complex, multi-step queries. However, current LLMs remain largely disconnected from real-world environments due to the absence of interactive tool integration, limiting their capacity to perform grounded, general-purpose, and complex tasks. Overcoming this limitation

† Equal contribution
‡ Correspondence to Wentao Zhang: zhangwent963@gmail.com, Yahui Zhou, Bo An: boan@ntu.edu.sg
[1]Website: `https://skyworkai.github.io/DeepResearchAgent/`

Preprint. Under review.

requires augmenting LLMs with executable tools as well as perceptual and action interfaces, thereby transforming LLMs into interactive agents capable of perceiving, acting, and reasoning across both virtual and real-world environments.

LLM-based agents have demonstrated strong capabilities in real-world tasks such as browser use [24, 19], computer use [24, 2, 27], code execution [36], game playing [35, 31], and research assistance [22, 7, 41]. In parallel, a new wave of generalist agents has emerged, characterized by their ability to operate across various domains rather than being restricted to single-task settings. Systems such as Manus [30] and open-source agent frameworks including OpenHands [37], OpenManus [15], and smolagents [29] exemplify this trend by demonstrating unified perception, reasoning, and tool-augmented action. Furthermore, the development of tool-use interfaces, including OpenAI's Function Calling [21] and Anthropic's Model-Context Protocol (MCP) [3], has enabled seamless tool integration and broadened the range of tasks. Collectively, these advances signify a transition from narrow, single-domain specialization toward integrated, general-purpose intelligence.

Despite recent advances, current generalist agents have yet to realize genuine general-purpose intelligence, particularly when confronted with complex, real-world multimodal tasks. They continue to face significant difficulties in robust perception, reasoning, and action across heterogeneous modalities and dynamic environments. Consequently, achieving truly general-purpose agent systems involves addressing several fundamental challenges: i) **Limited Generalization and Transferability**: Most existing agent frameworks are narrowly specialized for specific domains or tasks, exhibiting limited capacity to generalize across heterogeneous environments or adapt to novel, unseen scenarios. Such limitations significantly constrain their deployment in open-ended or real-world contexts. ii) **Insufficient Multimodal Perception and Reasoning**: Current agents often struggle to effectively perceive, align, and reason over diverse modalities such as text, image, audio, video, and structured data, thereby impeding their performance on complex tasks that require integrated multimodal understanding and reasoning. iii) **Limited Scalability and Maintainability**: Many prevailing agent architectures lack modularity and extensibility, complicating the incorporation of new models and tools, as well as adaptation to emerging application scenarios. This deficiency hinders the development of scalable and sustainable agent ecosystems. iv) **Inefficient Multi-Agent Collaboration and Communication**: Existing approaches rarely support efficient collaboration and communication among multiple agents, limiting their capacity for dynamic role allocation, coordinated planning, and effective teamwork on compositional or large-scale tasks.

To address these challenges, we propose `AgentOrchestra`, a hierarchical multi-agent framework for general-purpose task solving. `AgentOrchestra` features a top-level planning agent that coordinates specialized sub-agents equipped with domain-specific tools. This design enables flexible task decomposition, extensible collaboration, and unified handling of multimodal inputs, making the system well-suited for real-world applications. Contributions of this work are as follows:

- `AgentOrchestra` is a hierarchical framework that integrates a top-level planning agent with modular sub-agents. New capabilities can be seamlessly added by incorporating additional specialized sub-agents, making the system highly extensible and adaptable to diverse domains.
- Guided by the principles of extensibility, multimodality, modularity, and coordination, `AgentOrchestra` supports unified multimodal tool integration, flexible component combination, and efficient hierarchical collaboration. In addition, the framework possesses self-evolution capabilities through dynamic tool creation, intelligent retrieval, and automated reuse, further enhancing adaptability and scalability.
- Extensive experiments show that `AgentOrchestra` consistently outperforms existing agent baselines in generalization, multimodal understanding, and collaborative problem solving across diverse real-world benchmark tasks. Notably, on the GAIA benchmark testing dataset, `AgentOrchestra` achieves an average score of 83.39%, placing it among the top general-purpose agents.

2 Related Work

2.1 Tool-Augmented Agent Systems

The integration of tools with LLMs marks a paradigm shift in AI agent development. Compared to traditional rule-based agents, tool-augmented LLM agents exhibit greater flexibility, cross-domain reasoning, and natural language interaction [14]. These agents have shown strong capabilities in

web browsing [24, 19], computer operation [2, 27], code execution [36], and game playing [35, 31]. Standardized tool interfaces, such as OpenAI's Function Calling and Anthropic's Model-Context Protocol (MCP), have further streamlined tool integration and expanded the range of executable tasks [21, 3]. Recent work on frameworks like ToolMaker [40] enables automatic transformation of code-based research into LLM-compatible tools, reducing reliance on manual tool development.

2.2 General-Purpose Agent Frameworks

The rise of generalist agents and open-source frameworks, such as Manus [30], OpenHands [37], OpenManus [15], and smolagents [29], has advanced unified perception, reasoning, and tool-augmented action beyond domain-specific applications. Recent work like Alita [28] introduces a novel approach to generalist agents through minimal predefinition and maximal self-evolution, enabling agents to autonomously construct and refine external capabilities by generating task-related Model Context Protocol (MCP) tools from open source. These frameworks target broader, general-purpose intelligence across diverse tasks. Comprehensive surveys [16] have documented this evolution, highlighting the shift from task-specific agents to more flexible, general-purpose systems.

2.3 Multi-Agent Collaboration Systems

The field of multi-agent systems has seen substantial growth, with research focusing on both task-oriented communication (collaborative exchanges and adversarial interactions) and open-ended conversations. Recent systems such as MetaGPT [9] demonstrate how multiple specialized agents can coordinate to solve complex problems beyond the reach of single agents. Li et al.[13] explored personal LLM agents, examining their capabilities, efficiency, and security in collaborative settings. Ni et al.[20] introduced Collaborative Reasoner, a self-improving social agent framework that leverages synthetic conversations to enhance its reasoning capabilities. While these advances highlight the adaptability and promise of multi-agent collaboration, many existing approaches still lack mechanisms for efficient communication, dynamic role allocation, and coordinated teamwork in large-scale tasks. In this work, we address these challenges by proposing a hierarchical multi-agent framework designed to achieve more efficient execution and collaboration compared to existing agent and multi-agent systems.

3 AgentOrchestra

`AgentOrchestra` is a hierarchical multi-agent framework designed to systematically address the key challenges of generalization, multimodal reasoning, scalability, and collaboration in complex task-solving environments. The framework adopts a two-tier architecture, where a top-level planning agent decomposes tasks and coordinates modular sub-agents responsible for domain-specific processing and multimodal reasoning. This design enables flexible composition, seamless collaboration, and robust adaptation across diverse domains, while naturally scaling to complex and heterogeneous tasks through the dynamic expansion of specialized sub-agents. Section 3.1 introduces the core design principles of our framework. Section 3.2 details the implementation of the planning agent, and Section 3.3 discusses the architecture and interaction patterns of specialized sub-agents.

3.1 Agent Design Principles

Agent. An agent is an autonomous computational entity that perceives and interprets the environment, maintains a history of actions and observations, and flexibly generates actions to accomplish a wide variety of user-specified tasks across diverse domains.

Environment. The environment represents the external context and resources within which the agent operates. This includes the computational infrastructure (operating system, file system, network access), available tools and APIs, user-provided files and data, and any external systems or services the agent needs to interact with. The environment provides the interface through which the agent can execute actions, access information, and observe the results of its operations. Our framework supports diverse environments ranging from local development setups to cloud-based platforms, enabling agents to adapt to different deployment scenarios and resource constraints.

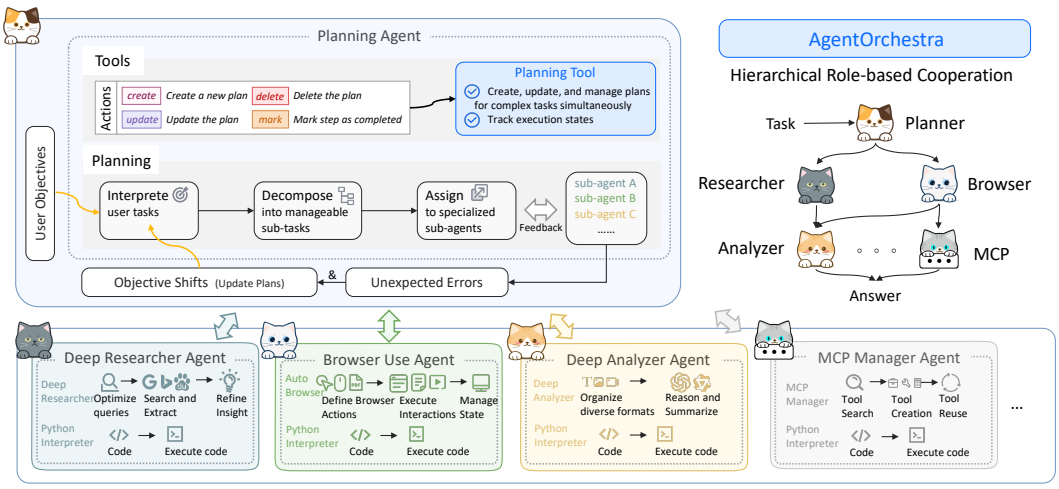

Figure 1: Architecture of `AgentOrchestra`.

