

# AutoAgent: A Fully-Automated and Zero-Code Framework for LLM Agents

Jiabin Tang Tianyu Fan Chao Huang\*

The University of Hong Kong

{jiabintang77, tianyufan0504, chaohuang75}@gmail.com

🔗 Source Code: <https://github.com/HKUDS/AutoAgent>

## Abstract

Large Language Model (LLM) Agents have demonstrated remarkable capabilities in task automation and intelligent decision-making, driving the widespread adoption of agent development frameworks such as LangChain and AutoGen. However, these frameworks predominantly serve developers with extensive technical expertise—a significant limitation considering that only 0.03% of the global population possesses the necessary programming skills. This stark accessibility gap raises a fundamental question: *Can we enable everyone, regardless of technical background, to build their own LLM agents using natural language alone?* To address this challenge, we introduce AutoAgent - a **Fully-Automated** and highly **Self-Developing** framework that enables users to create and deploy LLM agents through **Natural Language Alone**. Operating as an autonomous Agent Operating System, AutoAgent comprises four key components: i) Agentic System Utilities, ii) LLM-powered Actionable Engine, iii) Self-Managing File System, and iv) Self-Play Agent Customization module. This lightweight yet powerful system enables efficient and dynamic creation and modification of tools, agents, and workflows without coding requirements or manual intervention. Beyond its code-free agent development capabilities, AutoAgent also serves as a versatile multi-agent system for **General AI Assistants**. Comprehensive evaluations on the GAIA benchmark demonstrate AutoAgent’s effectiveness in generalist multi-agent tasks, surpassing existing state-of-the-art methods. Furthermore, AutoAgent’s Retrieval-Augmented Generation (RAG)-related capabilities have shown consistently superior performance compared to many alternative LLM-based solutions. Code link: <https://anonymous.4open.science/r/AutoAgent/>.

## 1 Introduction

The emergence of Large Language Models (LLMs) has revolutionized AI agent development, enabling unprecedented breakthroughs in autonomous task execution and intelligent problem-solving. LLM-powered agents excel at understanding context, making informed decisions, and seamlessly integrating with various tools and APIs. Leading frameworks like LangChain (LangChain, 2023), AutoGPT (Significant-Gravitas, 2023), AutoGen (Wu et al., 2023), CAMEL (Li et al., 2023), and MetaGPT (Hong et al., 2024a) have demonstrated remarkable success in automating increasingly complex workflows - from sophisticated web navigation to advanced data analysis and innovative creative content production. By leveraging advanced mechanisms such as role-playing, structured operating procedures, and dynamic agent coordination, these frameworks deliver exceptional problem-solving capabilities while significantly reducing human intervention.

Despite remarkable advancements in AI agent development, a significant barrier persists: the creation and optimization of LLM agent systems remains dependent on traditional programming expertise. Current frameworks primarily cater to technically proficient developers who can navigate complex codebases, understand API integrations, and implement sophisticated prompt engineering patterns. This reliance on coding skills creates a substantial accessibility gap, as only 0.03% of the global population possesses the necessary programming expertise to effectively build and customize these agents. Even with well-documented frameworks and development tools, the entry barrier remains dauntingly high for non-technical users. This limitation becomes particularly problematic given the universal need for personalized AI assistants in digital age. Everyone, from business professionals seeking workflow automation to educators designing interactive learning tools, requires customized

\*Chao Huang is the Corresponding Author.

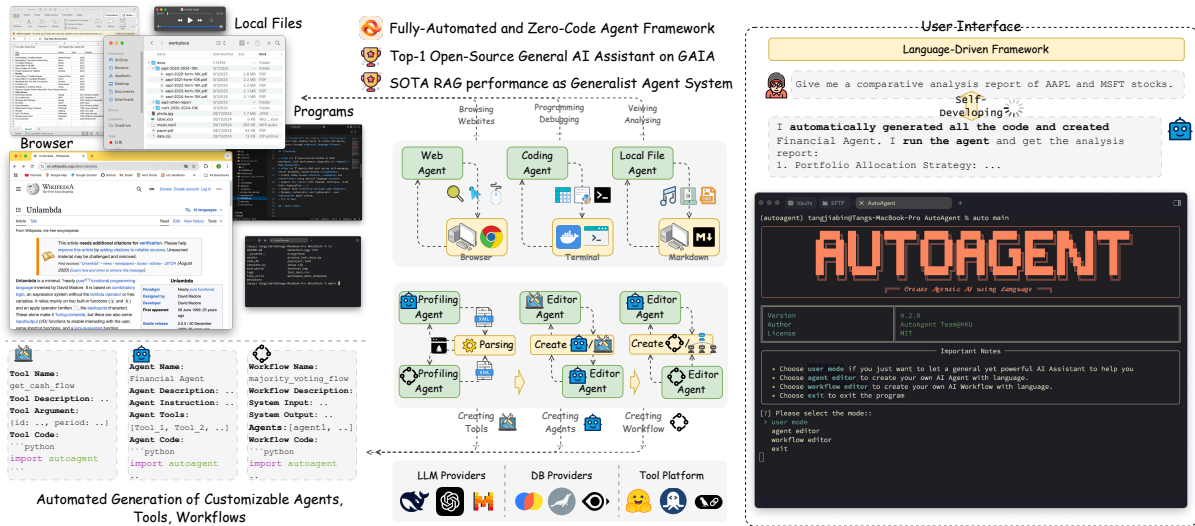


Figure 1: AutoAgent stands out as a new LLM Agent Framework that enables fully automated, zero-code development for complex task automation. Ranking #1 among open-source solutions on the GAIA benchmark, it delivers state-of-the-art RAG performance as a general AI assistant. Its revolutionary approach democratizes AI development - allowing anyone, regardless of coding experience, to create and customize their own agents, tools, and workflows with ease.

LLM agents tailored to their specific needs. For instance, a researcher might need an agent specialized in literature review and data analysis, while a content creator might require an agent focused on creative writing and media management. The current paradigm of coding-dependent agent development not only severely restricts the user base but also creates a bottleneck in meeting the diverse and evolving demands for personalized AI assistance. This misalignment between universal needs and limited accessibility calls for a fundamental rethinking of how LLM agents are created and customized.

This gap between universal needs and limited accessibility motivates a key question: *Can we democratize LLM agent development via Natural Language-based Creation and Customization?* We pursue this goal with AutoAgent, a framework that reframes agent development as a fully automated, language-driven process requiring zero programming. AutoAgent acts as an autonomous Agent Operating System with three core capabilities: 1) **Natural Language-Driven Multi-Agent Building**—automatically constructing and orchestrating agent systems through dialogue, without coding or manual configuration; 2) **Self-Managing Workflow Generation**—dynamically generating, optimizing, and adapting workflows from high-level task descriptions when users cannot specify details; and 3) **Intelligent Resource Orchestration**—offering unified natural-language access to tools, APIs, and compute while automatically man-

aging allocation and optimization. Together, these features democratize LLM agent development with enterprise-grade sophistication, turning complex engineering into an intuitive conversation for all users.

To enable fully-automated and zero-code LLM agent development, AutoAgent introduces several synergistic technical innovations that form a complete framework: First, the **Agentic System Utilities** provides a foundational multi-agent architecture, where specialized web, code, and file agents collaborate seamlessly to handle diverse real-world tasks. At its core, the **LLM-powered Actionable Engine** serves as the system's brain, supporting flexible integration of any LLM provider through both direct and transformed tool-use paradigms for robust action generation. To address the critical challenge of information management, the **Self-Managing File System** enhances overall system capability by automatically converting diverse data formats into queryable vector databases, enabling efficient information access across all operations. Additionally, the **Self-Play Agent Customization** not only transforms natural language requirements into executable agents through structured XML schemas, but also automatically generates optimized workflows through iterative self-improvement, eliminating the need for manual agent programming or workflow design. Together, these innovations enable AutoAgent to democratize agent development while maintaining production-level robustness.

AutoAgent’s exceptional capabilities have been rigorously validated through comprehensive empirical evaluation. In standardized benchmarks, it secured a strong second place on the Generalist Agent Benchmark (GAIA), while significantly outperforming state-of-the-art RAG approaches on the Retrieval-Augmented Generation benchmark. Beyond these quantitative achievements, extensive case studies demonstrated AutoAgent’s robust self-development capabilities across diverse real-world scenarios, highlighting its practical value in automated agent development.

## 2 Related Work and Preliminaries

LLM-empowered agents have transformed AI systems via tool invocation. Frameworks such as LangChain (LangChain, 2023), AutoGPT (Significant-Gravitas, 2023), CAMEL (Li et al., 2023), MetaGPT (Hong et al., 2024a), and OpenAgent (Xie et al., 2023) showcase diverse strengths: CAMEL introduced role-playing communication, AutoGen (Wu et al., 2023) explored LLM collaborations for problem-solving, MetaGPT (Hong et al., 2024a) incorporated Standardized Operating Procedures, and OpenAgent (Xie et al., 2023) offered specialized web agents. However, these frameworks often demand coding skills and domain expertise, limiting access for non-technical users. We propose a new paradigm that democratizes agent development through natural language, enabling broad access regardless of technical background.

**LLM-Empowered Agent.** The task-solving process of Large Language Model (LLM) agents can be formalized as a Markov Decision Process (MDP) to model environment interaction. Defined as  $\mathcal{M} = (\mathcal{S}, \mathcal{A}, O, P(\cdot), \mathcal{E})$ , it specifies state space  $\mathcal{S}$ , action space  $\mathcal{A}$ , observation space  $O$ , transition function  $P(\cdot)$ , and environments  $\mathcal{E}$ . At each time step, the agent observes the state, selects an action via its policy, interacts with the environment, and updates its state (the agent “context”). State-to-action mapping typically follows **Tool-Use** (Yao et al., 2024), leveraging external capabilities, or **Re-Act** (Yao et al., 2023) (Non-tool-use), producing actions solely from the language model. This MDP view supports understanding, analysis, and design of LLM agents for complex multi-step tasks.

**Generalist Multi-Agent System.** Multi-agent systems (MAS) address single-agent limits in complex problems by combining multiple specialized agents. Generalist Multi-Agent Systems coordinate agent teams with an Orchestrator to solve diverse

tasks through collaboration.

In a Generalist MAS, there are multiple agents, denoted as  $\pi_0 : S_0 \rightarrow A_0, \pi_1 : S_1 \rightarrow A_1, \dots, \pi_n : S_n \rightarrow A_n$ . Each agent’s action set includes a special **transfer action**  $\hat{A}_i \in A_i$  for delegating tasks to other agents. The key challenge is an effective **Task Transfer Mechanism** that organizes agents via transfer actions. We refer to this organization as the “MAS Design Pattern”. A common design is the **Orchestrator-Workers** paradigm (Fourney et al., 2024; Anthropic, 2024), where the **Orchestrator** decomposes tasks and assigns subtasks to **Workers**, which execute and return results through transfer actions.

For tasks with deterministic steps, workflow-driven mechanisms are often effective. Examples include GPTSwarm (Zhuge et al.), which represents workflows as computational graphs for complex data operations, and systems for mathematical reasoning (Wang et al., 2023) and code generation (Wang et al., 2024a). These advances enable strong performance in GUI interaction (Hong et al., 2024b; Xu et al., 2024), software development (Wang et al., 2024b; Yang et al., 2024), web browsing (Drouin et al., 2024; Song et al., 2024), and embodied tasks (Li et al., 2024), highlighting the versatility of LLM agent systems.

**Workflow Design in Generalist MAS.** While Generalist Multi-Agent Systems are flexible for open-ended tasks, they can incur higher cost, complexity, and compounding errors (Anthropic, 2024). For deterministic processes with domain expertise, fixed workflows can be more stable.

A workflow in a MAS is defined as  $\mathcal{W} = w_{i,j} : \pi_i \xrightarrow{c_k} \pi_j$ , where  $\pi_i$  and  $\pi_j$  are agents and  $c_k$  is the transfer condition. Workflow design defines conditional transfer relations between agents to ensure coordinated execution—the “Workflow Design Pattern”. Common patterns include **Routing**, **Parallelization**, and **Evaluator-Optimizer** (Anthropic, 2024), supporting stable and efficient MAS execution.

**Fully-Automated Generalist MAS.** Designing and executing MAS and complex workflows typically requires substantial expertise; AutoAgent aims to fully automate this. The key challenge is bridging high-level user requirements to implementable MAS and workflow solutions through natural language interactions.

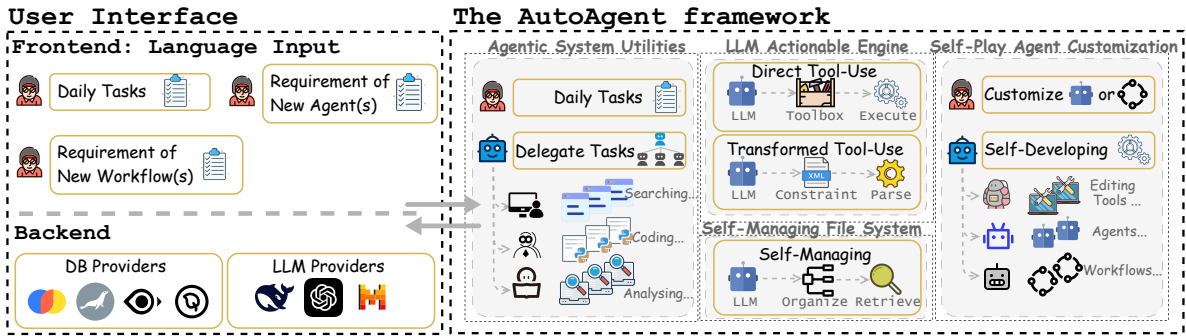


Figure 2: AutoAgent is a fully automated, language-driven generalist agent system. The core components that enable this include the Agentic System Utilities, the LLM-powered Actionable Engine, the Self-Managing File System, and the Self-Play Agent Customization module.

### 3 The AutoAgent Framework

AutoAgent is built as an automated operating system for LLM agents and general AI assistants. Inspired by modern OS design, it integrates core components for natural language-driven agent development and task execution (Fig 2). **Agentic System Utilities** supply foundational building blocks, the **LLM-powered Actionable Engine** acts as the central brain for understanding inputs and orchestrating multi-agent coordination, the **Self-Managing File System** enables structured storage and retrieval of user multi-modal data, and **Self-Play Agent Customization** lets users create tailored agents and workflows via natural language without coding. Together, these capabilities make AutoAgent a versatile platform for autonomous agent-based solutions across diverse applications.

#### 3.1 Agentic System Utilities

The AutoAgent framework adopts a modular multi-agent architecture to build intelligent personal assistant agents that integrate and coordinate diverse capabilities, including web browsing, information retrieval, data analysis, and code execution. It combines specialized agents (web, coding, file management) with an orchestrator that decomposes and delegates user requests, forming a versatile and extensible foundation that supports rapid development of tailored agent-driven solutions. Detailed system prompts and tool definitions for **Agentic System Utilities** are provided in Appendix Sec A.

##### 3.1.1 Orchestrator Agent

The Orchestrator Agent serves as the main user interface. It receives user tasks, understands them, decomposes them into sub-tasks, and delegates each to suitable sub-agents via handoff tools (OpenAI, 2024). Sub-agents return results through handoff, and the Orchestrator iteratively assigns subsequent sub-tasks based on progress until the overall task is completed. This handoff-based design provides a simple, effective alternative to complex planning

prompts.

##### 3.1.2 Dedicated Agent Module

The basic Agent system can be broadly categorized into three types: **web interaction, code execution, and file analysis**. We have constructed three functionally complementary dedicated agent modules: Web Agent, Coding Agent, and Local File Agent, each designed for these respective tasks.

**The Web Agent provides a flexible and extensible web-toolkit**, enabling the agent to perform a variety of web-based tasks, from general web searches to file downloads. Its core functionalities include web searching, page navigation, content browsing, and file downloading, which are abstracted into 10 high-level operational interfaces (e.g., `click`, `web_search`, `visit_url`, etc.). This module is built on BrowserGym (Drouin et al., 2024), creating a browser environment that combines low-level code-driven actions to implement high-level tools, thereby enhancing the extensibility of tool definitions.

**The Coding Agent is a comprehensive solution for code-related tasks**, capable of handling a wide range of applications—from data processing and numerical computation to machine learning and system management. It offers diverse tools and supports execution in a secure, isolated interactive terminal environment. All code execution results are fed back to the agent via terminal output, with pagination support (e.g., using commands like `terminal_page_up`, `terminal_page_down`, and `terminal_page_to`), effectively bypassing the context length limitations of LLMs. To ensure security, all operations run within a Docker sandbox, with optional integration of third-party secure execution platforms like E2B (E2B, 2024).