Model. LLMs are the core drivers of our framework. To enhance flexibility and adaptability, we introduce a unified LLM abstraction layer that supports both leading commercial models (e.g., GPT-4.1, Claude-4-Sonnet) and local open-source models (e.g., Qwen2.5). This design enables agents to dynamically select and switch between different LLMs during task execution, aligning each model's unique strengths with the requirements of specific sub-tasks and environments. By supporting seamless integration of new models as they become available, our modular architecture ensures long-term extensibility and consistently strong performance across a wide range of applications.

Observation. An observation encompasses all information perceived by the agent at each step of execution. This includes the user's task description, any attached files, screenshots, and other relevant contextual data. The observation also maintains a complete history (including error messages) of previous steps, providing the agent with a comprehensive view of the ongoing process. Each step's observation is stored in the agent's memory, enabling effective reasoning, error recovery, and context-aware decision making throughout the task-solving process.

Action. In our framework, actions are operationalized through a set of pre-defined tools [37, 15, 29], each exposed via function calling [21, 3] interfaces that specify parameters and expected behaviors. Importantly, actions are not equivalent to tools. A single tool can support multiple actions by accepting different action parameters. For example, a planning tool may support *create*, *update*, and *delete* actions, all accessible through a unified interface with action-specific arguments. This design improves modularity and extensibility, allowing the agent to perform diverse operations with fewer, more flexible tools.

To ensure safety and system integrity, all operations with potential side effects are executed within a docker-based sandbox, such as an isolated Linux container or virtual machine. This sandboxing mechanism provides strong isolation, preventing unintended modifications or security risks to the underlying environment while allowing the agent to safely and reliably interact with external tools.

Memory. Memory serves as a fundamental component of the agent, persistently recording the complete history of agent execution. This comprehensive execution trace, which records task sequences, observations, actions, and errors, allows the agent to reason effectively, recover from errors robustly, and make context-aware decisions. At each step, the agent summarizes and updates its memory, allowing it to utilize prior context and improve its ability to correct errors and adapt to evolving task requirements.

The execution flow of an agent in our framework follows an iterative cycle: the **agent** perceives the **environment** through **observation** (including task descriptions, attached files, and execution history), stores these observations in **memory**, leverages its memory and the **model** selected from the unified LLM abstraction layer to interpret the context and select an appropriate **action** from pre-defined tools, executes the action within a sandbox environment, and records the outcomes in memory for

subsequent reasoning and adaptation. This cycle repeats until the task is completed or a termination condition is met.

3.2 Planning Agent

Planning Agent Overview. The Planning Agent serves as the central orchestrator in our hierarchical framework, dedicated to high-level reasoning, task decomposition, and adaptive planning. Rather than directly interacting with the environment or executing low-level actions, the Planning Agent interprets user objectives and systematically decomposes complex, long-horizon tasks into manageable sub-tasks. These sub-tasks are then assigned to specialized sub-agents based on their expertise and the evolving context.

A distinctive feature of the Planning Agent is its ability to maintain a global perspective throughout the execution process, aggregating feedback from sub-agents and monitoring progress toward the overall objective. This enables the agent to perform dynamic plan updates, adapting its strategy in real time in response to intermediate results, unexpected challenges, or shifting user requirements. Such closed-loop coordination is crucial for robustness and generalization across diverse domains. To ensure modularity and scalability, the Planning Agent interacts with sub-agents using a standard interface. This design conceals domain-specific details and facilitates the addition of new types of agents. By separating high-level planning from specialized task execution, our architecture avoids the limitations of monolithic agents, such as low flexibility and poor adaptation to new tasks. The core workflow is managed by a Planning Tool, which oversees task planning and execution tracking.

Planning Tool Design. The Planning Tool offers structured functionalities for creating, updating, and managing plans for complex tasks. Each plan comprises a sequence of discrete steps, each explicitly grounded in the available tools and agent capabilities. The module supports key operations such as plan creation, modification, status marking, and progress monitoring, while ensuring that every complex task is decomposed into several actionable steps, each assigned to specialized sub-agents or tool invocations. The Planning Tool maintains unique identifiers for each plan, enables concurrent management of multiple plans, and tracks execution states (e.g., not started, in progress, completed, blocked) for all steps. Importantly, the Planning Tool dynamically updates and adapts the plan in response to the evolving execution context, intermediate outcomes, and feedback from sub-agents. All agent-plan interactions are conducted via a standardized, extensible interface, facilitating seamless integration of new task types and agent capabilities.

3.3 Specialized Sub-Agents

To address real-world challenges such as comprehensive information retrieval from the internet, acquisition of domain-specific or time-sensitive expertise, statistical analysis, gaming, and computational tasks, it is essential to possess a set of foundational capabilities. These include reasoning, multimodal information processing, web browsing, and, more generally, proficiency in tool use. To further enhance these foundational abilities and adapt to diverse, complex tasks, there is a growing need to incorporate an expandable set of low-level specialized sub-agents. By continually extending the repertoire and capabilities of these sub-agents, the hierarchical multi-agent system as a whole can achieve scalable improvements, mirroring scaling laws at the agent level and enabling more robust, generalizable problem-solving.

Therefore, we further instantiate our hierarchical multi-agent framework with a set of specialized sub-agents tailored for distinct stages of complex tasks. First, a Deep Researcher Agent is responsible for extensive information retrieval, efficiently scanning and filtering web pages to identify promising sources, much like the initial human exploration of online information. Next, a Browser Use Agent enables fine-grained interactions with web content, allowing direct engagement with videos, PDFs, and various HTML elements to extract precise information. A Deep Analyzer Agent is designed to perform advanced reasoning and comprehensive analysis, integrating and interpreting information collected by previous agents to address tasks such as statistical inference, image analysis, gaming, and market analysis. Additionally, an MCP Manager Agent provides intelligent tool evolution capabilities through automated creation, dynamic retrieval, and systematic reuse of MCP tools, enabling the system to autonomously extend its capabilities in response to task-specific requirements. In addition to their dedicated specialized tools, each sub-agent is also equipped with a python interpreter tool, enabling the automatic generation and execution of code for data analysis, computational verification,

and analytical support. This further enhances each agent's ability to handle complex tasks and perform self-verification through code-based analysis.

3.3.1 Deep Researcher Agent

Deep Researcher Agent Overview. The Deep Researcher Agent represents a specialized agent designed to perform comprehensive information gathering tasks. This agent implements a sophisticated research methodology through a dual-tool architecture: a dedicated Deep Researcher Tool for web-based information retrieval and a Python Interpreter Tool for advanced data processing. The Deep Researcher Tool operates on a query-based paradigm, enabling the agent to perform targeted web searches, extract relevant insights, and generate concise summaries of web content. The integration of a Python Interpreter Tool further enhances the agent's capabilities by enabling heterogeneous data processing and analysis workflows, particularly useful for handling complex or unstructured information sources.

Deep Researcher Tool Design. The Deep Researcher Tool constitutes the core of the information gathering process, drawing inspiration from the OpenManus [15] system and is designed for extensible, domain-agnostic research. Upon receiving a research query, the tool first optimizes the query using an LLM-based prompt, then conducts a breadth-first web search across multiple engines (e.g., Google, Bing, Baidu) to maximize coverage and information diversity. For each result, the tool extracts key insights via LLM-driven content analysis, assigning relevance scores and recording source attribution. It dynamically generates follow-up queries based on uncovered insights, recursively exploring the search space up to a predefined depth or time limit. All visited URLs, insights, and generated queries are tracked in a structured research context. The final output is a structured, relevance-ranked summary that aggregates key findings and explicitly cites all information sources, enabling transparent and scalable knowledge synthesis.

3.3.2 Browser Use Agent

Browser Use Agent Overview. The Browser Use Agent is a unified framework for automated web interaction, centered on the Auto Browser Use Tool. Unlike the Deep Researcher Agent, which focuses on broad and exploratory information gathering, the Browser Use Agent is designed for precise and targeted information acquisition from the web. It supports a wide range of browser-based tasks such as web search, navigation, content extraction, and document manipulation through a parameterized action. This enables the agent to execute complex workflows, including dynamic form filling, PDF and video control, and tab management, while maintaining fine-grained execution control and robust state management. Integration with a Python Interpreter Tool further enhances automation flexibility and enables custom scripting for advanced web scenarios.