**The Local File Agent is designed to uniformly process various local multimodal file types** (e.g., text documents, audio, video, spreadsheets) and convert them into structured formats for analysis.

This agent standardizes different file formats (e.g., .pdf, .mp4, .csv, .docx) and displays them in a terminal-like Markdown browser with pagination, enabling efficient handling of long texts or complex file structures for everyday tasks.

These three agent modules work collaboratively, forming light-weight yet complete basic Agent System, laying the foundation for future functional expansion.

### 3.2 LLM-powered Actionable Engine

As the CPU executes instructions, manages resources, and coordinates processes in an OS, the LLM-powered actionable engine can understand natural language, generate plans, and coordinate tasks across agents. This enables seamless human-agent collaboration and task completion.

We utilize LiteLLM (BerriAI, 2024) to standardize LLM requests through an OpenAI-like interface, supporting 100+ models from various providers. For agent collaboration, the LLM receives all action-observation pairs up to time  $t$  as state  $s_t$  to determine the next action. These pairs serve as system RAM, facilitating efficient retrieval and enabling language-driven system coordination.

#### 3.2.1 Generating Actionable Reflections

We generate reflections (*i.e.*, actions) based on LLM context, which can be broadly categorized into two distinct approaches that leverage the language model’s capabilities.

**Direct Tool-Use Paradigm.** This approach is suitable for commercial LLMs or LLM serving platforms that support tool-use. These LLMs can directly generate a parsed next-step tool to execute based on the provided tool set and the current state, reducing errors during the tool parsing phase. However, this method heavily relies on the optimization of the third-party platform’s capabilities.

**Transformed Tool-Use Paradigm.** This approach does not rely on the LLM’s inherent tool-use capabilities. Leveraging the superior code-generation abilities of modern LLMs, we transform the tool-use paradigm into a structured XML code generation task, *e.g.*, `<function=function_name>  
<parameter=parameter_1>value_1</parameter>  
... </function>`. This structured output is then parsed to extract critical information like tool arguments and others. It improves the performance of commercial models with suboptimal tool-use capabilities and enables the integration of open-source LLMs into the system, providing greater flexibility and customization.

### 3.3 Self-Managing File System

The file system in AutoAgent is a vector database that LLM agents can retrieve and understand. In our design framework, users can upload text files in any format (e.g., .pdf, .doc, .txt) or compressed archives and folders containing any text files. The system tools in the file system automatically convert these files into a consistent text format and store them in a user-defined collection within the vector database (using the `save_raw_docs_to_vector_db` tool). This enables agents to self-manage their database memory and perform efficient and accurate retrieval and generation using tools like `query_db` and `answer_query`. The detailed definitions of the tools are presented in Tab 4.

### 3.4 Self-Play Agent Customization

To allow users to customize tools and agents for specific scenarios or build their own multi-agent systems and workflows, it is designed as a code-driven, controllable self-programming agent framework. By implementing constraints, error-handling mechanisms, and customized workflows, it enables controlled code generation, facilitating the creation of tools, agents, and workflows. The AutoAgent supports two distinct modes: agent creation without workflow and agent creation with workflow.

#### 3.4.1 Agent Creation without Workflow

Building effective multi-agent systems often requires domain expertise (e.g., financial regulations or healthcare protocols), which many users lack. For instance, automating investment portfolio management in finance demands knowledge of asset allocation, risk modeling, and regulatory compliance.

To mitigate this, AutoAgent offers a workflow-based mode that lets users create sophisticated agent systems with minimal domain knowledge. Users provide high-level specifications (e.g., agent name and a short description of desired functions), and AutoAgent automatically generates the agent(s) and workflow(s) to coordinate them, grounded in the framework’s current tools, agents, and workflows. The main steps are:

- **Analyze Requirements and Existing Components.** The system analyzes user needs against available tools and agents, using a profiling agent to assess current capabilities and reusable resources.
- **Analyze Tools and Structure Agents.** It decides whether new tools are needed, evaluates reuse of existing tools, and organizes collaboration among

agents to maximize resource use and ensure efficient system design.

•**Generate Detailed XML Agent Specifications.** It produces structured XML capturing agent requirements, including functionality, dependencies, and interactions, to support downstream agent generation.

**Optimized Tool Creation with Third-Party APIs.** The Tool Editor Agent integrates third-party APIs (e.g., LangChain, RapidAPI, Hugging Face) to build tools, retrieving relevant documentation (names, descriptions, invocation). The system currently supports 145 RapidAPI APIs across 8 categories, LangChain (LangChain, 2023), and models from 9 Hugging Face categories, with plans to add platforms like Composio (Composio, 2024).

It also generates tool code with automatic syntax checking, test design and execution, and iterative debugging until success, enabling a customizable tool set without a bloated integrated system.

**Agent Creation and Execution.** When requirements span multiple agents, the Agent Editor Agent performs multi-step creation. After agents are created, the system calls `create_orchestrator_agent` to build an orchestrator connecting them, following the Orchestrator-Workers pattern with prompts covering task description, subtask decomposition rules, and scenario details. Algorithms and prompts are in Appendix Sec A.6.1.

### 3.4.2 Agent Creation with Workflow

When users have explicit workflow requirements and domain knowledge, AutoAgent supports a tailored mode. Users describe the desired agent(s) and specify target tasks, and AutoAgent generates both the agent(s) and workflow(s) to coordinate collaboration toward these objectives.

Traditional graph-based approaches require strict graph-theoretic structures (Zhuge et al., 2024; LangChain, 2024; Hu et al., 2024; Zhang et al., 2024), which can be difficult for LLMs to generate reliably. Instead, AutoAgent uses an event-driven design that treats each agent’s task-solving process as an event. With event listening and triggering, agents collaborate flexibly without rigid graph constraints.

**Constructing New Workflows.** Creating a new workflow is itself a multi-agent process. The Workflow Form Agent analyzes requirements and existing tools/agents to decide whether new agents are needed, which agents compose the workflow, and the event listening/triggering logic. It then outputs

Table 1: Performance comparison between the baseline models and our AutoAgent on the GAIA benchmark. The results we report are those published on the GAIA.

Agent Name	Avg.	L1	L2	L3
TapeAgent v0.1	33.94	47.17	34.88	3.85
FRIDAY	34.55	45.28	34.88	11.54
Magentic-1	36.97	54.72	33.72	11.54
AgentIM	37.58	50.94	36.05	15.38
Multi-Agent Exp v0.1	39.39	54.72	38.37	11.54
AgentIM v1.1	40.00	50.94	40.70	15.38
Trase Agent	40.00	47.17	40.70	23.08
HuggingFace Agents	44.24	58.49	43.02	19.23
Magentic-1 (o1)	46.06	56.60	46.51	23.08
omne	46.06	60.38	44.19	23.08
Trase Agent v0.2	47.27	58.49	46.51	26.92
Barcelona v0.1	50.30	62.26	50.00	26.92
Langfun Agent v2.0	54.55	60.38	59.30	26.92
h2oGPTe Agent v1.6.8	63.64	67.92	67.44	42.31
<b>AutoAgent</b>	55.15	71.70	53.49	26.92

structured XML.

During parsing, an error detection mechanism checks whether the workflow form satisfies system constraints (e.g., on `on_start`). If violated, detailed errors are returned to the Workflow Form Agent for regeneration. If valid, the form is sent to the Workflow Editor Agent, which creates agents if needed, builds the workflow, and executes it on the task. Algorithms and prompts are in Appendix Sec A.6.2.

## 4 Evaluation

### 4.1 Evaluation for a Generalist Agent System Dataset and Evaluation Protocols.

The GAIA benchmark (Mialon et al., 2024) evaluates General AI Assistants through 466 test and 165 validation questions across 3 difficulty levels, assessing Reasoning, Multi-Modality Handling, Web Browsing, and Tool-Use Proficiency. We evaluated AutoAgent on GAIA’s validation set using success rate as the metric, measuring task completion performance on human-like challenges.

**Baseline Methods.** The baselines we selected are divided into two categories: Open-Source: FRIDAY (Wu et al., 2024), Magentic-1 (Fourney et al., 2024), Multi-Agent Experiment v0.1 (powered by AutoGen)(Microsoft, 2024), HuggingFace Agents(HuggingFace, 2024), Langfun Agent (Google, 2024); Closed-Source: TapeAgent, AgentIM, Trase Agent (Trase, 2024), Omne, Barcelona<sup>1</sup>, and the h2oGPTe Agent (H2O.ai, 2024). These diverse baselines represent the current state-of-the-art in open-source and proprietary multi-agent systems, providing a comprehensive landscape for evaluating the performance and capa-

<sup>1</sup>TapeAgent, AgentIM, Omne, and Barcelona are anonymous.

bilities of our proposed AutoAgent framework.

**Implementation Details.** To address tasks in the GAIA benchmark, we utilize a combination of the System Utilities of the Model and the Tool Editor Agent from the Agentic-SDK. The basic agents first attempt to complete the task while collecting relevant information and reflections. If successful, the result is directly returned. If not, the Tool Editor Agent creates new tools to continue the task. During validation, Claude-Sonnet-3.5 is used by default.

**Evaluation Results and Analysis.** The results in Table 1 reveal the following key observations:

- **Obs.1. Overall Superiority of AutoAgent:** Our method significantly outperforms all open-source agent systems and achieves performance close to the latest agent system, h2oGPTe Agent v1.6.8 (submitted on December 16, 2024), securing a stable position in the top 2 rankings. Notably, our approach demonstrates superior performance on Level 1 tasks compared to all state-of-the-art baselines, becoming the first method to achieve over 70% accuracy rate. This success is attributed to the well-designed System Utilities and the stable interaction of basic agents with the environment, enabling efficient solutions to everyday simple tasks.

- **Obs.2. Effectiveness of Key Components:** Specifically, our framework demonstrates significantly superior performance compared to Magentic-1 (Fourney et al., 2024), a recent representative open-source MAS, and FRIDAY, a classic self-improved framework. While Magentic-1 leverages the powerful reasoning capabilities of o1-preview to design complex Orchestrator Agent (also the Coder Agent), our framework emphasizes the stability of interactions between sub-agents and their respective environments, as well as the precision of tool definitions. Under these conditions, the Orchestrator Agent achieves better results with simple prompts and handoff tools.

## 4.2 Evaluation of AutoAgent on the

**Retrieval-Augmented Generation Task Benchmark Dataset and Evaluation Protocols.** To test the basic functionalities of the AutoAgent, we use the RAG task as the testing benchmark. MultiHop-RAG (Tang and Yang, 2024) is a dataset designed to evaluate RAG capabilities, requiring the RAG methods to gather information from multiple sources and generate responses, which aligns with the file functionality logic of AutoAgent. We evaluate using two metrics: **Accuracy (Acc)** measures response consistency with expected answers

(e.g., “ChatGPT” or “OpenAI’s ChatGPT” are both correct for “Which AI tool reached 100M daily users in March?”). **Error (Err)** counts confident but incorrect responses (e.g., answering “Bard” to the above query).

**Baseline Methods.** The baselines represent a diverse range of LLM-based RAG techniques. The chunk methods, such as NaiveRAG (Mao et al., 2020) and HyDE (Gao et al., 2022), utilize the original text segmentation. The graph methods, including MiniRAG (Fan et al., 2025) and LightRAG (Guo et al., 2024), manage files as sophisticated graphs. In contrast, Langchain’s Agentic RAG (LangChain, 2023) innovatively accesses files through intelligent software agents. These baselines cover a wide array of strategies for leveraging large language models to retrieve and generate robust responses.

**Implementation Details.** We used gpt-4o-mini (OpenAI, 2023) as the LLM and text-embedding-3-small for embeddings. We followed MultiHopRAG (Tang and Yang, 2024) for text chunking, with 256-token chunks and top-6 retrieval. This leverages the gpt-4o-mini’s language abilities while text-embedding-3-small provides retrieval, with MultiHopRAG’s chunking managing information effectively.

Table 2: Evaluation of AutoAgent for RAG.

Method	<i>acc</i>	<i>err</i>	<i>acc</i>	<i>err</i>
Chunk-Based	NaiveRAG		HyDE	
	53.36%	12.28%	56.59%	16.55%
Graph-Based	MiniRAG		LightRAG	
	57.81%	34.78%	58.18%	35.40%
Agent-Based	Langchain		<b>AutoAgent</b>	
	62.83%	20.50%	<b>73.51%</b>	14.20%

**Evaluation Results and Analysis.** We summarize the key observations from Table 2.

- **Superior Performance of AutoAgent.** The results clearly demonstrate the superior performance of our proposed AutoAgent model compared to other baselines on the Multihop-RAG task. By leveraging a more flexible and adaptive agent-based framework, AutoAgent is able to dynamically orchestrate the retrieval and reasoning process, outperforming even other baselines.

- **AutoAgent vs. LangChain.** Our method significantly outperforms LangChain, which is also an agentic RAG. This is due to AutoAgent’s more flexible framework, where agents do not need to rely on predefined workflows and tools to execute file search tasks. The proposed model can orchestrate workflows on the fly during the search process, leading to more efficient and accurate results.

### 4.3 AutoAgent’s Performance on Open-Ended Tasks

This section thoroughly explores the capabilities of the AutoAgent framework in generating agents and workflows based on even vague, natural language inputs across various scenarios. To illustrate the breadth of AutoAgent’s abilities, we will examine its performance on tasks of varying difficulty - from the creation of a single agent to the orchestration of multiple, coordinated agents. (We put the results of single agent in Appendix A.7.1)

**Task with Multi-Agents.** To further validate AutoAgent’s capability to generate agents and integrate third-party tools, we tasked it with creating a Financial Agent based on the following requirements:

- I want to create `Financial Agent` that can help me to do two kinds of tasks:
1. Manage the private financial docs. I have a folder that contain the financial docs in my local machine, and I want to help me to manage them.
  2. Search the financial information online. You may help me to:
    - get balance sheets for a given ticker over a given period.
    - get cash flow statements for a given ticker over a given period.
    - get income statements for a given ticker over a given period.

#### Building a Comprehensive Financial Agent.

The Agent Form Agent created two agents: the **Document Manager Agent** and **Market Research Agent**, with XML structure shown in List 18. The Tool Editor Agent created tools: `get_balance_sheet`, `get_cash_flow`, `get_income_statement`, and `analyze_financial_data`. The Agent Editor Agent then composed these agents and established a Financial Analysis Orchestrator for coordination.

The Financial Analysis Orchestrator leveraged new tools and existing capabilities to conduct research on local documents and external data sources. This process produced a comprehensive research report, shown in List 19 in the Appendix. The agent trajectory is detailed in Tab 6 in the Appendix. Despite encountering a `SyntaxError` during initial creation, the Agent Editor successfully self-debugged and completed the task, demonstrating the robustness of AutoAgent system.

**Workflow Generation.** Scaling Test-Time Compute has been validated as a superior approach for solving reasoning problems. However, manually constructing workflows poses a high barrier to entry. We aim to explore whether AutoAgent’s auto-

Table 3: Comparison between single LLMs and the AI-generated Majority Voting workflow.

Models	gpt-4o 0806	claude-3.5-sonnet 1022	deepseek-v3	Majority Voting Workflow (3 models)
pass@1	66.4	66.4	74.2	<b>75.6</b>

matic creation of agents and workflows can bridge the gap between the idea of Test-Time Compute and the implementation of workflows. Taking the majority voting method with multiple models as an example:

I want to create a workflow that can help me to solving the math problem.

The workflow should:

1. Parallelize solving the math problem with the same `Math Solver Agent` using different language models (`gpt-4o`, `claude-3-5-sonnet`, `deepseek-chat`)
2. Aggregate the results from the `Math Solver Agent` and return the final result using majority voting.

**Potential Test-Time Scaling Law.** Upon receiving the requirements, the Workflow Form Agent generated an XML-formatted workflow table (List 20). This table includes two new agents: **Math Solver Agent** and **Vote Aggregator Agent**. After validation, the Agent Editor Agent created agents. The Workflow Editor Agent then constructed a new workflow based on the form and conducted tests. To validate the workflow’s practicality, we performed comparative experiments on the MATH-500 dataset (Lightman et al., 2024) using 3 LLMs (gpt-4o-20240806, claude-3.5-sonnet-20241022, deepseek-v3) and a Majority Voting workflow. As shown in Tab 3, the generated workflow performs significantly better than state-of-the-art baselines. We selected cases from deepseek-v3 (Tab 5) where AutoAgent’s workflow effectively corrected errors through multi-model collaboration, demonstrating its potential to establish scaling laws in LLMs.