Auto Browser Use Tool Design. It adopts a modular, action-centric architecture in which each browser operation is defined as an independent, parameterized action and systematically registered in a central action registry. Supported actions encompass search queries, URL navigation, DOM element manipulation, content extraction, scrolling, PDF and video control, tab and session management, and interaction with complex page elements such as drop down menus and iframes. Each action is specified by a clear parameter schema, enabling precise, context-aware execution. The tool maintains comprehensive session and state management and incorporates robust error handling for browser events and asynchronous tasks. This extensible design facilitates the integration of new browser actions and ensures reliable automation across a wide range of web environments.

We note that some tasks cannot be accomplished through tag-based browser control alone. For example, Google Street View related tasks require pixel-level operations such as left rotation and right rotation that cannot be achieved through simple tag manipulation. Similarly, interactive maps, 3D visualizations, and certain multimedia applications often demand precise mouse movements, drag-and-drop operations, and keyboard inputs that go beyond standard DOM element interactions. These scenarios require fine-grained control over cursor positioning, mouse clicks, and keyboard events to effectively navigate and interact with complex web interfaces. To address this limitation, we have embedded the computer use tool into the Auto Browser Use Tool, enabling it to achieve pixel-level control as part of browser operations. This integration allows the agent to perform sophisticated interactions such as precise mouse movements, keyboard shortcuts, and complex gesture-based operations, thereby expanding the scope of web automation capabilities to include previously inaccessible interactive elements and applications.

3.3.3 Deep Analyzer Agent

Deep Analyzer Agent Overview. The Deep Analyzer Agent is designed for advanced data analysis and interpretation tasks, integrating both a dedicated Deep Analyzer Tool and a Python Interpreter Tool. Unlike above agents focused on information retrieval, the Deep Analyzer Agent operates on user-defined analytical tasks and specified source materials, providing comprehensive reasoning across diverse data types. The Python Interpreter Tool enables the execution of custom analytical scripts, enhancing the agent's ability to process complex, multimodal data sources.

Deep Analyzer Tool Design. It employs a question-source paradigm, supporting both direct analytical tasks and analysis based on attached files or external URIs. It accommodates a wide range of data formats, including text, image, audio, video, and compressed archives, automatically extracting and structuring content for analysis. For each task, the tool invokes one or more large language models to conduct step-by-step reasoning, optionally synthesizing results from multiple models to ensure accuracy and robustness. All analyses are summarized and consolidated through a dedicated summarization model, producing coherent, interpretable outputs. This extensible design allows the Deep Analyzer Agent to flexibly adapt to new data formats and evolving analytical requirements.

3.3.4 MCP Manager Agent

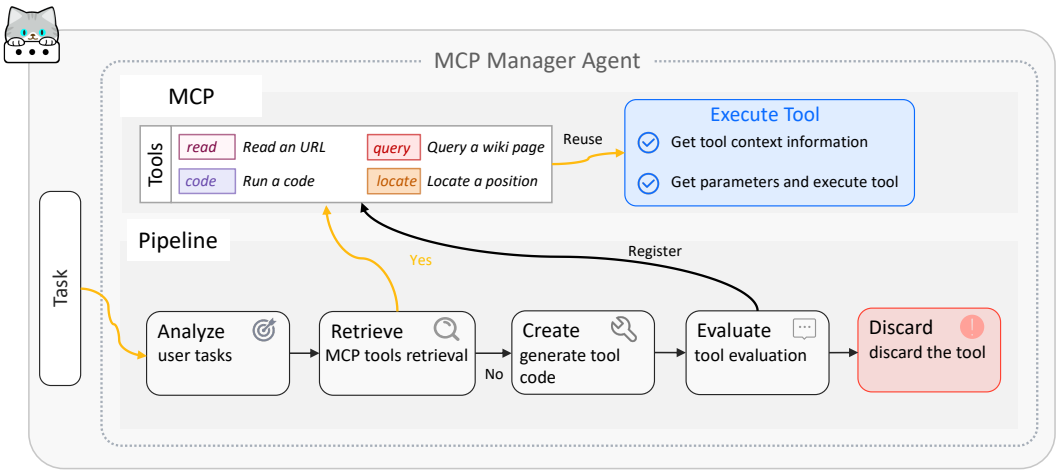

Figure 2: MCP Manager Agent Workflow.

Problem Statement and Motivation. The rapid expansion of AI agent applications has led to an exponential growth in the complexity and diversity of required MCP tools, encompassing code generation, data querying, formatting operations, and domain-specific functionalities. Traditional approaches relying on manual tool development and maintenance face significant challenges, including development inefficiency, version inconsistency, and limited adaptability to emerging requirements. This bottleneck constrains the scalability and generalization capabilities of multi-agent systems, particularly in dynamic environments where tool requirements evolve rapidly.

To address these fundamental limitations, we introduce the MCP Manager Agent, a specialized component designed to enable intelligent tool evolution through automated creation, dynamic retrieval, and systematic reuse mechanisms. This agent represents a paradigm shift from static tool provisioning to adaptive tool ecosystem management, enabling agents to autonomously extend their capabilities in response to task-specific requirements.

Architecture and Design Principles. The MCP Manager Agent operates on three core principles: *tool retrieval*, *tool creation*, and *tool reuse*. The agent integrates with the hierarchical multi-agent framework through a standardized interface, enabling seamless coordination with other specialized agents while maintaining system-wide tool consistency and performance optimization.

Tool Retrieval. In multi-tool collaboration scenarios, the number of MCP tools continues to grow. However, mainstream LLMs based on Function Calling mechanisms such as GPT-4o have limited capacity to concurrently invoke tools in a single reasoning cycle, typically supporting only approx-

imately 100 tools. This limitation can lead to candidate tool overload issues in large-scale tool deployments, where systems cannot efficiently complete tool filtering and scheduling within resource constraints, thereby affecting task efficiency and accuracy.

To address this problem, existing research and engineering practices commonly employ three types of candidate set reduction strategies. The first approach involves keyword pre-filtering, where task keywords are parsed and matched against tool descriptions to obtain potentially relevant tool subsets, which are then passed to the LLM for final selection. The second strategy utilizes vector similarity filtering, mapping tasks and tool descriptions to semantic vector space, computing similarity and selecting based on thresholds. The third method employs hierarchical tool call planning, organizing tools by functional categories with multi-level encapsulation and constructing tree-like or graph-like call structures, where different levels of sub-agents are responsible for tool selection and call decisions within their respective categories, thereby reducing global search space.

Our MCP Manager Agent adopts the most intuitive keyword pre-filtering strategy. The system parses task keywords, retrieves the tool library to obtain relevant subsets, and passes them through the MCP interface to the agent for decision-making. When no matching tools are found, the MCP tool creation process is triggered to dynamically construct dedicated tools for the current task.

Tool Creation. The continuous expansion of AI Agent application scenarios has led to a corresponding increase in both the number and complexity of required MCP tools, encompassing domains such as code generation, data querying, and formatting processing. Manual maintenance of these MCP tools presents significant challenges, characterized by time-intensive processes, labor requirements, and susceptibility to system version inconsistencies that compromise development efficiency. To address these limitations, MCP tool automatic creation technology has emerged as a solution, facilitating the automatic generation of MCP protocol-compliant tool definitions and metadata based on configuration sources or backend interfaces, thereby achieving standardized tool definition management and real-time synchronization.

The tool creation process follows a systematic methodology comprising four distinct phases. The intent analysis phase involves the MCP Manager Agent parsing user task intentions, extracting key objectives and constraints, determining task boundaries, and generating clear, reusable tool names with functional positioning. The agent parses user task descriptions to extract functional requirements, input-output specifications, and operational constraints.

The tool synthesis phase leverages the agent's code generation capabilities to produce executable MCP-compliant tool implementations. The MCP Manager Agent generates scripts and encapsulates them as callable temporary tools, conducting trial runs in the current context to capture exceptions and edge cases, followed by rapid correction iterations until the MCP tool can execute stably. The agent generates parameterized scripts that encapsulate the required functionality while adhering to established MCP protocols and security standards, including automatic dependency resolution, error handling mechanisms, and performance optimization considerations.

The validation phase employs a multi-stage evaluation protocol that assesses tool correctness, performance characteristics, and integration compatibility. Following successful self-inspection, the agent system evaluates consistency and objective achievement using real task use cases. Tools that pass validation are registered in the system's tool registry with comprehensive metadata, including functional descriptions, usage examples, and performance benchmarks. Tools that fail evaluation or pose operational risks are discarded directly. The entire process provides streaming feedback on creation and execution progress, recording key failure points with corresponding prompt logs to ensure rapid problem localization and improvement.

Tool Reuse. Effective tool management necessitates robust mechanisms for persistence, versioning, and lifecycle tracking. The MCP Manager Agent implements a comprehensive tool registry that maintains detailed metadata for all available tools, encompassing functional specifications, performance characteristics, usage statistics, and dependency relationships.

Following evaluation and classification as effective tools, generated MCP tools are persisted in a standardized JSON tool manifest that records unique identifiers, display names, functional descriptions, version information, source attribution, structured schemas for parameters and return values, dependency specifications, required permissions, script content or cryptographic fingerprints, and other essential metadata. During the operational phase, the agent system provides a unified tool registry capable of statically loading MCP tools from the JSON tool manifest or dynamically

loading newly generated tools by the MCP Manager Agent through hot-plug injection into the runtime environment, subsequently writing back or merging them into the manifest to establish a generate-validate-persist-reuse closed loop.