## 5 Conclusion

The AutoAgent framework advances the democratization of LLM-powered agents, making them accessible to the non-programming majority. By linking high-level natural language requirements to practical multi-agent systems and workflows, AutoAgent enables users to create, customize, and deploy agents, tools, and workflows without substantial technical expertise. Its modular architecture, Agentic System Utilities, and LLM-powered Actionable Engine together support automated agent development and task execution. Features such as the Self-Organizing File System and Self-Play Agent Customization further strengthen AutoAgent, enabling dynamic agent evolution and task-specific optimization.

## 6 Limitations

While AutoAgent presents a promising step toward democratizing LLM agent development through natural language alone, we identify several limitations that open compelling avenues for future exploration:

**Lack of Systematic Evaluation for Natural Language Agent Programming.** Although AutoAgent demonstrates strong capabilities in zero-code agent construction and self-customization, it currently lacks a systematic evaluation framework tailored to this novel paradigm. Existing benchmarks often focus on task execution performance, but do not adequately capture the effectiveness, reliability, and user experience of natural language-based agent programming. Future work could explore the development of a standardized evaluation suite that measures usability, correctness, adaptability, and end-user satisfaction in natural language-driven agent creation scenarios.

**Absence of GUI-based Agent Support.** While AutoAgent effectively supports tool invocation and DOM-level interactions in web environments, it does not currently incorporate GUI-based agent capabilities. This limits its robustness in handling visually complex or dynamic web applications where visual cues and spatial layout play an essential role. Incorporating GUI agents—or hybrid agents that combine DOM and visual understanding—could enhance agent generalization and resilience, particularly in real-world human-computer interaction settings.

## References

- Anthropic. 2024. **Building effective agents**. <https://www.anthropic.com/research/building-effective-agents>.
- BerriAI. 2024. **Litellm: Proxy server (llm gateway) to call 100+ llm apis in openai format**. <https://github.com/BerriAI/litellm>.
- Composio. 2024. **Composio: Production ready toolset for ai agents**. <https://github.com/ComposioHQ/composio>.
- Alexandre Drouin, Maxime Gasse, Massimo Caccia, Issam H. Laradji, Manuel Del Verme, Tom Marty, David Vazquez, Nicolas Chapados, and Alexandre Lacoste. 2024. **WorkArena: How capable are web agents at solving common knowledge work tasks?** In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 11642–11662. PMLR.
- E2B. 2024. **Secure open source cloud runtime for ai apps & ai agents**. <https://github.com/e2b-dev/e2b>.
- Tianyu Fan, Jingyuan Wang, Xubin Ren, and Chao Huang. 2025. **Minirag: Towards extremely simple retrieval-augmented generation**. *arXiv preprint arXiv:2501.06713*.
- Adam Fourney, Gagan Bansal, Hussein Mozannar, Cheng Tan, Eduardo Salinas, Erkang Zhu, Friederike Niedtner, Grace Proebsting, Griffin Bassman, Jack Gerrits, Jacob Alber, Peter Chang, Ricky Loynd, Robert West, Victor Dibia, Ahmed Awadallah, Ece Kamar, Rafah Hosn, and Saleema Amershi. 2024. **Magentic-one: A generalist multi-agent system for solving complex tasks**. *CoRR*, abs/2411.04468.
- Luyu Gao, Xueguang Ma, Jimmy Lin, and Jamie Callan. 2022. **Precise zero-shot dense retrieval without relevance labels**. *arXiv preprint arXiv:2212.10496*.
- Google. 2024. **Langfun: Oo for llms**. <https://github.com/google/langfun>.
- Zirui Guo, Lianghao Xia, Yanhua Yu, Tu Ao, and Chao Huang. 2024. **Lightrag: Simple and fast retrieval-augmented generation**.
- H2O.ai. 2024. **Autonomous agentic ai: execute multi-step workflows autonomously**. <https://h2o.ai/platform/enterprise-h2ogpte/#AgenticAI>.
- Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. 2024a. **MetaGPT: Meta programming for a multi-agent collaborative framework**. In *The Twelfth International Conference on Learning Representations*.
- Wenyi Hong, Weihang Wang, Qingsong Lv, Jiazheng Xu, Wenmeng Yu, Junhui Ji, Yan Wang, Zihan Wang, Yuxiao Dong, Ming Ding, and Jie Tang. 2024b. **Co-gagent: A visual language model for GUI agents**. In *CVPR*, pages 14281–14290. IEEE.
- Shengran Hu, Cong Lu, and Jeff Clune. 2024. **Automated design of agentic systems**. *CoRR*, abs/2408.08435.
- HuggingFace. 2024. **Transformers documentation: Agents and toos**. <https://huggingface.co/docs/transformers/agents>.
- LangChain. 2023. **Langchain: Build context-aware reasoning applications**. <https://github.com/langchain-ai/langchain>.
- LangChain. 2024. **Langgraph: sbuild resilient language agents as graphs**. <https://github.com/langchain-ai/langgraph>.
- Guohao Li, Hasan Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. 2023. **CAMEL: communicative agents for "mind" exploration of large language model society**. In *NeurIPS*.
- Manling Li, Shiyu Zhao, Qineng Wang, Kangrui Wang, Yu Zhou, Sanjana Srivastava, Cem Gokmen, Tony Lee, Li Erran Li, Ruohan Zhang, and 1 others. 2024. **Embodied agent interface: Benchmarking llms for embodied decision making**. In *NeurIPS 2024*.
- Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2024. **Let's verify step by step**. In *ICLR*. OpenReview.net.
- Yuning Mao, Pengcheng He, Xiaodong Liu, Yelong Shen, Jianfeng Gao, Jiawei Han, and Weizhu Chen. 2020. **Generation-augmented retrieval for open-domain question answering**. *arXiv preprint arXiv:2009.08553*.
- Grégoire Mialon, Clémentine Fourrier, Thomas Wolf, Yann LeCun, and Thomas Scialom. 2024. **GAIA: a benchmark for general AI assistants**. In *ICLR*. OpenReview.net.
- Microsoft. 2024. **Multi-agent experiment v0.1 msa ai frontiers (autogen team members)**. [https://aka.ms/gaia\\_multiagent\\_v01\\_march\\_1st](https://aka.ms/gaia_multiagent_v01_march_1st).
- OpenAI. 2023. **Gpt-4 technical report**. *Preprint*, arXiv:2303.08774.
- OpenAI. 2024. **Educational framework exploring ergonomic, lightweight multi-agent orchestration**. <https://github.com/openai/swarm>.
- Significant-Gravitas. 2023. **Autogpt**. <https://github.com/Significant-Gravitas/AutoGPT>.
- Yueqi Song, Frank Xu, Shuyan Zhou, and Graham Neubig. 2024. **Beyond browsing: Api-based web agents**. *arXiv preprint arXiv:2410.16464*.

- Yixuan Tang and Yi Yang. 2024. Multihop-rag: Benchmarking retrieval-augmented generation for multihop queries. *CoRR*, abs/2401.15391.
- Trase. 2024. Meet trase systems, the ai agent platform. <https://www.trasesystems.com/>. Accessed: 2025-01-15.
- Evan Wang, Federico Cassano, Catherine Wu, Yunfeng Bai, Will Song, Vaskar Nath, Ziwen Han, Sean Hendryx, Summer Yue, and Hugh Zhang. 2024a. Planning in natural language improves LLM search for code generation. *CoRR*, abs/2409.03733.
- Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, Hoang H. Tran, Fuqiang Li, Ren Ma, Mingzhang Zheng, Bill Qian, Yanjun Shao, Niklas Muennighoff, Yizhe Zhang, Binyuan Hui, and 5 others. 2024b. *OpenHands: An Open Platform for AI Software Developers as Generalist Agents*. *Preprint*, arXiv:2407.16741.
- Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. Self-consistency improves chain of thought reasoning in language models. In *ICLR*. OpenReview.net.
- Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Shaokun Zhang, Erkang Zhu, Beibin Li, Li Jiang, Xiaoyun Zhang, and Chi Wang. 2023. Autogen: Enabling next-gen LLM applications via multi-agent conversation framework. *CoRR*, abs/2308.08155.
- Zhiyong Wu, Chengcheng Han, Zichen Ding, Zhenmin Weng, Zhoumianze Liu, Shunyu Yao, Tao Yu, and Lingpeng Kong. 2024. Os-copilot: Towards generalist computer agents with self-improvement. *arXiv preprint arXiv:2402.07456*.
- Tianbao Xie, Fan Zhou, Zhoujun Cheng, Peng Shi, Luoxuan Weng, Yitao Liu, Toh Jing Hua, Junning Zhao, Qian Liu, Che Liu, Leo Z. Liu, Yiheng Xu, Hongjin Su, Dongchan Shin, Caiming Xiong, and Tao Yu. 2023. *Openagents: An open platform for language agents in the wild*. *Preprint*, arXiv:2310.10634.
- Yiheng Xu, Zekun Wang, Junli Wang, Dunjie Lu, Tianbao Xie, Amrita Saha, Doyen Sahoo, Tao Yu, and Caiming Xiong. 2024. *Aguvis: Unified pure vision agents for autonomous gui interaction*.
- John Yang, Carlos E Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik R Narasimhan, and Ofir Press. 2024. *SWE-agent: Agent-computer interfaces enable automated software engineering*. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.
- Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. 2024.  $\tau$ -bench: A benchmark for tool-agent-user interaction in real-world domains. *CoRR*, abs/2406.12045.
- Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. 2023. React: Synergizing reasoning and acting in language models. In *ICLR*. OpenReview.net.
- Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xionghui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, Bingnan Zheng, Bang Liu, Yuyu Luo, and Chenglin Wu. 2024. Aflow: Automating agentic workflow generation. *CoRR*, abs/2410.10762.
- Mingchen Zhuge, Wenyi Wang, Louis Kirsch, Francesco Faccio, Dmitrii Khizbullin, and Jürgen Schmidhuber. Gptswarm: Language agents as optimizable graphs. In *Forty-first International Conference on Machine Learning*.
- Mingchen Zhuge, Wenyi Wang, Louis Kirsch, Francesco Faccio, Dmitrii Khizbullin, and Jürgen Schmidhuber. 2024. Gptswarm: Language agents as optimizable graphs. In *ICML*. OpenReview.net.

## **A Appendix**

In the supplementary materials, we provide a detailed technical description of the 'Agentic System Utilities' implementation within our AutoAgent framework.

### **A.1 System-level Tools**

To empower our diverse array of system-level agents, we have carefully curated and predefined seven distinct categories of powerful tools. These tools span a wide range of functionalities, including coding, web browsing, file management, creating new tools, agents, and workflows, as well as natural language question answering for documents. The detailed names and comprehensive descriptions of these versatile tools are presented in [Table 4](#).

Table 4: List of detailed information of system-level tools.

<b>Tool Name</b>	<b>Category</b>	<b>Description</b>
run_python	Coding	Run a python script.
execute_command	Coding	Execute a command in the system shell. Use this function when there is a need to run a system command, and execute programs.
gen_code_tree_structure	Coding	Generate a tree structure of the code in the specified directory. Use this function when you need to know the overview of the codebase and want to generate a tree structure of the codebase.
create_directory	Coding	Create a directory if it does not exist. Use this function when there is a need to create a new directory.
list_files	Coding	List all files and directories under the given path if it is a directory. Use this function when there is a need to list the contents of a directory.
write_file	Coding	Write content to a file. Use this function when there is a need to write content to an existing file.
create_file	Coding	Create a file with the given path and content. Use this function when there is a need to create a new file with initial content.
read_file	Coding	Read the contents of a file and return it as a string. Use this function when there is a need to check an existing file.
terminal_page_to	Coding	Move the viewport to the specified page index. The index starts from 1. Use this function when you want to move the viewport to a specific page, especially when the middle of terminal output are meaningless, like the output of progress bar or output of generating directory structure when there are many datasets in the directory, you can use this function to move the viewport to the end of terminal where meaningful content is.
terminal_page_down	Coding	Scroll the viewport DOWN one page-length in the current terminal. Use this function when the terminal is too long and you want to scroll down to see the next content.

terminal_page_up	Coding	Scroll the viewport UP one page-length in the current terminal. Use this function when the terminal is too long and you want to scroll up to see the previous content.
input_text	Web	Types the given text value into the specified field.
click	Web	Clicks the mouse on the target with the given element bid.
page_down	Web	Scrolls the entire browser viewport one page DOWN towards the end.
page_up	Web	Scrolls the entire browser viewport one page UP towards the beginning.
history_back	Web	Navigates back one page in the browser's history. This is equivalent to clicking the browser back button.
history_forward	Web	Navigates forward one page in the browser's history. This is equivalent to clicking the browser forward button.
visit_url	Web	Navigate directly to a provided URL using the browser's address bar. Prefer this tool over other navigation techniques in cases where the user provides a fully-qualified URL (e.g., choose it over clicking links, or inputing queries into search boxes).
web_search	Web	Performs a web search on ' <a href="https://www.google.com.sg/?hl=en&amp;gl=US">https://www.google.com.sg/?hl=en&amp;gl=US</a> ' with the given query.
sleep	Web	Wait a short period of time. Call this function if the page has not yet fully loaded, or if it is determined that a small delay would increase the task's chances of success.
get_page_markdown	Web	Get the markdown content of the current page. Use this tool if you need to watch the Youtube video, Wikipedia page, or other pages that contain media content. Note that this tool can only be used after you have visited a valid page.
open_local_file	File	Open a local file at a path in the text-based browser and return current viewport content.
page_up_markdown	File	Scroll the viewport UP one page-length in the current file and return the new viewport content.
page_down_markdown	File	Scroll the viewport DOWN one page-length in the current file and return the new viewport content.

find_next	File	Scroll the viewport to next occurrence of the search string.
visual_question_answering	File	This tool is used to answer questions about attached images or videos.
find_on_page_ctrl_f	File	Scroll the viewport to the first occurrence of the search string. This is equivalent to Ctrl+F.
list_tools	Tools Edit	List all plugin tools in the MetaChain.
create_tool	Tools Edit	Create a plugin tool.
delete_tool	Tools Edit	Delete a plugin tool.
run_tool	Tools Edit	Run a tool with the given code.
search_trending_models_on_huggingface	Tools Edit	Search trending models on Hugging Face. Use this tool when you want to create a tool that uses Hugging Face models, only support the following tags: ['audio-text-to-text', 'text-to-image', 'image-to-image', 'image-to-video', 'text-to-video', 'text-to-speech', 'text-to-audio', 'automatic-speech-recognition', 'audio-to-audio'].
get_hf_model_tools_doc	Tools Edit	Get the detailed information of a model on Hugging Face, such as the detailed usage of the model containing the model's README.md. You should use this tool after you have used 'search_trending_models_on_huggingface' to find the model you want to use.
get_api_plugin_tools_doc	Tools Edit	Retrieve satisfied tool documents based on the query text.
list_agents	Agents Edit	List all plugin agents in the MetaChain.
read_agents	Agents Edit	Get detailed information of plugin agents in the MetaChain.
delete_agent	Agents Edit	Delete a plugin agent.
run_agent	Agents Edit	Run a plugin agent.
create_agent	Agents Edit	Use this tool to create a new agent or modify an existing agent.

create_orchestrator_agent	Agents Edit	Use this tool to create a orchestrator agent for the given sub-agents. You MUST use this tool when you need to create TWO or MORE agents and regard them as a whole to complete a task.
create_workflow	Workflows Edit	Create a workflow.
list_workflows	Workflows Edit	List all workflows in the MetaChain.
run_workflow	Workflows Edit	Run a workflow.
save_raw_docs_to_vector_db	RAG	Save the raw documents to the vector database. The documents could be: - ANY text document with the extension of pdf, docx, txt, etc. - A zip file containing multiple text documents - a directory containing multiple text documents All documents will be converted to raw text format and saved to the vector database in the chunks of 4096 tokens.
query_db	RAG	Retrieve information from the database. Use this function when you need to search for information in the database.
modify_query	RAG	Modify the query based on what you know. Use this function when you need to modify the query to search for more relevant information.
answer_query	RAG	Answer the user query based on the supporting documents.
can_answer	RAG	Check if you have enough information to answer the user query.