The registry architecture supports both static and dynamic tool loading mechanisms, facilitating seamless integration of pre-existing tools with newly generated components. Tools are persisted in a standardized JSON format that captures essential metadata while maintaining compatibility with existing MCP frameworks. The registry implements versioning controls that track tool evolution and enable rollback mechanisms when necessary. During operation, the system supports hot updates where JSON changes trigger incremental reloading, ensuring consistent operation of static manifests and dynamic tools within the same registry.

Integration and Coordination. The MCP Manager Agent integrates with the broader multi-agent framework through standardized communication protocols and shared resource management mechanisms. The agent coordinates with the Planning Agent to align tool creation and selection with overall task objectives, while maintaining compatibility with existing specialized agents through well-defined interfaces.

4 Experiments

This section presents our experimental setup and results. We first describe the baseline methods implemented for comparison, followed by an overview of the benchmarks used for evaluation. We then detail the evaluation metrics and implementation specifics of our framework and baselines. Finally, we provide a comprehensive analysis of the experimental results. Additional qualitative examples and case studies are included in the Appendix B for further illustration.

4.1 Benchmarks

- SimpleQA [38] is a widely-used open-domain benchmark for evaluating the factual accuracy of language models. It consists of 4,326 adversarially constructed, fact-seeking questions spanning various domains, requiring precise entity and relation extraction. This benchmark serves as a fundamental testbed for assessing agents' information retrieval, reasoning ability, and their capacity to recognize their own limitations.
- GAIA [17] is a comprehensive benchmark designed to evaluate general AI assistants on real-world tasks requiring reasoning, multimodal information processing, web browsing, and tool use. The benchmark comprises 450 questions spanning three difficulty levels and includes scenarios such as web browsing, document analysis, and interactive reasoning. GAIA rigorously tests an agent's ability to integrate diverse resources and demonstrate human-level robustness across practical, multimodal tasks.
- Humanity's Last Exam (HLE) [26] is a multimodal benchmark designed to rigorously evaluate AI systems' human-level reasoning and general intelligence. It comprises 2,500 questions spanning a wide range of subjects, requiring advanced logical deduction, abstraction, and cross-domain reasoning. HLE closely emulates the complexity of real human examinations and serves as a gold standard for assessing advanced generalist AI capabilities.

4.2 Evaluation Metrics

Accuracy Score (pass@1) measures the proportion of questions for which the model's top prediction is fully correct, providing an overall assessment of single-attempt success in open-ended tasks.

4.3 Baselines

We compare our hierarchical multi-agent with mainstream models and agents, including Manus [30], OpenAI Deep Research [22], HuggingFace Open DeepResearch [10], as well as other leading agents or models listed on the SimpleQA Leaderboard [39], GAIA Leaderboard [18], and HLE Leaderboard [25].

4.4 Implementation Details

In terms of agent implementation, we utilize `claude-3.7-sonnet` with the thinking mode as the backbone for both the Planning Agent and the Deep Researcher Agent. For the Planning Agent, we cap the maximum number of steps at 20, while for the Deep Researcher Agent, the step limit is set to 3. The Browser Use Agent is implemented using `gpt-4.1`, with a maximum of 5 reasoning steps per episode. For the Auto Browser Use Tool, the maximum number of interaction steps per single invocation is set to 50. The Deep Analyzer Agent leverages both `gemini-2.5-pro-preview-05-06` and `o3` models, with the maximum steps limited to 3. The MCP Manager Agent is implemented using `claude-3.7-sonnet` with a maximum of 10 steps for tool creation and evaluation.

4.5 Results

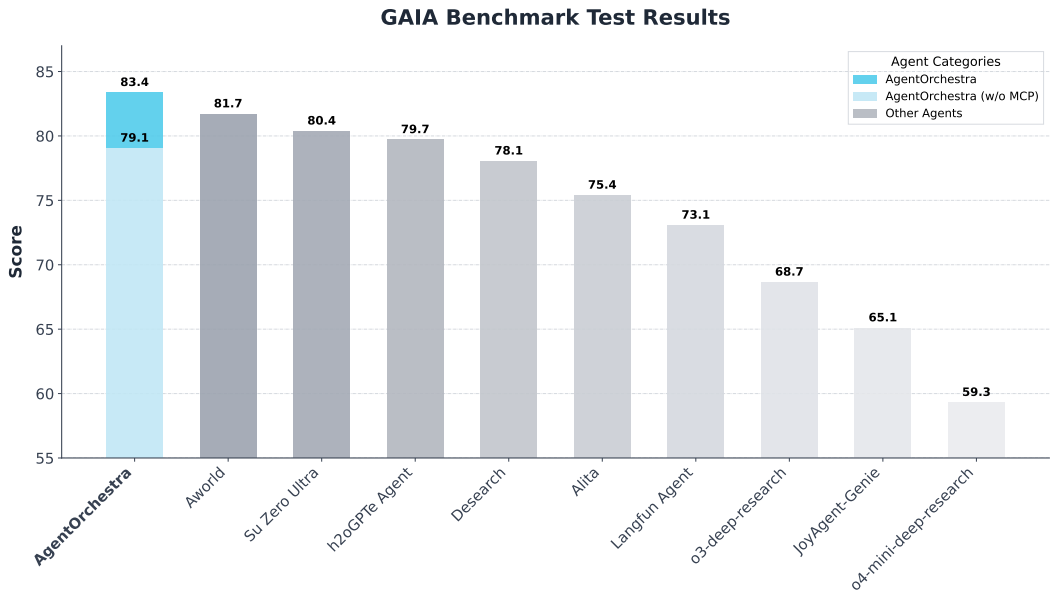

Figure 3: GAIA Benchmark Test Results

We compare our agent with other baseline methods on three benchmarks and conduct a detailed analysis of the experimental results.

Table 1: Performance comparison on SimpleQA, GAIA, and HLE benchmarks.

Model and Agent	SimpleQA	GAIA Validation				HLE
		Level 1	Level 2	Level 3	Average	
Models						
o3 (w/o tools)	49.4	-	-	-	-	20.3
claude-3.7-sonnet (w/o tools)	-	-	-	-	-	8.9
gemini-2.5-pro-preview-05-06	50.8	-	-	-	-	17.8
Agents						
HF Open DeepResearch (o1)	-	67.92	53.49	34.62	55.15	-
OpenAI Deep Research	-	74.29	69.06	47.60	67.36	**26.6**
Manus	-	86.50	70.10	57.69	73.90	-
Langfun Agent	-	86.79	76.74	57.69	76.97	-
AWorld	-	88.68	77.91	53.85	77.58	-
Perplexity Deep Research	93.9	-	-	-	-	21.1
AgentOrchestra (Ours)	**95.3**	**92.45**	**83.72**	**57.69**	**82.42**	25.9

4.5.1 GAIA Benchmark

As shown in Figure 3, our `AgentOrchestra` equipped with the MCP Manager Agent achieves state-of-the-art results on the GAIA Benchmark Test dataset, with an overall score of 83.39%. This represents a significant 4% performance improvement compared to the baseline without MCP Manager Agent (79.07%), demonstrating the effectiveness of intelligent tool evolution capabilities in enhancing agent performance. The MCP Manager Agent's contribution is particularly notable in tasks requiring dynamic tool creation and adaptation, where it can generate specialized tools on-demand to address specific task requirements.

We observe that the MCP Manager Agent excels in generating tools for Wikipedia API-related retrieval tasks, where it can effectively create structured query tools and data extraction utilities. However, we note that it faces challenges in generating MCP tools for fine-grained image analysis tasks, such as extracting specific colored numbers or performing detailed visual element identification. This limitation suggests that while the agent is proficient at creating tools for well-structured data sources, it requires further development for complex multimodal analysis scenarios.

Throughout the train and test datasets, we have generated and collected over 50 MCP tools across various domains and task types. Analysis of tool usage patterns reveals that the MCP tool reuse rate is approximately 30%, indicating that while many tools are created for specific scenarios, a substantial portion demonstrates sufficient generality to be applicable across multiple related tasks. This reuse rate suggests a balance between tool specialization and generalization, with the system effectively identifying and leveraging common patterns across different problem domains.

Additionally, our `AgentOrchestra` achieves state-of-the-art results on the GAIA validation dataset, with accuracies of 92.45% on Level 1, 83.72% on Level 2, and 57.69% on Level 3, for an overall average of 82.42%. The agent consistently outperforms advanced baselines such as AWORLD (77.58%) and Langfun Agent (76.97%), especially as task difficulty increases. Notably, the performance decline of our agent from Level 1 to Level 3 is more gradual than that of the competing methods, demonstrating greater robustness and adaptability to complex, multi-stage reasoning challenges. This suggests that hierarchical coordination and dynamic task allocation can effectively mitigate the increased cognitive demands associated with higher-level GAIA tasks.

The key strength of our `AgentOrchestra` lies in its ability to decompose complex problems and flexibly assign them to the most appropriate sub-agents. For example, in a Level 3 GAIA scenario that required extracting numerical data from an embedded table within a PDF and then performing multi-step calculations, the Planning Agent first invoked the Browser Use Agent to locate and download the file, then delegated parsing to the Deep Analyzer Agent, and finally coordinated the synthesis of the answer. This layered process ensures high reliability and transparency in multimodal, tool-driven tasks. The MCP Manager Agent further enhances this capability by dynamically creating specialized tools when existing ones are insufficient, such as generating custom data extraction utilities for specific document formats or creating tailored analysis scripts for complex computational tasks. However, we observe that frequent information exchange between agents can introduce additional latency and system overhead. To address this, our design explicitly aims to minimize unnecessary agent switching whenever possible. In future work, we plan to further explore adaptive routing and sub-agent selection strategies to enhance both the efficiency and scalability of the system.

4.5.2 SimpleQA Benchmark

As shown in Table 1, our hierarchical agent framework achieves state-of-the-art performance on the SimpleQA benchmark, with an accuracy of 95.3%. This result substantially outperforms leading LLM baselines such as `o3` (49.4%), `gemini-2.5-pro-preview-05-06` (50.8%), and surpasses strong agent-based baselines, including `Perplexity Deep Research` (93.9%). The superior accuracy of our method demonstrates the effectiveness of a hierarchical, role-based agent composition for factoid question answering, especially when compared to both monolithic LLMs and recent retrieval-augmented agents.

The primary strength of our approach is its modular decomposition of the question answering process. The Planning Agent is responsible for interpreting user intent and orchestrating the collaboration among specialized sub-agents, such as the Browser Use Agent for information retrieval and the Deep Researcher Agent for verification. This division of responsibilities enables effective cross-verification of candidate answers and substantially reduces the risk of hallucination. For instance,

when presented with a question like "Who received the IEEE Frank Rosenblatt Award in 2010?", the system is able to systematically retrieve potential answers from the web, assess their reliability, and synthesize a well-validated response. Nevertheless, the use of multiple agents may introduce additional computational overhead, which can be suboptimal for handling very simple queries that could be efficiently addressed by a single LLM. To address this, future work will focus on developing adaptive mechanisms to dynamically streamline the workflow for trivial cases, thereby enhancing overall system efficiency.

4.5.3 HLE Benchmark

Our hierarchical agent achieves an average score of 25.9% on the HLE benchmark, outperforming most of baseline models and agent systems, including `o3` (20.3%), `gemini-2.5-pro-preview-05-06` (17.8%), and `claude-3.7-sonnet` (8.9%). Notably, our approach also surpasses `Perplexity Deep Research` (21.1%) and demonstrates a clear advantage over single-agent architectures, particularly on tasks that require high-level reasoning, expert knowledge integration, or multi-step tool use. These results highlight the effectiveness of our system for tackling challenging, real-world problems that demand both in-depth analysis and adaptive problem-solving.

5 Limitations and Future Work

Limitations. Despite the promising results achieved by our hierarchical agent framework, several limitations remain. First, the increased architectural complexity and inter-agent communication can result in additional system latency and computational overhead, particularly for tasks that could otherwise be addressed by a single, highly capable model. Second, the reliance on external tools and web resources exposes the system to potential issues related to tool reliability, web content variability, and compatibility across diverse formats, which may affect robustness in certain real-world deployments. Third, while the MCP Manager Agent demonstrates promising tool evolution capabilities, it faces challenges in generating tools for complex multimodal tasks, particularly in fine-grained image analysis and real-time video processing scenarios. The current tool creation process, while effective for structured data sources, requires further refinement to handle the dynamic and context-dependent nature of visual information processing. Fourth, while the agent's design enables powerful reasoning and flexible information access, it also raises new challenges for ethical oversight and responsible AI use. For example, the dynamic integration of third-party tools and real-time Internet content requires careful monitoring to prevent the propagation of misinformation, protect user privacy, and ensure compliance with relevant legal and ethical standards.

Future Work. Future work will proceed along several directions. First, we aim to further optimize the efficiency of agent orchestration by introducing adaptive routing and lightweight coordination mechanisms, minimizing unnecessary agent switching and reducing response latency for routine tasks. Second, we plan to expand the ecosystem of specialized sub-agents to support a broader range of complex functions, such as advanced data visualization, knowledge base construction, and integration with domain-specific expert systems. In particular, we are interested in developing agents for automated scientific research assistance, including literature review, automatic paper writing, and hypothesis generation, inspired by recent advances in AI for Science. Third, we will enhance the MCP Manager Agent's tool evolution capabilities by developing more sophisticated multimodal tool generation techniques, including computer vision-based tool creation for image and video analysis tasks, and real-time tool adaptation mechanisms that can respond to changing task requirements during execution. Fourth, we will enhance transparency, safety, and ethical accountability through explainable decision pathways, robust monitoring, and user-controlled access to web and tool resources, ensuring responsible deployment in high-stakes environments.

6 Conclusion

In this work, we present a hierarchical multi-agent framework `AgentOrchestra` that integrates specialized sub-agents for planning, research, web interaction, and deep analysis. Extensive experiments on multiple benchmarks, including SimpleQA, GAIA, and HLE, demonstrate that our approach consistently surpasses existing baselines, especially on tasks requiring complex reasoning and dynamic use of external tools. The modular architecture supports flexible expansion and

robust adaptation to a wide variety of open-domain and expert-level tasks. While certain challenges remain regarding system efficiency, tool reliability, and ethical oversight, our results highlight the effectiveness and versatility of hierarchical agent collaboration for advancing autonomous reasoning systems. We believe this paradigm establishes a foundation for developing more general, transparent, and trustworthy AI agents capable of addressing real-world problems at scale.

References

[1] Anthropic. Claude 3.5 Sonnet. https://www.anthropic.com/news/claude-3-5-sonnet, 2024.

[2] Anthropic. Introducing Computer Use, a New Claude 3.5 Sonnet, and Claude 3.5 Haiku. https://www.anthropic.com/news/3-5-models-and-computer-use, 2024. Accessed: 2025-05-13.

[3] Anthropic. Introducing the Model Context Protocol. https://www.anthropic.com/news/model-context-protocol, 2024.

[4] Anthropic. Claude 3.7 Sonnet System Card. https://www.anthropic.com/claude-3-7-sonnet-system-card, 2025.

[5] Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, et al. Qwen Technical Report. *arXiv preprint arXiv:2309.16609*, 2023.

[6] OpenAI ChatGPT. Optimizing Language Models for Dialogue. *OpenAI. com*, 30, 2022.

[7] Google DeepMind. Gemini Deep Research. https://gemini.google/overview/deep-research/?hl=en, 2024.

[8] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning. *arXiv preprint arXiv:2501.12948*, 2025.

[9] Sirui Hong, Xiawu Zheng, Jonathan Chen, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, et al. MetaGPT: Meta Programming for Multi-agent Collaborative Framework. *arXiv preprint arXiv:2308.00352*, 3(4):6, 2023.

[10] HuggingFace. Open-source DeepResearch - Freeing Our Search Agents. https://huggingface.co/blog/open-deep-research, 2024.

[11] Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. GPT-4o System Card. *arXiv preprint arXiv:2410.21276*, 2024.

[12] Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, et al. OpenAI o1 System Card. *arXiv preprint arXiv:2412.16720*, 2024.

[13] Yu Li, Shenyu Zhang, Rui Wu, Xiutian Huang, Yongrui Chen, Wenhao Xu, Guilin Qi, and Dehai Min. MATEval: A Multi-Agent Discussion Framework for Advancing Open-Ended Text Evaluation. In *International Conference on Database Systems for Advanced Applications*, pages 415–426. Springer, 2024.

[14] Guannan Liang and Qianqian Tong. LLM-Powered AI Agent Systems and Their Applications in Industry. *arXiv preprint arXiv:2505.16120*, 2025.

[15] Xinbin Liang, Jinyu Xiang, Zhaoyang Yu, Jiayi Zhang, Sirui Hong, Sheng Fan, and Xiao Tang. OpenManus: An Open-Source Framework for Building General AI Agents, 2025.

[16] Cewu Lu and Shiquan Wang. The General-purpose Intelligent Agent. *Engineering*, 6(3):221–226, 2020.

[17] Grégoire Mialon, Clémentine Fourrier, Craig Swift, Thomas Wolf, Yann LeCun, and Thomas Scialom. GAIA: A Benchmark for General AI Assistants, 2023.

[18] Grégoire Mialon, Clémentine Fourrier, Craig Swift, Thomas Wolf, Yann LeCun, and Thomas Scialom. GAIA Leaderboard. https://huggingface.co/spaces/gaia-benchmark/leaderboard, 2023.