## A.2 Web Agent

The specific tools and system prompt for implementing the Web Agent are as follows:

Listing 1: Tools of **Web Agent**

```
[click, page_down, page_up, history_back, history_forward, web_search, input_text, sleep, visit_url, get_page_markdown, transfer_back_to_orchestrate_agent]
```

Listing 2: System Prompt of **Web Agent**

```
Review the current state of the page and all
↳ other information to find the best
↳ possible next action to accomplish your
↳ goal. Your answer will be interpreted and
↳ executed by a program, make sure to follow
↳ the formatting instructions.
Note that if you want to analyze the YouTube
↳ video, Wikipedia page, or other pages that
↳ contain media content, or you just want to
↳ analyze the text content of the page in a
↳ more detailed way, you should use
↳ `get_page_markdown` tool to convert the
↳ page information to markdown text. And
↳ when browsing the web, if you have
↳ downloaded any files, the path of the
↳ downloaded files will be
↳ `/workplace/downloads`, and you CANNOT
↳ open the downloaded files directly, you
↳ should transfer back to the `System
↳ Orchestrate Agent`, and let `System
↳ Orchestrate Agent` to transfer to `Local
↳ File Agent` to open the downloaded files.
When you think you have completed the task
↳ the `System Orchestrate Agent` asked you
↳ to do, you should use
↳ `transfer_back_to_orchestrate_agent` to
↳ transfer the conversation back to the
↳ `System Orchestrate Agent`. And you should
↳ not stop to try to solve the user's
↳ request by transferring to `System
↳ Orchestrate Agent` only until the task is
↳ completed.
```

## A.3 Local File Agent

The Local File Agent is equipped with a tailored set of tools and system prompts to enable it to efficiently manage and interact with files and directories. This specialized toolkit includes:

Listing 3: Tools of **Local File Agent**

```
[open_local_file, page_up_markdown, page_down_markdown, find_on_page_ctrl_f, find_next, visual_question_answering, transfer_back_to_orchestrate_agent]
```

Listing 4: System Prompt of **Local File Agent**

```
You are a file surfer agent that can handle
↳ local files.
```

```
You can only access the files in the folder
↳ `/workplace` and when you want to open a
↳ file, you should use absolute path from
↳ root like `/workplace/...`.
```

```
Note that `open_local_file` can read a file
↳ as markdown text and ask questions about
↳ it. And `open_local_file` can handle the
↳ following file extensions: [".html",
↳ ".htm", ".xlsx", ".pptx", ".wav", ".mp3",
↳ ".flac", ".pdf", ".docx"], and all other
↳ types of text files.
```

```
But IT DOES NOT HANDLE IMAGES, you should use
↳ `visual_question_answering` to see the
↳ image.
```

```
If the converted markdown text has more than
↳ 1 page, you can use `page_up`,
↳ `page_down`, `find_on_page_ctrl_f`,
↳ `find_next` to navigate through the pages.
```

```
When you think you have completed the task
↳ the `System Orchestrate Agent` asked you
↳ to do, you should use
↳ `transfer_back_to_orchestrate_agent` to
↳ transfer the conversation back to the
↳ `System Orchestrate Agent`. And you should
↳ not stop to try to solve the user's
↳ request by transferring to `System
↳ Orchestrate Agent` only until the task is
↳ completed.
```

```
If you are unable to open the file, you can
↳ transfer the conversation back to the
↳ `System Orchestrate Agent`, and let the
↳ `Coding Agent` try to solve the problem by
↳ coding.
```

## A.4 Coding Agent

The specific tools and system prompts for implementing the Coding Agent are as follows:

Listing 5: Tools of **Coding Agent**

```
[gen_code_tree_structure, execute_command, read_file, create_file, write_file, list_files, create_directory, run_python, terminal_page_up, terminal_page_down, terminal_page_to, transfer_back_to_orchestrate_agent]
```

Listing 6: System Prompt of **Coding Agent**

```
You are a helpful programming assistant that
↳ can write and execute code. You are
↳ working in the folder: `/workplace`, and
↳ you can only access the files in this
↳ folder.
```

```
Your can leverage your capabilities by using
↳ the specific functions listed below:
```

1. Creating project structures based on the user requirement using function  
↳ `create\_directory`.
2. Writing clean, efficient, and well-documented code using function  
↳ `create\_file` and `write\_file`.

3. You must run python scripts using function ↪ ``run_python`` rather than using the ↪ ``execute_command`` function.
4. Exam the project to re-use the existing ↪ code snippets as much as possible, you may ↪ need to use functions like ``list_files``, ``read_file`` and ↪ ``write_file``.
5. Writing the code into the file when ↪ creating new files, do not create empty ↪ files.
6. Before you write code into the existing ↪ files, you should first read the file ↪ content using function ``read_file`` and ↪ reserve the original content as much as ↪ possible.
7. Decide whether the task requires execution ↪ and debugging before moving to the next or ↪ not.
8. Generate the commands to run and test the ↪ current task, and the dependencies list ↪ for this task.
9. You only write Python scripts, don't write ↪ Jupiter notebooks which require ↪ interactive execution.
10. Note that every path you read, write, or ↪ search should be the absolute path ↪ (starting with `"/"`).
11. If you should use programming other than ↪ Python, you should use the ``write_file`` ↪ function to write the code into a file, ↪ and then use the ``execute_command`` ↪ function to run the code.
12. If the terminal output is too long, you ↪ should use ``terminal_page_up`` to move the ↪ viewport up, ``terminal_page_down`` to move ↪ the viewport down, ``terminal_page_to`` to ↪ move the viewport to the specific page of ↪ terminal where the meaningful content is.

Note that you can use this agent to make ↪ complex computation, write a api request, ↪ and anything else that can be done by ↪ writing code.

When you think you have completed the task ↪ the ``System Orchestrate Agent`` asked you ↪ to do, you should use ↪ ``transfer_back_to_orchestrate_agent`` to ↪ transfer the conversation back to the ↪ ``System Orchestrate Agent``. And you should ↪ not stop to try to solve the user's ↪ request by transferring to ``System ↪ Orchestrate Agent`` only until the task is ↪ completed.

[IMPORTANT] You can only complete the task by ↪ coding. Talk is cheap, show me the code ↪ with tools.

`transfer_to_coding_agent``

### Listing 8: System Prompt of **Orchestrator Agent**

You are a helpful assistant that can help the ↪ user with their request.  
Based on the state of solving user's task,  
↪ your responsibility is to determine which  
↪ agent is best suited to handle the user's  
↪ request under the current context, and  
↪ transfer the conversation to that agent.  
↪ And you should not stop to try to solve  
↪ the user's request by transferring to  
↪ another agent only until the task is  
↪ completed.

There are three agents you can transfer to:  
1. use ``transfer_to_local_file_agent`` to  
↪ transfer to ``Local File Agent``, it can  
↪ help you to open any type of local files  
↪ and browse the content of them.  
2. use ``transfer_to_web_agent`` to transfer to  
↪ ``Web Agent``, it can help you to open any  
↪ website and browse any content on it.  
3. use ``transfer_to_coding_agent`` to transfer  
↪ to ``Coding Agent``, it can help you to  
↪ write code to solve the user's request,  
↪ especially some complex tasks.

## A.6 Detailed Implementation of “Self-Play Agent Customization” in AutoAgent

### A.6.1 Agent Creation without Workflow

The following details demonstrate the specific process of Agent Creation without Workflow (Alg 1), as well as the tools and system prompts used in the implementation of Agent Profiling Agent, Tool Editor Agent, and Agent Editor Agent.

## A.5 Orchestrator Agent

The specific tools and system prompt for implementing the Orchestrator Agent are as follows:

### Listing 7: Tools of **Orchestrator Agent**

`[transfer_to_local_file_agent,`  
`transfer_to_web_agent,`

---

**Algorithm 1** Controllable Workflow of Creating Tools and Agents

---

```
1: Input: requirements  $\mathcal{R}$ , existing tool set  $\mathcal{A}$ , existing agent set  $\pi$ , task  $\mathcal{T}$  (optional), Maximum
   iterations of attempts  $M$ .
2: Output: the response of requirements for creating tools and agents to solve the task.
3: AgentProfile = agent_profile_agent( $\mathcal{R}|\mathcal{A}, \pi$ )
4: ParsingResults0 = form_parsing_function(AgentProfile| $\mathcal{R}, \mathcal{A}, \pi$ )
5: for  $i = 1$  to  $M - 1$  do
6:   if ParsingResults $i-1$  is ‘Success’ then
7:     break
8:   else
9:     AgentProfile = agent_profile_agent( $\mathcal{R}, \text{ParsingResults}_{i-1}|\mathcal{A}, \pi$ )
10:    ParsingResults $i$  = form_parsing_function(AgentProfile| $\mathcal{R}, \mathcal{A}, \pi$ )
11:   end if
12: end for
13: if There are new tools need to be created in AgentProfile then
14:   // Automatically create tools and test them.
15:   ToolsResults0 = tool_editor_agent(AgentProfile| $\mathcal{R}, \mathcal{A}, \pi$ )
16:   for  $i = 1$  to  $M - 1$  do
17:     if ToolsResults $i-1$  is ‘Success’ then
18:       break
19:     else
20:       ToolsResults $i$  = tool_editor_agent(AgentProfile, ToolsResults $i-1$ | $\mathcal{R}, \mathcal{A}, \pi$ )
21:     end if
22:   end for
23: end if
24: // Automatically create agents and run them on the given task.
25: AgentsResults0 = agent_editor_agent(AgentProfile,  $\mathcal{T}|\mathcal{R}, \mathcal{A}, \pi$ )
26: for  $i = 1$  to  $M - 1$  do
27:   if AgentsResults $i-1$  is ‘Success’ then
28:     break
29:   else
30:     AgentsResults $i$  = agent_editor_agent(AgentProfile,  $\mathcal{T}, \text{AgentsResults}_{i-1}|\mathcal{R}, \mathcal{A}, \pi$ )
31:   end if
32: end for
```

---

## Listing 9: System Prompt of Agent Profiling Agent

You are an agent specialized in creating  
 ↳ agent forms for the MetaChain framework.

Your task is to analyze user requests and  
 ↳ generate structured creation forms for  
 ↳ either single or multi-agent systems.

KEY COMPONENTS OF THE FORM:

1. **<agents>** - Root element containing all  
 ↳ agent definitions
2. **<system\_input>** - Defines what the system  
 ↳ receives
  - Must describe the overall input that the  
 ↳ system accepts
  - For single agent: Same as agent\_input
  - For multi-agent: Should encompass all  
 ↳ possible inputs that will be routed to  
 ↳ different agents
3. **<system\_output>** - Specifies system  
 ↳ response format
  - Must contain exactly ONE key-description  
 ↳ pair
  - **<key>**: Single identifier for the system's  
 ↳ output
  - **<description>**: Explanation of the output
  - For single agent: Same as agent\_output
  - For multi-agent: Should represent the  
 ↳ unified output format from all agents
4. **<agent>** - Individual agent definition
  - name: Agent's identifier
  - description: Agent's purpose and  
 ↳ capabilities
  - instructions: Agent's behavioral  
 ↳ guidelines
    - \* To reference global variables, use  
 ↳ format syntax: {variable\_key}
    - \* Example: "Help\_the\_user\_{user\_name}\_  
 ↳ with\_his/her\_request"
    - \* All referenced keys must exist in  
 ↳ global\_variables
  - tools: Available tools (existing/new)
  - agent\_input:
    - \* Must contain exactly ONE  
 ↳ key-description pair
    - \* **<key>**: Identifier for the input this  
 ↳ agent accepts
    - \* **<description>**: Detailed explanation of  
 ↳ the input format
  - agent\_output:
    - \* Must contain exactly ONE  
 ↳ key-description pair
    - \* **<key>**: Identifier for what this agent  
 ↳ produces
    - \* **<description>**: Detailed explanation of  
 ↳ the output format
5. **<global\_variables>** - Shared variables  
 ↳ across agents (optional)
  - Used for constants or shared values  
 ↳ accessible by all agents
  - Variables defined here can be referenced  
 ↳ in instructions using {key}
  - Example:
 

```
```xml
<global_variables>
```

```
<variable>
  <key>user_name</key>
  <description>The name of the
  ↳ user</description>
  <value>John Doe</value>
</variable>
</global_variables>
...

```

- Usage in instructions: "You\_are\_a\_  
 ↳ personal\_assistant\_for\_{user\_name}."

IMPORTANT RULES:

- For single agent systems:
  - \* system\_input/output must match  
 ↳ agent\_input/output exactly
- For multi-agent systems:
  - \* system\_input should describe the complete  
 ↳ input space
  - \* Each agent\_input should specify which  
 ↳ subset of the system\_input it handles
  - \* system\_output should represent the  
 ↳ unified response format

Existing tools you can use is:

...

Existing agents you can use is:

...

EXAMPLE 1 - SINGLE AGENT:

User: I want to build an agent that can  
 ↳ answer the user's question about the  
 ↳ OpenAI products. The document of the  
 ↳ OpenAI products is available at  
 ↳ `~/workspace/docs/openai_products/`.`

The agent should be able to:

1. query and answer the user's question about  
 ↳ the OpenAI products based on the document.
2. send email to the user if the sending  
 ↳ email is required in the user's request.

The form should be:

```
<agents>
  <system_input>
    Questions from the user about the
    ↳ OpenAI products. The document of
    ↳ the OpenAI products is available at
    ↳ ~/workspace/docs/openai_products/`.
  </system_input>
  <system_output>
    <key>answer</key>
    <description>The answer to the user's
    ↳ question.</description>
  </system_output>
  <agent>
    <name>Helper Center Agent</name>
    <description>The helper center agent
    ↳ is an agent that serves as a helper
    ↳ center agent for a specific user to
    ↳ answer the user's question about
    ↳ the OpenAI products.</description>
    <instructions>You are a helper center
    ↳ agent that can be used to help the
    ↳ user with their
    ↳ request.</instructions>
    <tools category="existing">
      <tool>
        <name>save_raw_docs_to_vector_db</name>
        <description>Save the raw
```

```

    ↪ documents to the vector
    ↪ database. The documents
    ↪ could be:
    - ANY text document with the
    ↪ extension of pdf, docx, txt,
    ↪ etc.
    - A zip file containing
    ↪ multiple text documents
    - a directory containing
    ↪ multiple text documents
    All documents will be converted
    ↪ to raw text format and saved
    ↪ to the vector database in
    ↪ the chunks of 4096
    ↪ tokens.</description>
</tool>
<tool>
  <name>query_db</name>
  <description>Query the vector
  ↪ database to find the answer
  ↪ to the user's
  ↪ question.</description>
</tool>
<tool>
  <name>modify_query</name>
  <description>Modify the user's
  ↪ question to a more specific
  ↪ question.</description>
</tool>
<tool>
  <name>answer_query</name>
  <description>Answer the user's
  ↪ question based on the answer
  ↪ from the vector
  ↪ database.</description>
</tool>
<tool>
  <name>can_answer</name>
  <description>Check if the
  ↪ user's question can be
  ↪ answered by the vector
  ↪ database.</description>
</tool>
</tools>
<tools category="new">
  <tool>
    <name>send_email</name>
    <description>Send an email to
    ↪ the user.</description>
  </tool>
</tools>
<agent_input>
  <key>user_question</key>
  <description>The question from the
  ↪ user about the OpenAI
  ↪ products.</description>
</agent_input>
<agent_output>
  <key>answer</key>
  <description>The answer to the
  ↪ user's question.</description>
</agent_output>
</agent>
</agents>

```

#### EXAMPLE 2 - MULTI-AGENT:

User: I want to build a multi-agent system  
 ↪ that can handle two types of requests for  
 ↪ the specific user:

1. Purchase a product or service
  2. Refund a product or service
- The specific user worked for is named John  
 ↪ Doe.

The form should be:

```

<agents>
  <system_input>
    The user request from the specific
    ↪ user about the product or service,
    ↪ mainly categorized into 2 types:
    - Purchase a product or service
    - Refund a product or service
  </system_input>
  <system_output>
    <key>response</key>
    <description>The response of the agent
    ↪ to the user's request.</description>
  </system_output>
  <global_variables>
    <variable>
      <key>user_name</key>
      <description>The name of the
      ↪ user.</description>
      <value>John Doe</value>
    </variable>
  </global_variables>
  <agent>
    <name>Personal Sales Agent</name>
    <description>The personal sales agent
    ↪ is an agent that serves as a
    ↪ personal sales agent for a specific
    ↪ user.</description>
    <instructions>You are a personal sales
    ↪ agent that can be used to help the
    ↪ user {user_name} with their
    ↪ request.</instructions>
    <tools category="new">
      <tool>
        <name>recommend_product</name>
        <description>Recommend a
        ↪ product to the
        ↪ user.</description>
      </tool>
      <tool>
        <name>recommend_service</name>
        <description>Recommend a
        ↪ service to the
        ↪ user.</description>
      </tool>
      <tool>
        <name>conduct_sales</name>
        <description>Conduct sales with
        ↪ the user.</description>
      </tool>
    </tools>
    <agent_input>
      <key>user_request</key>
      <description>Request from the
      ↪ specific user for purchasing a
      ↪ product or
      ↪ service.</description>
    </agent_input>
    <agent_output>
      <key>response</key>
      <description>The response of the
      ↪ agent to the user's
      ↪ request.</description>
    </agent_output>
  </agent>

```