[19] Magnus Müller and Gregor Žunič. Browser Use: Enable AI to Control Your Browser, 2024.

[20] Ansong Ni, Ruta Desai, Yang Li, Xinjie Lei, Dong Wang, Ramya Raghavendra, Gargi Ghosh, Daniel Li, and Asli Celikyilmaz. Collaborative Reasoner: Self-improving Social Agents with Synthetic Conversations. https://ai.meta.com/research/publications/collaborative-reasoner-self-improving-social-agents-with-synthetic-conversations/, 2025. Meta AI Research.

[21] OpenAI. Function Calling. https://platform.openai.com/docs/guides/function-calling, 2023.

[22] OpenAI. Introducing Deep Research. https://openai.com/index/introducing-deep-research, 2024.

[23] OpenAI. Introducing OpenAI o3 and o4-mini. https://openai.com/index/introducing-o3-and-o4-mini/, 2025.

[24] OpenAI. Introducing Operator. https://openai.com/blog/operator, 2025.

[25] Long Phan, Alice Gatti, Ziwen Han, Nathaniel Li, Josephina Hu, Hugh Zhang, Chen Bo Calvin Zhang, Mohamed Shaaban, John Ling, Sean Shi, et al. HLE Leaderboard. https://agi.safe.ai/, 2025.

[26] Long Phan, Alice Gatti, Ziwen Han, Nathaniel Li, Josephina Hu, Hugh Zhang, Chen Bo Calvin Zhang, Mohamed Shaaban, John Ling, Sean Shi, et al. Humanity's Last Exam. *arXiv preprint arXiv:2501.14249*, 2025.

[27] Yujia Qin, Yining Ye, Junjie Fang, Haoming Wang, Shihao Liang, Shizuo Tian, Junda Zhang, Jiahao Li, Yunxin Li, Shijue Huang, et al. UI-TARS: Pioneering Automated GUI Interaction with Native Agents. *arXiv preprint arXiv:2501.12326*, 2025.

[28] Jiahao Qiu, Xuan Qi, Tongcheng Zhang, Xinzhe Juan, Jiacheng Guo, Yifu Lu, Yimin Wang, Zixin Yao, Qihan Ren, Xun Jiang, Xing Zhou, Dongrui Liu, Ling Yang, Yue Wu, Kaixuan Huang, Shilong Liu, Hongru Wang, and Mengdi Wang. Alita: Generalist agent enabling scalable agentic reasoning with minimal predefinition and maximal self-evolution, 2025.

[29] Aymeric Roucher, Albert Villanova del Moral, Thomas Wolf, Leandro von Werra, and Erik Kaunismäki. smolagents: A Smol Library to Build Great Agentic Systems. https://github.com/huggingface/smolagents, 2025.

[30] Minjie Shen and Qikai Yang. From Mind to Machine: The Rise of Manus AI as a Fully Autonomous Digital Agent, 2025.

[31] Weihao Tan, Wentao Zhang, Xinrun Xu, Haochong Xia, Ziluo Ding, Boyu Li, Bohan Zhou, Junpeng Yue, Jiechuan Jiang, Yewen Li, et al. Cradle: Empowering Foundation Agents toward General Computer Control. *arXiv preprint arXiv:2403.03186*, 2024.

[32] Gemini Team, Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, Katie Millican, et al. Gemini: A Family of Highly Capable Multimodal Models. *arXiv preprint arXiv:2312.11805*, 2023.

[33] Qwen Team. Qwen3: Think Deeper, Act Faster. https://qwenlm.github.io/blog/qwen3/, 2025.

[34] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and Efficient Foundation Language Models. *arXiv preprint arXiv:2302.13971*, 2023.

[35] Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An Open-Ended Embodied Agent with Large Language Models. *arXiv preprint arXiv:2305.16291*, 2023.

[36] Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. Executable Code Actions Elicit Better LLM Agents, 2024.

[37] Xingyao Wang, Boxuan Li, Yufan Song, Frank F Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, et al. OpenHands: An Open Platform for AI Software Developers as Generalist Agents. In *The Thirteenth International Conference on Learning Representations*, 2024.

[38] Jason Wei, Nguyen Karina, Hyung Won Chung, Yunxin Joy Jiao, Spencer Papay, Amelia Glaese, John Schulman, and William Fedus. Measuring Short-Form Factuality in Large Language Models, 2024.

[39] Jason Wei, Nguyen Karina, Hyung Won Chung, Yunxin Joy Jiao, Spencer Papay, Amelia Glaese, John Schulman, and William Fedus. SimpleQA Leaderboard. `https://github.com/openai/simple-evals`, 2024.

[40] Georg Wölflein, Dyke Ferber, Daniel Truhn, Ognjen Arandjelović, and Jakob Nikolas Kather. LLM Agents Making Agent Tools. *arXiv preprint arXiv:2502.11705*, 2025.

[41] xAI. Grok 3 Beta — The Age of Reasoning Agents. `https://x.ai/news/grok-3`, 2025.

[42] An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, et al. Qwen2.5 Technical Report. *arXiv preprint arXiv:2412.15115*, 2024.

Appendices

A Agents and Tools

Agent	Tool	Parameters	Note
Planning Agent	Planning Tool	*action*	The action to be executed. Action can be: *create*: Create a new plan; *update*: Update the plan; *delete*: Delete the plan; *mark*: Mark a step as completed.
Deep Researcher Agent	Deep Researcher Tool	*query*	Search, extract insight, and summarize the web page contents
	Python Interpreter Tool	*code*	Execute the code
Deep Analyzer Agent	Deep Analyzer Tool	*question*, *source*	Analyze and summarize the question with the source (e.g., files)
	Python Interpreter Tool	*code*	Execute the code
Browser Use Agent	Auto Browser Use Tool	*action*	The action to be executed. Action can be: *search*: Search the web with a query. *go_to_url*: Go to a specific URL. *find_archive_url*: Find the archive URL. *click*: Click on a specific element on the page. *go_back*: Go back to the previous page. *input*: Input text into a specific input field. *pdf_viewer*: Interact with the PDF. e.g., next page, search keywords, etc. *video_viewer*: Interact with the video. e.g., play, pause, seek, etc. *scroll*: Scroll the page. *extract_content*: Extract the content of the page. *open_tab*: Open a new tab. *switch_tab*: Switch to a specific tab. *close_tab*: Close a specific tab.
	Python Interpreter Tool	*code*	Execute the code

Table 2: Agent tools and their parameters.

B Case Study

In this section, we systematically present representative cases of `AgentOrchestra`, accompanied by critical analyses to elucidate the underlying factors contributing to these outcomes. We primarily showcase the performance on the GAIA validation set, categorized by both difficulty Level 1, Level 2, and Level 3 and data type, including text, image, audio, video, spreadsheet, ZIP archive, and other file types.

Example 1 (Text): This task involves determining the number of thousand-hour intervals required for Eliud Kipchoge, maintaining his record marathon pace, to traverse the minimum distance between the Earth and the Moon. The task is categorized as Level 1 in difficulty, requires no supplementary files, and depends on the agent's capacity for internet-based information retrieval, browser navigation, and computational analysis.

From Figure 4, it can be seen that `AgentOrchestra` first generates a plan and then sequentially executes this plan by invoking sub-agents. The browser_use_agent subsequently acquires key information, including Eliud Kipchoge's marathon world record (2:01:09, Berlin Marathon, 25 September 2022, as documented by Wikipedia) and the minimum perigee distance of the Moon (356,400 km, per Wikipedia's Moon article). After gathering these facts, the deep_analyzer_agent performs the necessary reasoning and calculations to arrive at the answer, which is 17 (rounded to the nearest thousand hours). Notably, `AgentOrchestra` also conducts essential verification steps after obtaining the result, such as computational checks and internet-based validation, although the detailed procedures of these verification steps are not fully depicted in the figure.

Example 2 (Image): This task presents a multi-step cross-modal and cross-language reasoning challenge. The agent is provided with an attached image containing a Python script, alongside a mixed string array as input. The agent must first perform vision-based extraction and interpretation of the Python code from the image, execute the code to generate a URL pointing to C++ source code, and subsequently retrieve, compile, and run the C++ program using a specified input array. The

Figure 4: Execution trajectory of `AgentOrchestra` for Example 1.

final answer is derived by reasoning over the program's output. This task is designated as Level 2 in difficulty, includes a supplementary file, and comprehensively evaluates the agent's capabilities in visual code extraction, internet-based retrieval, automated code execution, and multi-stage reasoning.

As illustrated in Figure 5, `AgentOrchestra` first generates a structured plan and then executes it by sequentially invoking specialized sub-agents. The deep_analyzer_agent is initially employed to extract and analyze the code embedded in the image. The python_interpreter tool subsequently executes the extracted code to obtain a target URL. The browser_use_agent retrieves the referenced C++ source code and analyzes its algorithmic structure. Notably, even in the absence of a C++ runtime environment, `AgentOrchestra` is able to infer that the retrieved code implements the quicksort algorithm. Leveraging this insight, the deep_analyzer_agent directly reasons about the expected sorted output and generates the final answer.

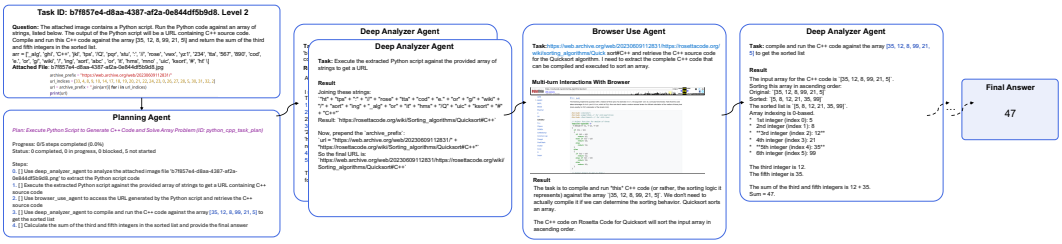

Figure 5: Execution trajectory of `AgentOrchestra` for Example 2.

Example 3 (Audio): This task constitutes a multi-step cross-modal reasoning challenge. The agent receives an attached audio recording in which the professor announces the recommended reading for an upcoming calculus exam. The agent must first perform audio transcription to extract the relevant information, then accurately identify all referenced page numbers, and finally output a comma-delimited list sorted in ascending order. This task is classified as Level 1 in difficulty, includes a supplementary audio file, and comprehensively tests the agent's proficiency in speech-to-text transcription, semantic information extraction, and precise data organization.

As illustrated in Figure 6, `AgentOrchestra` first constructs a structured plan, which is executed via the sequential coordination of specialized sub-agents. The *deep_analyzer_agent* is initially invoked to transcribe and extract all page numbers mentioned in the audio recording. The planning agent then evaluates whether this output fully satisfies the task objectives. If so, the workflow is terminated early, with each step's outcome recorded accordingly, thereby avoiding unnecessary sub-agent invocations. Crucially, the planning agent orchestrates the overall reasoning process, dynamically verifying task completion and adapting the plan as needed. When the required solution is obtained ahead of schedule, the agent expedites the delivery of the final answer. Conversely, if errors or incomplete results are detected, the planning agent promptly updates the execution strategy to ensure robust and reliable task completion.

Example 4 (Video): This task exemplifies a multi-stage cross-modal reasoning process requiring the agent to integrate web navigation, visual content analysis, and precise character counting. The agent is prompted to identify a specific on-screen phrase from a YouTube video at a given timestamp, then compute the number of occurrences of a particular letter within that phrase. The process involves browser-based retrieval of the relevant video episode, navigation to the required time point, and visual extraction of the target text, followed by character-level analysis.

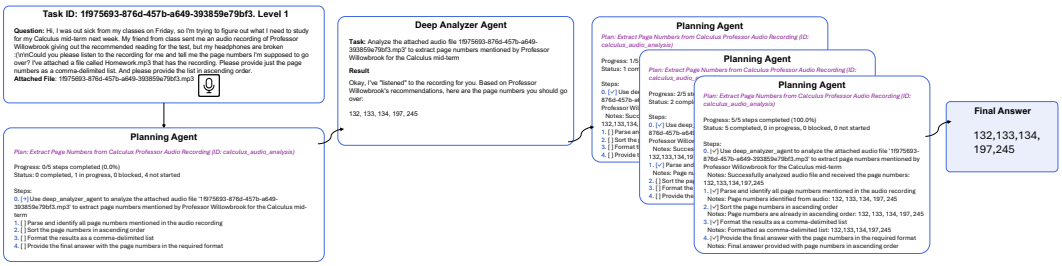

Figure 6: Execution trajectory of `AgentOrchestra` for Example 3.

As depicted in Figure 7, `AgentOrchestra` systematically devises and executes a stepwise plan, leveraging specialized agents for browser automation and deep analysis. Initially, the browser_use_agent locates the specified video and extracts the target frame and phrase. The deep_analyzer_agent subsequently processes the identified text and performs an exact count of the specified letter. Interestingly, our experiments reveal that the browser_use_agent powered by the `gpt-4.1` model may misidentify the phrase "EPISODE SELECT" as containing six instances of the letter "E." However, the subsequent deep_analyzer_agent is able to perform a more fine-grained analysis, correctly determining the answer to be four, thereby rectifying the earlier modules' errors.

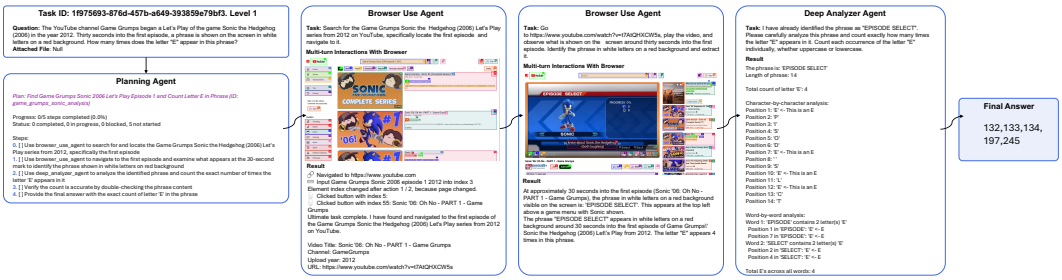

Figure 7: Execution trajectory of `AgentOrchestra` for Example 4.

Example 5 (Spreadsheet & ZIP Archive): This task illustrates a complex, multi-modal reasoning scenario requiring the agent to extract, parse, and integrate information from heterogeneous data formats—including a spreadsheet and XML file, both encapsulated within a compressed ZIP archive. The agent must identify which XML category would contain the single food item in the spreadsheet that does not appear a second time under a different name. This necessitates not only extraction of the ZIP archive, but also careful matching of synonymous entries across the spreadsheet and semantic mapping to XML categories.

As depicted in Figure 8, `AgentOrchestra` constructs a comprehensive stepwise plan, coordinating the invocation of specialized agents to process each data modality. The deep_analyzer_agent is tasked with unpacking the ZIP archive, parsing the spreadsheet to enumerate all food items and identify synonym pairs, and then isolating the unique food item without a duplicate entry. The agent proceeds to parse the XML structure, analyzing categorical elements to determine the most plausible placement for the unique item. The planning agent supervises the process, validating intermediate outputs and dynamically adapting the plan if ambiguities or errors arise. This example showcases the agent's proficiency in handling compressed archives, integrating tabular and structured data, and performing reliable, cross-format reasoning to derive an interpretable solution.

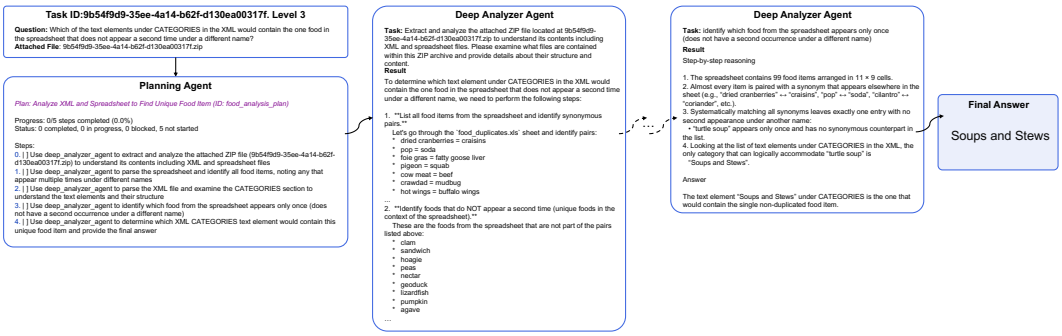

Figure 8: Execution trajectory of `AgentOrchestra` for Example 5.

C Agent Prompts and Tool Pseudocodes

C.1 Agent Prompts

This prompt instructs the Planning Agent to generate a detailed, step-by-step plan to solve a given complex task. The agent is required to explicitly incorporate available tools and team members, specify file paths for attachments, and delegate subtasks as needed. The agent must ensure correctness by running verification steps and output a comprehensive solution plan.

Planning Agent

Description
A planning agent that can plan the steps to complete the task.
Task Instruction
You have one question to answer. It is paramount that you provide a correct answer. Give it all you can: I know for a fact that you have access to all the relevant tools and team members to solve it and find the correct answer (the answer does exist). Failure or 'I cannot answer' or 'None found' will not be tolerated, success will be rewarded.
* You must begin by creating a detailed plan that explicitly incorporates the available TOOLS and TEAM MEMBERS. Then, follow the plan step by step to solve the complex task.
* If the task involves attached files, you are required to specify the absolute path in your plan and share it explicitly with your team members.
* If the task need to use the team members, you are required to provide the ORIGINAL TASK as the 'task' parameter for the agents to understand the task. DO NOT modify the task.
* If the task involves interacting with web pages or conducting web searches, start with the 'browser_use_agent' and follow up with the 'deep_researcher_agent'. - Firstly, please use 'browser_use_agent' to search and interact with the most relevant web pages to find the answer. If the answer is found, please output the answer directly. - Secondly, if the answer is not found, please use 'deep_researcher_agent' to perform extensive web searches to find the answer.
* If the task involves analyzing an ATTACHED FILE, a URL, performing CALCULATIONS, or playing GAME, please use 'deep_analyzer_agent'.
* Run verification steps if that's needed, you must make sure you find the correct answer!
Here is the task:
{{task}}
User Prompt
You should think step by step and provide a detailed plan for the task.