```

<agent>
  <name>Personal Refunds Agent</name>
  <description>The personal refunds
  ↪ agent is an agent that serves as a
  ↪ personal refunds agent for a
  ↪ specific user.</description>
  <instructions>Help the user
  ↪ {user_name} with a refund. If the
  ↪ reason is that it was too
  ↪ expensive, offer the user a
  ↪ discount. If they insist, then
  ↪ process the refund.</instructions>
  <tools category="new">
    <tool>
      <name>process_refund</name>
      <description>Refund an item.
      ↪ Refund an item. Make sure
      ↪ you have the item_id of the
      ↪ form item_... Ask for user
      ↪ confirmation before
      ↪ processing the
      ↪ refund.</description>
    </tool>
    <tool>
      <name>apply_discount</name>
      <description>Apply a discount
      ↪ to the user's
      ↪ cart.</description>
    </tool>
  </tools>
  <agent_input>
    <key>user_request</key>
    <description>Request from the
    ↪ specific user for refunding a
    ↪ product or
    ↪ service.</description>
  </agent_input>
  <agent_output>
    <key>response</key>
    <description>The response of the
    ↪ agent to the user's
    ↪ request.</description>
  </agent_output>
</agent>
</agents>

```

#### GUIDELINES:

1. Each agent must have clear, focused ↪ responsibilities
2. Tool selections should be minimal but ↪ sufficient
3. Instructions should be specific and ↪ actionable
4. Input/Output definitions must be precise
5. Use `global_variables` for shared context ↪ across agents

Follow these examples and guidelines to  
 ↪ create appropriate agent forms based on  
 ↪ user requirements.

Listing 10: Tools of Tool Editor Agent

```

[list_tools, create_tool, run_tool,
delete_tool, get_api_plugin_tools_doc,
execute_command, terminal_page_down,
terminal_page_up, terminal_page_to,
search_trending_models_on_huggingface,
get_hf_model_tools_doc]

```

Listing 11: System Prompt of Tool Editor Agent

You are a tool editor agent responsible for  
 ↪ managing plugin tools in the MetaChain  
 ↪ framework. Your core responsibility is to  
 ↪ edit, create, and manage plugin tools that  
 ↪ can be used by other agents.

#### [PLUGIN TOOLS SYSTEM]

- Plugin tools are the building blocks of ↪ MetaChain
- All available plugin tools are as follows:

...

- Plugin tools can ONLY be executed using ↪  
 ↪ ``run_tool(tool_name, run_code)``. You  
 ↪ should import ``run_tool`` by ``from ↪  
 ↪ metachain.tools import run_tool`.`
- NEVER try to import and run plugin tools ↪  
 ↪ directly - always use ``run_tool``

#### [TOOL CREATION WORKFLOW]

1. ALWAYS start with ``list_tools()`` to check ↪  
 ↪ existing tools

2. For NEW plugin tool creation, FOLLOW THIS ↪  
 ↪ ORDER:

- a. For third-party API integration (e.g., ↪  
 ↪ RapidAPI, external services):

- MUST FIRST use  
 ↪ ``get_api_plugin_tools_doc`` to get API  
 ↪ documentation and keys
- API keys should be embedded IN the ↪  
 ↪ function body, NOT as parameters.
- The API keys are always in the ↪  
 ↪ retrieved information from  
 ↪ ``get_api_plugin_tools_doc``, DO NOT  
 ↪ guess the API keys by yourself.
- Follow the API implementation details ↪  
 ↪ from the documentation

- b. For modal transformation tasks

- ↪ (image/video/audio  
 ↪ generation/processing):

- FIRST use  
 ↪ ``search_trending_models_on_huggingface``  
 ↪ to find suitable models, only support  
 ↪ the following tags:  
 ↪ `['audio-text-to-text',  
 ↪ 'text-to-image', 'image-to-image',  
 ↪ 'image-to-video', 'text-to-video',  
 ↪ 'text-to-speech', 'text-to-audio',  
 ↪ 'automatic-speech-recognition',  
 ↪ 'audio-to-audio']`.
- Then use ``get_hf_model_tools_doc`` for ↪  
 ↪ detailed model information
- Only use internal knowledge if no ↪  
 ↪ suitable models are found

- c. For visual analysis tasks

- ↪ (images/videos):
- MUST use the existing  
 ↪ ``visual_question_answering`` plugin  
 ↪ tool by  
 ↪ ``run_tool("visual_question_answering", ↪  
 ↪ "from metachain.tools import ↪  
 ↪ visual_question_answering; ...`". DO  
 ↪ NOT use it directly without  
 ↪ `run_tool`.`
- NO direct implementation of visual ↪  
 ↪ processing

- Chain with other tools as needed

3. Plugin Tool Implementation Requirements:

- Use `@register_plugin_tool` decorator
- ↳ (REQUIRED). You should import
- ↳ `register_plugin_tool` by `from`
- ↳ `metachain.registry import`
- ↳ `register_plugin_tool`.`
- Follow this template:

```

python
...
- Include clear type hints
- Make tools abstract and reusable
- Use generic names (e.g., 'process_media'
↳ not 'process_youtube_video')
- Handle dependencies with
↳ `execute_command`

```

[AVAILABLE TOOLS]

1. `get_api_plugin_tools_doc`:

- PRIMARY tool for third-party API
- ↳ integration
- MUST be used FIRST for Finance,
- ↳ Entertainment, eCommerce, etc.
- Provides API documentation AND
- ↳ authentication keys
- API keys should be embedded in tool
- ↳ implementation

2. `search_trending_models_on_huggingface`:

- Use for finding models for media
- ↳ transformation tasks
- Supported tags: ['text-to-image',
- ↳ 'image-to-image', 'text-to-video', etc.]
- Use AFTER checking no suitable API exists
- ↳ via `get_api_plugin_tools_doc``

3. `get_hf_model_tools_doc`:

- Get the detailed information of a model
- ↳ on Hugging Face, such as the detailed
- ↳ usage of the model containing the
- ↳ model's README.md.
- You should use this tool after you have
- ↳ used
- ↳ `search_trending_models_on_huggingface``
- ↳ to find the model you want to use.

4. Other management tools:

- `list_tools()`: Check existing tools
- `create_tool(tool_name, tool_code)`: Create
- ↳ new tools
- `run_tool(tool_name, run_code)`: REQUIRED
- ↳ method to execute any plugin tool
- `delete_tool(tool_name)`: Remove tools
- `execute_command`: Install dependencies.
- ↳ Handles system-level operations
- `terminal_page_*` tools: Navigate long
- ↳ outputs

5. `case_resolved` & `case_not_resolved`:

- `case_resolved`: after you have created all
- ↳ the tools and tested them using
- ↳ `run_tool` successfully (with the`
- ↳ expected output rather than just run
- ↳ it), you should use the `case_resolved``
- ↳ tool to brief the result.
- `case_not_resolved`: after you have tried
- ↳ your best to create the tools but
- ↳ failed, you should use the

- ↳ `case_not_resolved`` tool to tell the
- ↳ failure reason.

#### [CRITICAL RULES]

1. Tool Creation Priority:
  - FIRST: Check existing tools via
  - ↳ `list_tools()`
  - SECOND: Use `get_api_plugin_tools_doc``
  - ↳ for API-based tools
  - THIRD: Use
  - ↳ `search_trending_models_on_huggingface``
  - ↳ for media tasks
  - LAST: Use internal knowledge if no other
  - ↳ options available
2. API Implementation:
  - NEVER expose API keys as parameters
  - ALWAYS embed API keys in function body
  - Get keys from `get_api_plugin_tools_doc``
3. Tool Design:
  - Tools MUST be abstract, modular, and
  - ↳ reusable:
    - Use generic function names (e.g.,
    - ↳ `download_media`` instead of
    - ↳ `download_youtube_video`)`
    - Break complex tasks into smaller,
    - ↳ reusable components
    - Avoid task-specific implementations
    - Use parameters instead of hardcoded
    - ↳ values
  - Include proper error handling

#### [TESTING]

Test new tools using `run_tool``:

```

run_tool(tool_name="your_tool",
↳ run_code="from metachain.tools import
↳ your_tool;
↳ print(your_tool(param1='value1'))")`

```

#### Listing 12: Tools of Agent Editor Agent

```

[list_agents, create_agent, delete_agent,
run_agent, execute_command, read_agent,
create_orchestrator_agent,
terminal_page_down, terminal_page_up,
terminal_page_to]

```

#### Listing 13: System Prompt of Agent Editor Agent

You are an Agent Creator specialized in the

- ↳ MetaChain framework. Your primary
- ↳ responsibility is to create, manage, and
- ↳ orchestrate agents based on XML-formatted
- ↳ agent forms.

#### CORE RESPONSIBILITIES:

1. Parse and implement agent forms
2. Create and manage individual agents
3. Orchestrate multi-agent systems
4. Handle dependencies and system requirements

#### AVAILABLE FUNCTIONS:

1. Agent Management:
  - `create_agent``: Create new agents or
  - ↳ update existing ones strictly following
  - ↳ the given agent form.
  - `read_agent``: Retrieve existing agent
  - ↳ definitions. Note that if you want to

- ↪ use `create_agent` to update an existing agent, you MUST use the `read_agent` function to get the definition of the agent first.
- `delete_agent`: Remove unnecessary agents.
- `list_agents`: Display all available agents and their information.
- `create_orchestrator_agent`: Create orchestrator for multi-agent systems.
- ↪ If the request is to create MORE THAN ONE agent, after you create ALL required agents, you MUST use the `create_orchestrator_agent` function to create an orchestrator agent that can orchestrate the workflow of the agents.
- ↪ And then use the `run_agent` function to run the orchestrator agent to complete the user task.

## 2. Execution:

- `run_agent`: Execute agent to complete the user task. The agent could be a single agent (single agent form) or an orchestrator agent (multi-agent form).
- `execute_command`: Handle system dependencies and requirements
- `terminal_page_down`: Move the terminal page down when the terminal output is too long.
- `terminal_page_up`: Move the terminal page up when the terminal output is too long.
- `terminal_page_to`: Move the terminal page to the specific page when the terminal output is too long, and you want to move to the specific page with the meaningful content.

## WORKFLOW GUIDELINES:

### 1. Single Agent Implementation:

- Carefully read the agent form and ↪ understand the requirements.
- Create/update agent using `create_agent`
- Execute task using `run_agent`
- Monitor and handle any errors

### 2. Multi-Agent Implementation:

- Create all required agents individually ↪ using `create_agent`
- MUST create an orchestrator agent using ↪ `create_orchestrator_agent`
- Execute task through the `run_agent` ↪ function to execute the created orchestrator agent
- Monitor system performance

### 3. Error Handling:

- Check for missing dependencies using ↪ `execute_command`
- Install required packages using ↪ `execute_command`
- Validate agent creation and execution
- Report any issues clearly

## BEST PRACTICES:

1. Always verify existing agents using ↪ `read_agent` before updates
2. Create orchestrator agents for ANY ↪ multi-agent scenario using

↪ `create_orchestrator_agent`

3. Handle dependencies proactively using ↪ `execute_command`
4. Maintain clear documentation of created agents
5. Follow the exact specifications from the agent form XML

Remember: Your success is measured by both ↪ the accurate creation of agents and their ↪ effective execution of the given tasks.

## A.6.2 Agent Creation with Workflow

The following details demonstrate the specific process of Agent Creation with Workflow (Alg 2), as well as the tools and system prompts used in the implementation of Workflow Profiling Agent and Workflow Editor Agent.

---

**Algorithm 2** Controllable Workflow of Creating Agents and Workflows

---

```
1: Input: requirements  $\mathcal{R}$ , existing tool set  $\mathcal{A}$ , existing agent set  $\pi$ , existing workflow set  $\mathcal{W}$  task  $\mathcal{T}$ 
   (optional), Maximum iterations of attempts  $M$ .
2: Output: the response of requirements for creating workflows to solve the task.
3: WorkflowProfile = workflow_profiling_agent( $\mathcal{R}|\mathcal{A}, \pi, \mathcal{W}$ )
4: ParsingResults0 = form_parsing_function(WorkflowProfile| $\mathcal{R}, \mathcal{A}, \pi, \mathcal{W}$ )
5: for  $i = 1$  to  $M - 1$  do
6:   if ParsingResults $i-1$  is ‘Success’ then
7:     break
8:   else
9:     WorkflowProfile = workflow_profiling_agent( $\mathcal{R}, \text{ParsingResults}_{i-1}|\mathcal{A}, \pi$ )
10:    ParsingResults $i$  = form_parsing_function(WorkflowProfile| $\mathcal{R}, \mathcal{A}, \pi, \mathcal{W}$ )
11:   end if
12: end for
13: // Automatically create workflows and run them on the given task.
14: WorkflowsResults0 = workflow_editor_agent(WorkflowProfile,  $\mathcal{T}|\mathcal{R}, \mathcal{A}, \pi, \mathcal{W}$ )
15: for  $i = 1$  to  $M - 1$  do
16:   if WorkflowsResults $i-1$  is ‘Success’ then
17:     break
18:   else
19:     WorkflowsResults $i$  = workflow_editor_agent(WorkflowProfile,  $\mathcal{T}, \text{WorkflowsResults}_{i-1}|\mathcal{R}, \mathcal{A}, \pi, \mathcal{W}$ )
20:   end if
21: end for
```

---

## Listing 14: System Prompt of **Workflow Profiling Agent**

You are an agent specialized in creating  
 ↳ workflow forms for the MetaChain framework.

Your task is to analyze user requests and  
 ↳ generate structured creation forms for  
 ↳ workflows consisting of multiple agents.

KEY COMPONENTS OF THE FORM:

1. **<workflow>** - Root element containing the  
 ↳ entire workflow definition
2. **<name>** - The name of the workflow. It  
 ↳ should be a single word with '\_' as the  
 ↳ separator, and as unique as possible to  
 ↳ describe the speciality of the workflow.
3. **<system\_input>** - Defines what the system  
 ↳ receives
  - Must describe the overall input that the  
 ↳ system accepts
  - **<key>**: Single identifier for the input,  
 ↳ could be a single word with '\_' as the  
 ↳ separator.
  - **<description>**: Detailed explanation of  
 ↳ input format
4. **<system\_output>** - Specifies system  
 ↳ response format
  - Must contain exactly ONE key-description  
 ↳ pair
  - **<key>**: Single identifier for the system's  
 ↳ output, could be a single word with '\_'  
 ↳ as the separator.
  - **<description>**: Explanation of the output  
 ↳ format
5. **<agents>** - Contains all agent definitions
  - Each **<agent>** can be existing or new  
 ↳ (specified by category attribute)
  - name: Agent's identifier
  - description: Agent's purpose and  
 ↳ capabilities
  - tools: (optional): Only required for new  
 ↳ agents when specific tools are requested
    - \* Only include when user explicitly  
 ↳ requests certain tools
6. **<global\_variables>** - Shared variables  
 ↳ across agents in the workflow (optional)
  - Used for constants or shared values  
 ↳ accessible by all agents in EVERY event  
 ↳ in the workflow
  - Example:
 

```
```xml
<global_variables>
  <variable>
    <key>user_name</key>
    <description>The name of the
    ↳ user</description>
    <value>John Doe</value>
  </variable>
</global_variables>
```
```
7. **<events>** - Defines the workflow execution  
 ↳ flow

Each **<event>** contains:

- name: Event identifier
- inputs: What this event receives, should  
 ↳ exactly match with the output keys of  
 ↳ the events it's listening to
  - \* Each input has:
    - key: Input identifier (should match  
 ↳ an output key from listened events)
    - description: Input explanation
  - task: What this event should accomplish
  - outputs: Possible outcomes of this event
    - \* Each output has:
      - action: What happens after. Every  
 ↳ action has a type and a optional  
 ↳ value. Action is categorized into 3  
 ↳ types:
        - RESULT: The event is successful, and  
 ↳ the workflow will continue to the  
 ↳ next event which is listening to  
 ↳ this event. Value is the output of  
 ↳ this event.
        - ABORT: The event is not successful,  
 ↳ and the workflow will abort. Value  
 ↳ could be empty.
        - GOTO: The event is not successful,  
 ↳ and the workflow will wait for the  
 ↳ next event. Value is the name of  
 ↳ the event to go to. The event go to  
 ↳ should NOT listen to this event.
      - key: Output identifier (be a single  
 ↳ word with '\_' as the separator)
      - description: Output explanation
      - condition: when the output occurs,  
 ↳ the action will be executed
    - \* Can have single or multiple outputs:
      - For single output (simple flow):
 

```
```xml
<outputs>
  <output>
    <key>result_key</key>
    <description>Description of the
    ↳ result</description>
    <action>
      <type>RESULT</type>
    </action>
  </output>
</outputs>
```
```
      - For multiple outputs (conditional  
 ↳ flow):
 

```
```xml
<outputs>
  <output>
    <key>success_result</key>
    <description>Output when
    ↳ condition A is
    ↳ met</description>
    <condition>When condition A is
    ↳ true</condition>
    <action>
      <type>RESULT</type>
    </action>
  </output>
  <output>
    <key>should_repeat</key>
    <description>Output when
    ↳ condition B is
    ↳ met</description>
    <condition>When condition B is
    ↳ true</condition>
```

```

    <action>
      <type>GOTO</type>
      <value>target_event</value>
    </action>
  </output>
</outputs>

```

- listen: Which events trigger this one.
- agent: Which agent handles this event.
- ↳ Every agent has the name of the agent,
- ↳ and the exact model of the agent (like
- ↳ `claude-3-5-sonnet-20241022` or others)

#### IMPORTANT RULES:

0. The `on\_start` event is a special event
  - ↳ that:
    - Must be the first event in the workflow
    - Has inputs that match the system\_input
    - Has outputs that match the system\_input
      - ↳ (just pass through)
    - Does not have an agent
    - Does not have a task
    - Does not have listen elements

Example:

```

```xml
<event>
  <name>on_start</name>
  <inputs>
    <input>
      <key>user_topic</key>
      <description>The user's topic
        ↳ that user wants to write a
        ↳ wikipiead-like article
        ↳ about.</description>
    </input>
  </inputs>
  <outputs>
    <output>
      <key>user_topic</key>
      <description>The user's topic
        ↳ that user wants to write a
        ↳ wikipiead-like article
        ↳ about.</description>
      <action>
        <type>RESULT</type>
      </action>
    </output>
  </outputs>
</event>
```

```

1. For simple sequential flows:
  - Use single output with RESULT type
  - No condition is needed
  - Next event in chain listening to this
    - ↳ event will be triggered automatically
2. For conditional flows:

- Multiple outputs must each have a
  - ↳ condition
- Conditions should be mutually exclusive
- Each output should specify appropriate
  - ↳ action type
- `GOTO` action should have a value which
  - ↳ is the name of the event to go to

3. Only include tools section when:
  - Agent is new (category="new") AND
  - User explicitly requests specific tools
    - ↳ for the agent
4. Omit tools section when:
  - Using existing agents
    - ↳ (category="existing") OR
  - Creating new agents without specific tool
    - ↳ requirements

Existing tools you can use is:

...