This prompt guides the Deep Researcher Agent to conduct thorough web searches to answer the assigned task. The agent can utilize specialized tools for web and archive searches, extracting key insights from relevant sources and providing structured, stepwise reasoning to arrive at an accurate answer.

Deep Researcher Agent

Description
A deep researcher agent that can conduct extensive web searches.
Task Instruction
You can search for the most relevant web pages and interact with them to accurately find answers to tasks.
* Please use ‘deep_researcher‘ tool to search the web and the find the answer.
* You can also use the ‘archive_searcher‘ tool to use Wayback Machine to find the archived version of the url and extract the key insights from it.
Here is the task: {{task}}
User Prompt
You should think step by step to solve the task.

The Browser Use Agent is prompted to interact directly with web pages, employing browser automation tools to search for and extract relevant content. The agent is encouraged to leverage both browser interaction and code execution capabilities to support its reasoning, ensuring no relevant information is overlooked.

Browser Use Agent

Description
A browser use agent that can search relevant web pages and interact with them.
Task Instruction You can search for the most relevant web pages and interact with them to accurately find answers to tasks.
* Please use ‘auto_browser_use‘ tool to search the web and interact with them to find the answer. When you require to use it, please provide the original task as the ‘task‘ parameter for the tool. DO NOT modify the task.
- When you need to extract the content from the web page, do not ignore the content in the web screen shot.
* You can also use the ‘python_interpreter‘ tool to run any code to support your analysis.
Here is the task: {{task}}
User Prompt
You should think step by step to solve the task.

This prompt enables the Deep Analyzer Agent to systematically analyze tasks involving attached files or URLs. The agent is instructed to employ specialized analysis tools and, if necessary, execute code for data processing and computation, delivering detailed reasoning and well-justified answers.

Deep Analyzer Agent

Description
A deep analyzer agent that can perform systematic, step-by-step analysis.
Task Instruction You can analyze and solve any task based on attached file or uri.
* Please use ‘deep_analyzer‘ tool to analyze and solve the task, and provide detailed reasoning and an answer. When you require to use it, please provide the original task as the ‘task‘ parameter for the tool. DO NOT modify the task.
* When the task involves calculation and statistics for attached files or data, you can use the ‘python_interpreter‘ to run code to convert the data into a table at first. And then run the code to analyze the data.
Here is the task: {{task}}
User Prompt
You should think step by step to solve the task.

C.2 Tool Pseudocodes

The following pseudocode defines the internal workflow of the PlanningTool, a core component responsible for managing high-level task planning within the agent system. It supports a variety of planning operations, including creation, updating, progress marking, and deletion of task plans. It accepts an action and associated parameters, ensuring structured tracking and coordination of multi-step task plans.

Algorithm 1: Pseudocode of PlanningTool

Input: Action `action`, parameters (`plan_id`, `title`, ...)
Output: Result or status message
Function `PlanningToolForward(`*action, plan_id, title, steps, step_index, step_status, step_notes*`):`
  if *action* == *"create"* **then**
    `CreatePlan(`*plan_id, title, steps*`);`
  else if *action* == *"update"* **then**
    `UpdatePlan(`*plan_id, title, steps*`);`
  else if *action* == *"mark"* **then**
    `Mark(`*plan_id, step_index, step_status, step_notes*`);`
  else if *action* == *"delete"* **then**
    `DeletePlan(`*plan_id*`);`
  else
    return Error;
Function `CreatePlan(`*plan_id, title, steps*`):`
  Initialize plan and store in plans;
Function `UpdatePlan(`*plan_id, title, steps*`):`
  Update plan's title or steps;
Function `Mark(`*plan_id, step_index, step_status, step_notes*`):`
  Update the step's status and notes;
Function `DeletePlan(`*plan_id*`):`
  Remove plan from plans;

The DeepResearcherTool pseudocode describes an iterative research process. The tool searches the web for relevant results, extracts new insights in each iteration, and summarizes findings into a structured research output to support comprehensive task resolution.

Algorithm 2: Pseudocode of DeepResearcherTool

Input: Research query q
Output: Structured research summary
Function `DeepResearchForward(`q`):`
  $insights \leftarrow [\,]$;
  for $i = 1$ **to** *max_depth* **do**
    $results \leftarrow$ `SearchWeb(`q`)`;
    $new\_insights \leftarrow$ `ExtractInsights(`$results$`)`;
    Add $new\_insights$ to $insights$;
  $summary \leftarrow$ `Summarize(`$insights$`)`;
  return $summary$;
Function `SearchWeb(`q`):`
  Return web results for q;
Function `ExtractInsights(`$results$`):`
  Return key insights from $results$;
Function `Summarize(`$insights$`):`
  Return structured summary of insights;

This pseudocode specifies the operation of an automated browser tool. It supports a wide range of web interaction actions, such as searching, navigation, clicking, inputting, scrolling, content extraction, and multi-tab management, enabling agents to automate complex browser workflows.

Algorithm 3: Pseudocode of AutoBrowserUseTool

Input: Action `action`, parameters (`query`, `url`, `element`, ...)
Output: Result or page state
Function `AutoBrowserUseForward`(*action, params*)**:**
  if *action == "search"* **then**
    `Search`(*params.query*);
  else if *action == "go_to_url"* **then**
    `GoToUrl`(*params.url*);
  else if *action == "find_archive_url"* **then**
    `FindArchiveUrl`(*params.url, params.date*);
  else if *action == "click"* **then**
    `Click`(*params.element*);
  else if *action == "go_back"* **then**
    `GoBack`();
  else if *action == "input"* **then**
    `InputText`(*params.element, params.text*);
  else if *action == "pdf_viewer"* **then**
    `PdfViewer`(*params*);
  else if *action == "video_viewer"* **then**
    `VideoViewer`(*params*);
  else if *action == "scroll"* **then**
    `Scroll`(*params.amount*);
  else if *action == "extract_content"* **then**
    `ExtractContent`(*params.goal*);
  else if *action == "open_tab"* **then**
    `OpenTab`(*params.url*);
  else if *action == "switch_tab"* **then**
    `SwitchTab`(*params.tab_id*);
  else if *action == "close_tab"* **then**
    `CloseTab`(*params.tab_id*);
  else
    return Error;

Function `Search`(*query*)**:**
  Search the web with the query.
Function `GoToUrl`(*url*)**:**
  Navigate to the given URL.
Function `FindArchiveUrl`(*url, date*)**:**
  Find the archive URL for the given URL and date.
Function `Click`(*element*)**:**
  Click on the specified page element.
Function `GoBack`()**:**
  Go back to the previous page.
Function `InputText`(*element, text*)**:**
  Input text into the specified field.
Function `PdfViewer`(*params*)**:**
  Interact with the PDF (e.g., scroll, jump, search, etc.).
Function `VideoViewer`(*params*)**:**
  Interact with the video (e.g., play, pause, seek, etc.).
Function `Scroll`(*amount*)**:**
  Scroll the page by the specified amount.
Function `ExtractContent`(*goal*)**:**
  Extract content from the page for the given goal.
Function `OpenTab`(*url*)**:**
  Open a new tab with the specified URL.
Function `SwitchTab`(*tab_id*)**:**
  Switch to the tab with the given ID.
Function `CloseTab`(*tab_id*)**:**
  Close the tab with the given ID.

The DeepAnalyzerTool pseudocode outlines a systematic analysis process that can handle both tasks and external sources. It distributes the analysis across multiple models, integrates their outputs, and summarizes the results to deliver a comprehensive answer or report.

Algorithm 4: Pseudocode of DeepAnalyzerTool

Input: Task description `task` (optional), source file or uri `source` (optional)
Output: Final comprehensive analysis and summary
Function `DeepAnalyzerForward(`*task, source*`):`
    if *task* == *None* ***and*** *source* == *None* **then**
        return Error ("At least one of task or source must be provided");
    foreach *model* ∈ *analyzer_models* **do**
        analysis[model] ← `Analyze(`*model, task, source*`)`;
    summary ← `Summarize(`*summary_model, analysis*`)`;
    return Formatted output of analysis and summary;
Function `Analyze(`*model, task, source*`):`
    if *task is None* **then**
        task ← "Please write a detailed caption for the attached file or uri.";
    if *source is image* **then**
        Append image to input;
    else if *source is file* **then**
        Extract file content and append to input;
    Send input to model and return response;
Function `Summarize(`*summary_model, analysis*`):`
    Format step-by-step comparison and summary prompt with all model outputs;
    Send to summary_model and return final summary;