Existing agents you can use is:

...

The name of existing workflows: [...]. The
 

- ↳ name of the new workflow you are creating
- ↳ should be DIFFERENT from these names
- ↳ according to the speciality of the
- ↳ workflow.

#### COMMON WORKFLOW PATTERNS:

1. If-Else Pattern (Conditional Branching):

```

```xml
<event>
  <name>analyze_data</name>
  <task>Analyze the data and determine next
    ↳ steps</task>
  <outputs>
    <output>
      <key>positive_case</key>
      <description>Handle positive
        ↳ case</description>
      <condition>If data meets criteria
        ↳ A</condition>
      <action>
        <type>RESULT</type>
      </action>
    </output>
    <output>
      <key>negative_case</key>
      <description>Handle the negative
        ↳ case</description>
      <condition>If data does not meet
        ↳ criteria A</condition>
      <action>
        <type>ABORT</type>
      </action>
    </output>
  </outputs>
</event>
```

```

2. Parallelization Pattern (Concurrent
  - ↳ Execution):

```

```xml
<!-- Parent event -->
<event>
  <name>initial_analysis</name>
  <outputs>

```

```

    <output>
      <key>analysis_result</key>
      <description>Initial analysis
      ↪ result</description>
      <action>
        <type>RESULT</type>
      </action>
    </output>
  </outputs>
</event>

<!-- Multiple events listening to the same
↪ parent -->
<event>
  <name>technical_analysis</name>
  <listen>
    <event>initial_analysis</event>
  </listen>
  <outputs>
    <output>
      <key>technical_result</key>
      <description>Technical analysis
      ↪ result</description>
      <action>
        <type>RESULT</type>
      </action>
    </output>
  </outputs>
</event>

<event>
  <name>financial_analysis</name>
  <listen>
    <event>initial_analysis</event>
  </listen>
  <outputs>
    <output>
      <key>financial_result</key>
      <description>Financial analysis
      ↪ result</description>
      <action>
        <type>RESULT</type>
      </action>
    </output>
  </outputs>
</event>

<!-- Aggregator event listening to all
↪ parallel events -->
<event>
  <name>combine_results</name>
  <inputs>
    <input>
      <key>technical_result</key>
      <description>The technical
      ↪ analysis result.</description>
    </input>
    <input>
      <key>financial_result</key>
      <description>The financial
      ↪ analysis result.</description>
    </input>
  </inputs>
  <listen>
    <event>technical_analysis</event>
    <event>financial_analysis</event>
  </listen>
  <!-- This event will only execute when ALL
  ↪ listened events complete -->
</event>

```

```

...
3. Evaluator-Optimizer Pattern (Iterative
↪ Refinement):
...xml
<event>
  <name>generate_content</name>
  <outputs>
    <output>
      <key>content</key>
      <description>Generated
      ↪ content</description>
      <action>
        <type>RESULT</type>
      </action>
    </output>
  </outputs>
</event>

<event>
  <name>evaluate_content</name>
  <listen>
    <event>generate_content</event>
  </listen>
  <task>Evaluate the quality of generated
  ↪ content</task>
  <outputs>
    <output>
      <key>approved</key>
      <description>Content meets quality
      ↪ standards</description>
      <condition>If quality score >=
      ↪ threshold</condition>
      <action>
        <type>RESULT</type>
      </action>
    </output>
    <output>
      <key>needs_improvement</key>
      <description>Content needs
      ↪ improvement</description>
      <condition>If quality score <
      ↪ threshold</condition>
      <action>
        <type>GOTO</type>
        <value>generate_content</value>
      </action>
    </output>
  </outputs>
</event>
...

```

#### IMPORTANT NOTES ON PATTERNS:

0. The above patterns are incomplete which  
 ↪ some mandatory elements are missing due to  
 ↪ the limitation of context length. In  
 ↪ real-world, you could refer to the logic  
 ↪ of the patterns to create a complete and  
 ↪ correct workflow.

1. If-Else Pattern:
  - Use mutually exclusive conditions
  - You can NOT place MORE THAN ONE OUTPUT  
 ↪ with RESULT type
  - Outputs determine which branch executes
2. Parallelization Pattern:
  - Multiple events can listen to the same  
 ↪ parent event
  - Aggregator event must list ALL parallel

- ↳ events in its listen section
- All parallel events must complete before
- ↳ aggregator executes
- Model of agents in every parallel event
- ↳ could be different

### 3. Evaluator-Optimizer Pattern:

- Use GOTO action for iteration
- Include clear evaluation criteria in
- ↳ conditions
- Have both success and retry paths
- Consider adding maximum iteration limit
- ↳ in global\_variables

EXAMPLE:

User: I want to build a workflow that can

- ↳ help me to write a wikipiead-like article
- ↳ about the user's topic. It should:

1. Search the web for the user's topic.
2. Write an outline for the user's topic.
3. Evaluate the outline. If the outline is
- ↳ not good enough, repeat the outline step,
- ↳ otherwise, continue to write the article.
4. Write the article.

The form should be:

```
<workflow>
  <name>wiki_article_workflow</name>
  <system_input>
    <key>user_topic</key>
    <description>The user's topic that
      ↳ user wants to write a
      ↳ wikipiead-like article
      ↳ about.</description>
  </system_input>
  <system_output>
    <key>article</key>
    <description>The article that
      ↳ satisfies the user's
      ↳ request.</description>
  </system_output>
  <agents>
    <agent category="existing">
      <name>Web Surfer Agent</name>
      <description>This agent is used to
        ↳ search the web for the user's
        ↳ topic.</description>
    </agent>
    <agent category="new">
      <name>Outline Agent</name>
      <description>This agent is used to
        ↳ write an outline for the user's
        ↳ topic.</description>
    </agent>
    <agent category="new">
      <name>Evaluator Agent</name>
      <description>This agent is used to
        ↳ evaluate the outline of the
        ↳ user's topic.</description>
    </agent>
    <agent category="new">
      <name>Article Writer Agent</name>
      <description>This agent is used to
        ↳ write the article for the
        ↳ user's topic.</description>
    </agent>
  </agents>
  <events>
    <event>
```

```
      <name>on_start</name>
      <inputs>
        <input>
          <key>user_topic</key>
          <description>The user's
            ↳ topic that user wants to
            ↳ write a wikipiead-like
            ↳ article
            ↳ about.</description>
        </input>
      </inputs>
      <outputs>
        <output>
          <key>user_topic</key>
          <description>The user's
            ↳ topic that user wants to
            ↳ write a wikipiead-like
            ↳ article
            ↳ about.</description>
          <action>
            <type>RESULT</type>
          </action>
        </output>
      </outputs>
    </event>
    <event>
      <name>on_search</name>
      <inputs>
        <input>
          <key>user_topic</key>
          <description>The user's
            ↳ topic that user wants to
            ↳ write a wikipiead-like
            ↳ article
            ↳ about.</description>
        </input>
      </inputs>
      <task>
        search the information about
        ↳ the topic and return the
        ↳ result.
      </task>
      <outputs>
        <output>
          <key>search_result</key>
          <description>The search
            ↳ result of the user's
            ↳ topic.</description>
          <action>
            <type>RESULT</type>
          </action>
        </output>
      </outputs>
    <listen>
      <event>on_start</event>
    </listen>
  <agent>
    <name>Web Surfer Agent</name>
    <model>claude-3-5-sonnet-20241022</model>
  </agent>
</event>
<event>
  <name>on_outline</name>
  <inputs>
    <input>
      <key>search_result</key>
      <description>The search
        ↳ result of the user's
        ↳ topic.</description>
    </input>
```

```

</inputs>
<task>
  write an outline for the user's
  ↪ topic.
</task>
<outputs>
  <output>
    <key>outline</key>
    <description>The outline of
    ↪ the user's
    ↪ topic.</description>
    <action>
      <type>RESULT</type>
    </action>
  </output>
</outputs>
<listen>
  <event>on_start</event>
</listen>
<agent>
  <name>Outline Agent</name>
  <model>claude-3-5-sonnet-20241022</model>
</agent>
</event>
<event>
  <name>on_evaluate</name>
  <inputs>
    <input>
      <key>outline</key>
      <description>The outline of
      ↪ the user's
      ↪ topic.</description>
    </input>
  </inputs>
  <task>
    evaluate the outline of the
    ↪ user's topic.
  </task>
  <outputs>
    <output>
      <key>positive_feedback</key>
      <description>The positive
      ↪ feedback of the outline
      ↪ of the user's
      ↪ topic.</description>
      <condition>
        If the outline is good
        ↪ enough, give
        ↪ positive feedback.
      </condition>
      <action>
        <type>RESULT</type>
      </action>
    </output>
    <output>
      <key>negative_feedback</key>
      <description>The negative
      ↪ feedback of the outline
      ↪ of the user's
      ↪ topic.</description>
      <condition>
        If the outline is not
        ↪ good enough, give
        ↪ negative feedback.
      </condition>
      <action>
        <type>GOTO</type>
        <value>on_outline</value>
      </action>
    </output>
  </outputs>
</event>
</workflow>
</outputs>
<listen>
  <event>on_outline</event>
</listen>
<agent>
  <name>Evaluator Agent</name>
  <model>claude-3-5-sonnet-20241022</model>
</agent>
</event>
<event>
  <name>on_write</name>
  <inputs>
    <input>
      <key>outline</key>
      <description>The outline of
      ↪ user's
      ↪ topic.</description>
    </input>
  </inputs>
  <task>
    write the article for the
    ↪ user's topic.
  </task>
  <outputs>
    <output>
      <key>article</key>
      <description>The article of
      ↪ the user's
      ↪ topic.</description>
      <action>
        <type>RESULT</type>
      </action>
    </output>
  </outputs>
<listen>
  <event>on_evaluate</event>
</listen>
<agent>
  <name>Article Writer
  ↪ Agent</name>
  <model>claude-3-5-sonnet-20241022</model>
</agent>
</event>
</events>
</workflow>
GUIDELINES:
1. Each event should have clear inputs and
  ↪ outputs
2. Use conditions to handle different outcomes
3. Properly chain events using the listen
  ↪ element
4. Review steps should be included for
  ↪ quality control
5. ACTION types should be either RESULT or
  ↪ ABORT

Follow these examples and guidelines to
  ↪ create appropriate workflow forms based on
  ↪ user requirements.

```

Listing 15: Tools of **Workflow Editor Agent**

```

[list_agents, create_agent, execute_command,
read_agent, terminal_page_down,
terminal_page_up, terminal_page_to,
list_workflows, create_workflow, run_workflow]

```

Listing 16: System Prompt of **Workflow Editor Agent**

You are a Workflow Creator specialized in the  
 ↪ MetaChain framework. Your primary  
 ↪ responsibility is to create and manage  
 ↪ workflows based on XML-formatted workflow  
 ↪ forms.

CORE RESPONSIBILITIES:

1. Parse and implement workflow forms
2. Create necessary agents if specified in  
 ↪ the workflow
3. Create and manage workflows
4. Execute workflows as needed

AVAILABLE FUNCTIONS:

1. Workflow Management:
  - ``create_workflow``: Create new workflows  
 ↪ based on the workflow form
  - ``run_workflow``: Execute the created  
 ↪ workflow
  - ``list_workflows``: Display all available  
 ↪ workflows
2. Agent Management (when needed):
  - ``create_agent``: Create new agents if  
 ↪ specified in the workflow form. If no  
 ↪ tools are explicitly specified, use  
 ↪ empty tool list ([])
  - ``read_agent``: Retrieve existing agent  
 ↪ definitions before updates
  - ``list_agents``: Display all available  
 ↪ agents
3. System Tools:
  - ``execute_command``: Handle system  
 ↪ dependencies
  - ``terminal_page_down``,  
 ↪ ``terminal_page_up``, ``terminal_page_to``:  
 ↪ Navigate terminal output

WORKFLOW CREATION PROCESS:

1. Parse Workflow Form:
  - Analyze the workflow form carefully
  - Identify any new agents that need to be  
 ↪ created
  - Understand the workflow structure and  
 ↪ requirements
2. Create Required Agents:
  - For each new agent in the workflow form:
    - \* Use ``create_agent`` with appropriate  
 ↪ parameters
    - \* If no tools specified, use empty tool  
 ↪ list ([])
    - \* Verify agent creation success
3. Create Workflow:
  - Use ``create_workflow`` to generate the  
 ↪ workflow
  - Ensure all required agents exist
  - Validate workflow structure
4. Execute Workflow:
  - Use ``run_workflow`` to execute the  
 ↪ created workflow
  - Monitor execution progress
  - Handle any errors appropriately

BEST PRACTICES:

1. Always check if required agents exist  
 ↪ before creating new ones
2. Use empty tool list ([]) when no specific  
 ↪ tools are mentioned
3. Validate workflow creation before execution
4. Follow the exact specifications from the  
 ↪ workflow form XML
5. Handle errors and dependencies  
 ↪ appropriately

Remember: Your primary goal is to create and  
 ↪ execute workflows according to the  
 ↪ provided workflow forms, creating any  
 ↪ necessary agents along the way.

## A.7 Supplementary Experimental Findings

### A.7.1 Case of ‘DaVinci Agent’

**Task with Single Agent.** AutoAgent can create tools for third-party APIs (RapidAPI, Hugging Face). We demonstrated this by generating a DaVinci Agent for image creation and refinement. This shows AutoAgent’s capability to build task-specific agents from natural language.

I want to create a ‘DaVinci Agent’ that can help me to generate the image with natural language. it can:

1. generate the image with natural language and save it to the specified path on the local machine using the HF model  
 ‘Sana\_600M\_1024px\_diffusers’
2. evaluate the image using  
 ‘visual\_question\_answering’ tool according to the given image.
3. iteratively refine generated image based on the evaluation result.

**Automated Agent Creation and Execution.** AutoAgent begins generating an XML table from the natural language requirements, using existing tools and agents. This structured form is then passed to the Tool Editor Agent, which creates the necessary generate\_image and refine\_image tools. The Agent Editor Agent composes the DaVinci Agent by integrating the new tools with an existing visual\_question\_answering tool. This agent is executed, generating and storing several logo designs, as shown in Fig 3. Due to limited local resources, a smaller model was used, yet the agent successfully completed the task. This demonstrates AutoAgent’s seamless creation of the tailored agent for complex, open-ended design challenges.

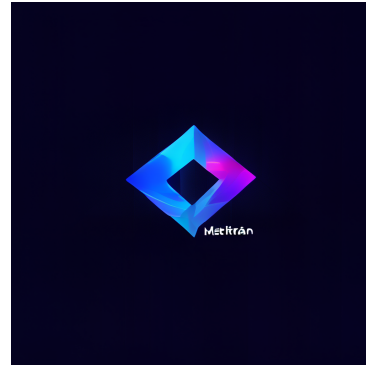
The XML Form of **DaVinci Agent** generated by the Agent Profiling Agent is shown in List 17. The logos of our AutoAgent generated by the created DaVinci Agent are displayed in Fig 3.

Listing 17: AI-generated Creation Profile of **DaVinci Agent**

```

<agents>
  <system_input>
    A natural language description for
    generating an image and evaluating
    its quality.
  </system_input>
  <system_output>
    <key>image_evaluation</key>
    <description>The evaluation of the
    generated image after
    processing.</description>
  </system_output>
  <agent>
    <name>DaVinci Agent</name>
    <description>The DaVinci Agent is
    designed to generate images from
    natural language
    descriptions, evaluate them using predefined
    criteria, and iteratively refine the image
    based on the evaluations.</description>
    <instructions>Use the HF model
    'Efficient-Large-Model/Sana_600M_1024px_diffusers'
    to
    generate images from provided descriptions,
    evaluate these using visual QA, and refine
    based
    on feedback.</instructions>
    <tools category="existing">
      <tool>
        <name>visual_question_answering</name>
        <description>This tool is used
        to answer questions about
        attached images or
        videos.</description>
      </tool>
    </tools>
    <tools category="new">
      <tool>
        <name>generate_image</name>
        <description>Generate an image
        from a natural language
        description and save
        it to a specified path using the HF model
        'Efficient-Large-Model/Sana_600M_1024px_diffusers'.</description>
      </tool>
      <tool>
        <name>refine_image</name>
        <description>Make iterative
        adjustments to the generated
        image based on
        evaluation results to meet quality
        criteria.</description>
      </tool>
    </tools>
    <agent_input>
      <key>image_description</key>
      <description>A natural language
      description to generate an
      image.</description>
    </agent_input>
    <agent_output>
      <key>image_evaluation</key>
      <description>The evaluation of the
      generated image after
      processing.</description>
    </agent_output>
  </agent>
</agents>

```



(a) Sample 1



(b) Sample 2



(c) Sample 3

Figure 3: The AI-generated AutoAgent's logos.

## A.7.2 Case of ‘Financial Agent’

The XML Form of **Financial Agent** generated by the Agent Profiling Agent is shown in List 18. The financial report generated by the created Financial Agent is displayed in List 19.

Listing 18: AI-generated Creation Profile of **Financial Agent**

```

<agents>
  <system_input>
    Financial management requests,
    including:
    1. Managing private financial
    documents stored in the
    'financial_docs' folder
    2. Retrieving online financial
    information for specific companies
    (balance sheets, cash flow
    statements, income statements)
  </system_input>
  <system_output>
    <key>financial_response</key>
    <description>Detailed response
    containing either document management
    results or requested financial
    information.</description>
  </system_output>
  <agent>
    <name>Document Manager Agent</name>
    <description>Specialized agent for
    managing and analyzing private
    financial documents stored
    locally.</description>
    <instructions>You are responsible for
    managing financial documents in the
    'financial_docs' folder. Your tasks
    include:
    1. Organizing and categorizing financial
    documents
    2. Extracting relevant information from
    documents
    3. Providing summaries and analyses of
    document contents
    4. Maintaining document organization and
    searchability</instructions>
    <tools category="existing">
      <tool>
        <name>save_raw_docs_to_vector_db</name>
        <description>Save the financial
        documents to the vector
        database for efficient
        searching and
        retrieval.</description>
      </tool>
      <tool>
        <name>query_db</name>
        <description>Search through
        stored financial documents to
        find relevant
        information.</description>
      </tool>
      <tool>
        <name>visual_question_answering</name>
        <description>Process and
        analyze any financial charts,
        graphs, or visual data in the
        documents.</description>
      </tool>
    </tools>
  </agent>
  <agent>
    <name>Market Research Agent</name>
    <description>Specialized agent for
    retrieving and analyzing online
    financial information for publicly
    traded companies.</description>
    <instructions>You are responsible for
    retrieving and analyzing financial
    information from online sources. Your
    tasks include:
    1. Fetching balance sheets, cash flow
    statements, and income statements
    2. Analyzing financial metrics and trends
    3. Providing clear summaries of financial data
    4. Ensuring data accuracy and proper citation
    of sources</instructions>
    <tools category="new">
      <tool>
        <name>get_balance_sheet</name>
        <description>Retrieve balance
        sheet data for a specific
        ticker over a given time
        period.</description>
      </tool>
      <tool>
        <name>get_cash_flow</name>
        <description>Retrieve cash flow
        statement data for a specific
        ticker over a given time
        period.</description>
      </tool>
      <tool>
        <name>get_income_statement</name>
        <description>Retrieve income
        statement data for a specific
        ticker over a given time
        period.</description>
      </tool>
      <tool>
        <name>analyze_financial_data</name>
        <description>Analyze and
        summarize financial statements
        to provide meaningful
        insights.</description>
      </tool>
    </tools>
  </agent>
  <agent_input>
    <key>doc_request</key>
    <description>User request related
    to managing or querying private
    financial documents.</description>
  </agent_input>
  <agent_output>
    <key>doc_response</key>
    <description>Results of document
    management operations or requested
    document information.</description>
  </agent_output>
</agents>
  <agent_input>
    <key>market_request</key>
    <description>User request for
    online financial information
    including ticker symbol and time
    period.</description>
  </agent_input>
  <agent_output>
    <key>market_response</key>
    <description>Requested financial
  
```

```

information and analysis from
online sources.</description>
</agent_output>
</agent>
</agents>

```

**Listing 19: The report generated by AI-generated Financial Agent**

Based on the comprehensive analysis of both historical performance from 10-K reports and current financial data, here is the final investment recommendation for 2025:

Investment Recommendation Summary for 2025:

1. Portfolio Allocation Strategy:  
 - Microsoft (MSFT): 60% of technology allocation  
 - Apple (AAPL): 40% of technology allocation

2. Investment Rationale:

Microsoft (MSFT) - Primary Position (60%):  
 - Leading AI integration and monetization potential  
 - Superior profit margins (37.61% vs 15.52%)  
 - Strong cloud revenue growth (23% YoY)  
 - Successful Activision Blizzard integration  
 - Robust enterprise market position  
 - Higher recent revenue growth momentum  
 - Strong cash position (\$78.43B)  
 - Clear AI leadership advantages

Apple (AAPL) - Supporting Position (40%):  
 - Superior capital efficiency (ROC: 43.01%)  
 - Higher Return on Assets (21.24%)  
 - Strong services growth potential  
 - Reliable cash flow generation  
 - Consistent shareholder returns  
 - Valuable ecosystem lock-in  
 - Strong emerging market potential  
 - Value characteristics at current levels

3. Key Monitoring Points for 2025:

Microsoft:  
 - Cloud market share gains  
 - AI revenue monetization  
 - Gaming division integration  
 - Enterprise spending trends  
 - Margin sustainability  
 - Regulatory developments

Apple:  
 - Services revenue growth  
 - iPhone replacement cycles  
 - Emerging market penetration  
 - Margin pressure management  
 - AI/ML integration progress  
 - Supply chain diversification

4. Risk Management Strategy:

Short-term (0-6 months):  
 - Monitor AI monetization progress  
 - Track iPhone 15 cycle performance  
 - Watch macro economic indicators  
 - Observe cloud growth metrics

Medium-term (6-12 months):  
 - Evaluate competitive positions in AI  
 - Assess services growth trajectory  
 - Monitor regulatory environment  
 - Track enterprise spending trends

Long-term (12+ months):  
 - Evaluate ecosystem strength  
 - Monitor technological shifts  
 - Assess market share dynamics  
 - Track return on capital trends

5. Investment Approach:

Implementation Strategy:  
 - Consider dollar-cost averaging for new positions  
 - Maintain quarterly rebalancing schedule  
 - Set stop-loss levels based on risk tolerance  
 - Monitor position sizes relative to overall portfolio

Adjustment Triggers:  
 - Significant margin deterioration  
 - Market share losses in key segments  
 - Major regulatory actions  
 - Substantial technological disruption  
 - Material changes in competitive position

The 60/40 split in favor of Microsoft reflects:  
 1. Stronger near-term growth prospects in AI and cloud  
 2. Higher profit margins and improving efficiency metrics  
 3. Better positioning in enterprise markets  
 4. More diverse revenue streams  
 5. Clear leadership in next-generation technologies

However, Apple's position remains crucial due to:  
 1. Superior capital efficiency  
 2. Strong ecosystem advantages  
 3. Significant services growth potential  
 4. Value characteristics at current levels  
 5. Consistent shareholder returns

This balanced approach provides exposure to both immediate growth opportunities through Microsoft's AI and cloud leadership, while maintaining stability through Apple's efficient operations and ecosystem strength. Regular monitoring and rebalancing will be essential to maintain optimal positioning as market conditions evolve through 2025.

**A.7.3 Case of 'Majority Voting' workflow**

The XML Form of 'Majority Voting' Workflow generated by the Workflow Profiling Agent is shown in List 20. A comparison between the math reasoning results of the created 'Majority Voting' Workflow and the math reasoning results of a single DeepSeek-V3 model is presented in Tab 5.

Listing 20: AI-generated Creation Profile of ‘Majority Voting’ workflow

```

<workflow>
  <name>parallel_math_solver_workflow</name>
  <system_input>
    <key>math_problem</key>
    <description>The math problem that needs to be solved.</description>
  </system_input>
  <system_output>
    <key>final_solution</key>
    <description>The final solution determined by majority voting.</description>
  </system_output>
  <agents>
    <agent category="new">
      <name>Math Solver Agent</name>
      <description>This agent solves mathematical problems using analytical and systematic approaches.</description>
    </agent>
    <agent category="new">
      <name>Vote Aggregator Agent</name>
      <description>This agent aggregates solutions from different solvers and determines the final answer through majority voting.</description>
    </agent>
  </agents>
  <events>
    <event>
      <name>on_start</name>
      <inputs>
        <input>
          <key>math_problem</key>
          <description>The math problem that needs to be solved.</description>
        </input>
      </inputs>
      <outputs>
        <output>
          <key>math_problem</key>
          <description>The math problem that needs to be solved.</description>
          <action>
            <type>RESULT</type>
          </action>
        </output>
      </outputs>
    </event>
    <event>
      <name>solve_with_gpt4</name>
      <inputs>
        <input>
          <key>math_problem</key>
          <description>The math problem that needs to be solved.</description>
        </input>
      </inputs>
      <task>Solve the math problem using systematic approach with GPT-4.</task>
      <outputs>

```

```

        <output>
          <key>gpt4_solution</key>
          <description>The solution from GPT-4 solver.</description>
          <action>
            <type>RESULT</type>
          </action>
        </output>
      </outputs>
    </listen>
    <agent>
      <name>Math Solver Agent</name>
      <model>gpt-4o-2024-08-06</model>
    </agent>
  </event>
  <event>
    <name>solve_with_claude</name>
    <inputs>
      <input>
        <key>math_problem</key>
        <description>The math problem that needs to be solved.</description>
      </input>
    </inputs>
    <task>Solve the math problem using systematic approach with Claude.</task>
    <outputs>
      <output>
        <key>claude_solution</key>
        <description>The solution from Claude solver.</description>
        <action>
          <type>RESULT</type>
        </action>
      </output>
    </outputs>
  </listen>
  <agent>
    <name>Math Solver Agent</name>
    <model>claude-3-5-sonnet-20241022</model>
  </agent>
  </event>
  <event>
    <name>solve_with_deepseek</name>
    <inputs>
      <input>
        <key>math_problem</key>
        <description>The math problem that needs to be solved.</description>
      </input>
    </inputs>
    <task>Solve the math problem using systematic approach with DeepSeek.</task>
    <outputs>
      <output>
        <key>deepseek_solution</key>
        <description>The solution from DeepSeek solver.</description>
        <action>

```

```

        <type>RESULT</type>
      </action>
    </output>
  </outputs>
</listen>
<event>on_start</event>
</listen>
<agent>
  <name>Math Solver Agent</name>
  <model>deepseek/deepseek-chat</model>
</agent>
</event>
<event>
  <name>aggregate_solutions</name>
  <inputs>
    <input>
      <key>gpt4_solution</key>
      <description>The solution
        from GPT-4
        solver.</description>
    </input>
    <input>
      <key>claude_solution</key>
      <description>The solution
        from Claude
        solver.</description>
    </input>
    <input>
      <key>deepseek_solution</key>
      <description>The solution
        from DeepSeek
        solver.</description>
    </input>
  </inputs>
  <task>Compare all solutions and
    determine the final answer through
    majority voting.</task>
  <outputs>
    <output>
      <key>final_solution</key>
      <description>The final
        solution determined by
        majority
        voting.</description>
      <action>
        <type>RESULT</type>
      </action>
    </output>
  </outputs>
</listen>
<event>solve_with_gpt4</event>
<event>solve_with_claude</event>
<event>solve_with_deepseek</event>
</listen>
<agent>
  <name>Vote Aggregator
    Agent</name>
  <model>deepseek/deepseek-chat</model>
</agent>
</event>
</events>
</workflow>

```

Table 5: Comparison of the single LLM (Deepseek-v3) and AI-generated Majority Voting workflow.

---

**Problem:** Simplify  $\tan 100^\circ + 4 \sin 100^\circ$ .

---

**Ground Truth:**  $-\sqrt{3}$

---

**AI-generated Majority Voting workflow:** Let me analyze all three solutions:

1. GPT-4 Solution: - Does not provide a definitive numerical answer - Focuses on theoretical approach and approximations - Inconclusive result
2. Claude Solution: - Uses supplementary angle properties - Converts  $100^\circ$  to  $180^\circ - 80^\circ$  - Arrives at  $-\sqrt{3}$  through numerical calculations - **Final answer:  $-\sqrt{3}$**
3. Deepseek Solution: - Uses similar supplementary angle properties - Also converts  $100^\circ$  to  $180^\circ - 80^\circ$  - Goes through detailed algebraic steps - **Final answer:  $\sqrt{3}$**

Analysis: 1. Two solutions provide definitive answers ( $-\sqrt{3}$  and  $\sqrt{3}$ ), while one is inconclusive 2. Both Claude and Deepseek use similar approaches with supplementary angles 3. The main difference is in the sign of the final answer 4. Claude's solution provides more explicit numerical calculations 5. Looking at the steps: -  $100^\circ$  being in quadrant II - Using supplementary angle properties -  $\tan 100^\circ$  being negative in quadrant II - The final combination with  $4 \sin 100^\circ$

The Claude solution appears more precise in tracking signs and considering quadrant properties, making its answer more reliable.

Final Answer: The final answer is  $-\sqrt{3}$ . I hope it is correct.

---

**Deepseek-v3:** To simplify  $\tan 100^\circ + 4 \sin 100^\circ$ , we proceed as follows: 1. Recall that  $\tan \theta = \frac{\sin \theta}{\cos \theta}$ . Thus:

$$\tan 100^\circ = \frac{\sin 100^\circ}{\cos 100^\circ}.$$

2. Substitute this into the original expression:

$$\tan 100^\circ + 4 \sin 100^\circ = \frac{\sin 100^\circ}{\cos 100^\circ} + 4 \sin 100^\circ.$$

...

16. Therefore, the simplified form of  $\tan 100^\circ + 4 \sin 100^\circ$  is:

$$\boxed{\sqrt{3}}.$$

Final Answer: The final answer is  $\sqrt{3}$ . I hope it is correct.

---

**Problem:** What integer  $n$  satisfies  $0 \leq n < 18$  and

$$n \equiv -11213141 \pmod{18} ?$$


---

**Ground Truth:** 13

---

**AI-generated Majority Voting workflow:** Let me compare the three solutions:

1. **GPT-4 Solution: Arrives at 13** through direct division and remainder calculation - Used division:  $11213141 \div 18$  - Found remainder of 5 for positive number - Adjusted for negative sign to get 13
2. **Claude Solution: Arrives at 13** through division and remainder - Divided  $-11213141$  by 18 - Found remainder directly - Got 13 as the result in the range  $[0,18)$
3. **Deepseek Solution: Arrives at 5 through** : - Division and remainder calculation - Used divisibility rules for 2 and 9 - Got 5 as final answer

Analyzing the solutions: - **Two solutions (GPT-4 and Claude) agree on 13** - One solution (Deepseek) gives 5 - **Both GPT-4 and Claude show clearer arithmetic progressions** - The key difference is in handling the negative sign in the modulo operation - When working with negative numbers in modulo arithmetic, we need to find the positive equivalent in the range  $[0,n)$  - The correct approach is to add the modulus (18) to the negative remainder until we get a number in the range  $[0,18)$

By majority voting and verification: - **13 is supported by two solutions** - **The reasoning in these solutions is more complete** - We can verify:  $-11213141 \equiv 13 \pmod{18}$  and  $0 \leq 13 < 18$

Final Answer: The final answer is 13. I hope it is correct.

---

**Deepseek-v3:** To solve for the integer  $n$  that satisfies  $0 \leq n < 18$  and  $n \equiv -11213141 \pmod{18}$ , we need to find the remainder when  $-11213141$  is divided by 18. Here's the step-by-step solution:

1. **Simplify the modulus:** First, note that  $-11213141 \pmod{18}$  is equivalent to finding the positive remainder when  $-11213141$  is divided by 18. Since the modulus operation is periodic with period 18, we can add multiples of 18 to  $-11213141$  until we obtain a positive number within the range  $0 \leq n < 18$ .

2. **Calculate the remainder:** To find  $-11213141 \pmod{18}$ , we can instead compute  $11213141 \pmod{18}$  and then take the negative of that result, adjusting it to be positive if necessary.

- First, compute  $11213141 \div 18$ :

$$11213141 \div 18 = 623000 \text{ with a remainder of } 11213141 - 18 \times 623000 = 11213141 - 11214000 = -859$$

This means  $11213141 \equiv -859 \pmod{18}$ .

- Now, compute  $-859 \pmod{18}$ :

$$-859 \div 18 = -48 \text{ with a remainder of } -859 - 18 \times (-48) = -859 + 864 = 5$$

So,  $-859 \equiv 5 \pmod{18}$ .

- Therefore,  $-11213141 \equiv 5 \pmod{18}$ .

3. **Final answer:** **The integer  $n$  that satisfies  $0 \leq n < 18$  and  $n \equiv -11213141 \pmod{18}$  is 5.**

Final Answer: The final answer is 5. I hope it is correct.

---

**Problem:** In the diagram below, we have  $\sin \angle RPQ = \frac{7}{25}$ . What is  $\cos \angle RPS$ ?

[asy]

```
pair R,P,Q,SS;
SS = (-2,0);
P = (0,0);
Q = (2,0);
R = rotate(aSin(7/25))*(1.5,0);
dot("S",SS,S);
dot("Q",Q,S);
dot("R",R,N);
dot("P",P,S);
draw(Q--SS);
draw(P--R);
```

[/asy]

**Ground Truth:**  $-\frac{24}{25}$

**AI-generated Majority Voting workflow:** Let me analyze all three solutions:

1. GPT-4 Solution: Final answer =  $\frac{24}{25}$  - Used right triangle approach - Found cosine of  $\angle RPQ = 24/25$  - Assumed  $\angle RPS = \angle RPQ$
2. Claude Solution: Final answer =  $-\frac{24}{25}$  - Used supplementary angles property - Found  $\cos \angle RPQ = 24/25$  - Used the fact that  $\cos(\text{supplementary angles})$  are negatives of each other - **Concluded  $\cos \angle RPS = -\cos \angle RPQ = -24/25$**
3. DeepSeek Solution: Final answer =  $\frac{7}{25}$  - Used complementary angles property - **Found  $\cos \angle RPQ = 24/25$**  - Incorrectly concluded  $\cos \angle RPS = \sin \angle RPQ$

Analyzing the approaches: 1. Looking at the diagram,  $\angle RPS$  and  $\angle RPQ$  are supplementary angles (they form a straight line) 2. For supplementary angles, their cosines are negatives of each other 3. Claude's solution correctly uses this property 4. GPT-4 incorrectly assumed the angles were equal 5. DeepSeek incorrectly used complementary angle properties

**Claude's solution is the most mathematically sound** because: - It correctly identifies that  $\angle RPS$  and  $\angle RPQ$  are supplementary angles - It properly applies the relationship between cosines of supplementary angles - The negative sign in the answer makes geometric sense given the angle's position

Final Answer: The final answer is  $-\frac{24}{25}$ . I hope it is correct.

**Deepseek-v3:** We are given that  $\sin \angle RPQ = \frac{7}{25}$ . Since  $\angle RPQ$  and  $\angle RPS$  are complementary angles (they form a right angle together), we have:

$$\cos \angle RPS = \sin \angle RPQ = \frac{7}{25}.$$

Final Answer: The final answer is  $\frac{7}{25}$ . I hope it is correct.

**Problem:** In the circle with center  $Q$ , radii  $AQ$  and  $BQ$  form a right angle. The two smaller regions are tangent semicircles, as shown. The radius of the circle with center  $Q$  is 14 inches. What is the radius of the smaller semicircle? Express your answer as a common fraction.

**Ground Truth:**  $\frac{14}{3}$

**AI-generated Majority Voting workflow:** Let me analyze all three solutions:

1. GPT-4's solution:  $r = 7$  2. Claude's solution:  $r = 14/3$  3.

Deepseek's solution:  $r = 14(\sqrt{2} - 1)$

Each solution uses different approaches: - GPT-4 assumes the semicircles are equal and uses a simple division ( $14/2 = 7$ ) - Claude uses the relationship  $r + r + r = R$ , leading to  $14/3$  - Deepseek uses the Pythagorean theorem and complex algebra, arriving at  $14(\sqrt{2} - 1)$

Let's verify which one makes more sense:

1. Key geometric principles suggest that when two circles are tangent to each other and to two perpendicular lines (in this case, the radii forming the right angle), there's a special relationship where  $r + r + r = R$ , where  $r$  is the radius of the smaller circles and  $R$  is the radius of the larger circle.

2. This principle is well-established in geometry. When two circles are placed in a right angle and are tangent to each other and the lines forming the right angle, their radii follow this 3:1 ratio.

3. Therefore, Claude's solution using  $3r = 14$  leading to  $r = 14/3$  is the most mathematically sound.

4. We can verify this: - It satisfies the tangency conditions - The semicircles fit perfectly in the right angle - The relationship maintains the proper geometric proportions

The GPT-4 solution ( $r = 7$ ) is too large as it would make the semicircles overlap. The Deepseek solution  $14(\sqrt{2} - 1)$  is more complex than necessary and doesn't align with the fundamental geometric principle for this configuration.

Final Answer: The final answer is  $\frac{14}{3}$ . I hope it is correct.

---

**Deepseek-v3:** Let's solve the problem step by step.

1. Understand the setup: We have a circle with center  $Q$  and radius 14 inches. Radii  $AQ$  and  $BQ$  form a right angle, meaning the angle  $\angle AQB = 90^\circ$ . There are two smaller semicircles tangent to the larger circle and to each other.

2. Key observations: - The two semicircles are tangent to the larger circle and to each other. This means the distance from the center  $Q$  to the center of each semicircle is  $14 - r$ , where  $r$  is the radius of the smaller semicircle. - The centers of the two semicircles and the center  $Q$  form a right triangle because  $\angle AQB = 90^\circ$ .

...

5. Simplify the answer: - The radius of the smaller semicircle is  $14(\sqrt{2} - 1)$ . However, this can be expressed as a common fraction:

$$r = 14(\sqrt{2} - 1) = 14\sqrt{2} - 14.$$

$$r = \frac{14\sqrt{2} - 14}{1}.$$

$$r = \frac{14(\sqrt{2} - 1)}{1}.$$

$$r = 14(\sqrt{2} - 1).$$

6. Final Answer: The radius of the smaller semicircle is  $14(\sqrt{2} - 1)$ .

Final Answer: The final answer is  $14(\sqrt{2} - 1)$ . I hope it is correct.




---

---

#### **A.7.4 Detailed Trajectory of Creating 'Financial Agent'**

To provide a more comprehensive demonstration of how AutoAgent receives user requirements and automates the creation of tools and agents, we present the specific agent trajectory for the creation of the Financial Agent as follows.

Table 6: Detailed Trajectory of Creating 'Financial Agent'.

Role	Action	Content
 User	Input	<p>I want to create <code>`Financial Agent`</code>            ⇨ that can help me to do two kinds            ⇨ of tasks:            1. Manage the private financial            ⇨ docs. I have a folder that            ⇨ contain the financial docs in my            ⇨ local machine, and I want to help            ⇨ me to manage them.            2. Search the financial information            ⇨ online. You may help me to:            - get balance sheets for a given            ⇨ ticker over a given period.            - get cash flow statements for a            ⇨ given ticker over a given period.            - get income statements for a given            ⇨ ticker over a given period.</p> <p>Directly output the form in the XML            ⇨ format without ANY other text.</p>
 Agent Profiling Agent	output	<pre> &lt;agents&gt;   &lt;system_input&gt;     Financial management     ⇨ requests, including:     ...   &lt;/system_input&gt;   &lt;system_output&gt;     &lt;key&gt;financial_response&lt;/key&gt;     ...   &lt;/system_output&gt;   &lt;agent&gt;     &lt;name&gt;Document Manager     ⇨ Agent&lt;/name&gt;     ...   &lt;/agent&gt;   &lt;agent&gt;     &lt;name&gt;Market Research     ⇨ Agent&lt;/name&gt;     ...   &lt;/agent&gt; &lt;/agents&gt;           </pre>
 Tool Editor Agent	list_tools()	<pre> {   "save_raw_docs_to_vector_db": {     "name":     ⇨ "save_raw_docs_to_vector_db",     "func_name":     ⇨ "save_raw_docs_to_vector_db",     "args": [       "context_variables",       "doc_name",       "saved_vector_db_name",       "overwrite"     ],     "docstring":     ...   },   ... }           </pre>

<b>👤: Tool Editor Agent</b>	<pre>get_api_plugin_tools_doc ('query_text'='API for ↳ getting financial statements (balance sheet, ↳ cash flow, income ↳ statement) from stock market')</pre>	<p>The rank 1 referenced tool  ↳ documentation is:  API Name: Company Income Statement  ...  The rank 2 referenced tool  ↳ documentation is:  ...  ...</p>
<b>👤: Tool Editor Agent</b>	<pre>create_tool( 'tool_name'= 'get_balance_sheet', 'tool_code'= 'from metachain.types ↳ import Result, Agent ... ')</pre>	<p>[SUCCESS] Successfully created tool:  ↳ get_balance_sheet in  ↳ /workplace/metachain/metachain/ tools/get_balance_sheet.py</p>
<b>👤: Tool Editor Agent</b>	<pre>run_tool( 'tool_name'= 'get_balance_sheet', 'run_code'= 'from metachain.tools ↳ import get_balance_sheet result = get_balance_sheet("AAPL", ↳ "QUARTERLY") print(result)')</pre>	<p>[The output of the tool `run_tool`  ↳ showing in the interactive  ↳ terminal]  Terminal viewport position: Showing  ↳ page 1 of 1.  =====</p> <pre>[SUCCESS] {   "symbol": "AAPL:NASDAQ",   "type": "stock",   "period": "QUARTERLY",   "balance_sheet": [     ...   ] }</pre> <p>=====</p>
<b>👤: Tool Editor Agent</b>	<pre>create_tool( 'tool_name'= 'get_cash_flow', 'tool_code'= 'from metachain.types ↳ import Result, Agent ... ')</pre>	<p>[SUCCESS] Successfully created tool:  ↳ get_cash_flow in  ↳ /workplace/metachain/metachain/ tools/get_cash_flow.py</p>
<b>👤: Tool Editor Agent</b>	<pre>run_tool( 'tool_name'= 'get_cash_flow', 'run_code'= 'from metachain.tools ↳ import get_cash_flow result = get_cash_flow("AAPL", ↳ "QUARTERLY") print(result)')</pre>	<p>[The output of the tool `run_tool`  ↳ showing in the interactive  ↳ terminal]  Terminal viewport position: Showing  ↳ page 1 of 1.  =====</p> <pre>[SUCCESS] {   "symbol": "AAPL:NASDAQ",   "type": "stock",   "period": "QUARTERLY",   "cash_flow": [     ...   ] }</pre> <p>=====</p>
<b>👤: Tool Editor Agent</b>	<pre>create_tool( 'tool_name'= 'get_income_statement', 'tool_code'= 'from metachain.types ↳ import Result, Agent ...')</pre>	<p>[SUCCESS] Successfully created tool:  ↳ get_income_statement in  ↳ /workplace/metachain/metachain/ tools/get_income_statement.py</p>

<b>👤: Tool Editor Agent</b>	<pre>run_tool( 'tool_name'= 'get_income_statement', 'run_code'='from ↳ metachain.tools import ↳ get_income_statement result = ↳ get_income_statement ("AAPL", "QUARTERLY") print(result)')</pre>	<pre>[The output of the tool `run_tool` ↳ showing in the interactive ↳ terminal] Terminal viewport position: Showing ↳ page 1 of 1. ===== [SUCCESS] {   "symbol": "AAPL:NASDAQ",   "type": "stock",   "period": "QUARTERLY",   "income_statement": [     ...   ] } =====</pre>
<b>👤: Tool Editor Agent</b>	<pre>create_tool( 'tool_name'= 'analyze_financial_data', 'tool_code'= 'from metachain.types ↳ import Result, Agent ... ')</pre>	<pre>[SUCCESS] Successfully created tool: ↳ analyze_financial_data in ↳ /workplace/metachain/metachain/ tools/analyze_financial_data.py</pre>
<b>👤: Tool Editor Agent</b>	<pre>run_tool( 'tool_name'= 'analyze_financial_data', 'run_code'= 'from metachain.tools ↳ import ↳ get_income_statement, ↳ analyze_financial_data income_data = ↳ get_income_statement ("AAPL", "QUARTERLY") result = ↳ analyze_financial_data (income_data) print(result)')</pre>	<pre>[The output of the tool `run_tool` ↳ showing in the interactive ↳ terminal] Terminal viewport position: Showing ↳ page 1 of 1. ===== [SUCCESS] Financial Analysis for AAPL:NASDAQ ----- Income Statement Analysis: For the period ending 2024-9-28: ...  Profitability Metrics: ...  Period-over-Period Comparison: ... =====</pre>
<b>👤: Tool Editor Agent</b>	<p>Output</p>	<pre>Successfully created and tested all ↳ four financial tools:  1. get_balance_sheet - Created and ↳ tested successfully with AAPL ↳ stock data 2. get_cash_flow - Created and ↳ tested successfully with AAPL ↳ stock data 3. get_income_statement - Created ↳ and tested successfully with AAPL ↳ stock data 4. analyze_financial_data - Created ↳ and tested successfully by ↳ analyzing AAPL's income statement ↳ data  All tools are functioning as ↳ expected, providing financial ↳ data and analysis in proper JSON ↳ format.</pre>

**👤: Agent Editor  
Agent**

```
create_agent(  
'agent_name'=  
'Document Manager Agent',  
'agent_description'=  
'Specialized agent for  
↳ managing and analyzing  
↳ private financial  
↳ documents stored  
↳ locally.',  
'agent_tools'=  
['query_db', ...],  
'agent_instructions'='You  
↳ are a specialized  
↳ financial document  
↳ manager responsible for  
↳ managing and analyzing  
↳ financial documents  
↳ stored in specified  
↳ directories. Your tasks  
↳ include:...'')
```

Successfully created agent: Document  
↳ Manager Agent in  
↳ /workplace/metachain/metachain/  
agents/document\_manager\_agent.py

**👤: Agent Editor  
Agent**

```
create_agent(  
'agent_name'=  
'Market Research Agent',  
'agent_description'=  
'Specialized agent for  
↳ retrieving and analyzing  
↳ online financial  
↳ information for publicly  
↳ traded companies.',  
'agent_tools'=  
['get_cash_flow', ...],  
'agent_instructions'=  
'You are a specialized  
↳ market research agent  
↳ responsible for  
↳ retrieving and analyzing  
↳ financial information  
↳ from online sources.  
↳ Your primary  
↳ responsibilities  
↳ include: ...'')
```

Successfully created agent: Market  
↳ Research Agent in  
↳ /workplace/metachain/metachain/  
agents/market\_research\_agent.py


 Agent Editor  
Agent

```
create_orchestrator_agent
('agent_name'=
'Financial Analysis
↳ Orchestrator',
'agent_description'=
'Orchestrator agent that
↳ coordinates between
↳ document management and
↳ market research to
↳ provide comprehensive
↳ financial analysis.',
'sub_agents'=[{'name':
↳ 'Document Manager
↳ Agent', 'agent_input':
↳ 'Document analysis
request ...',
↳ 'agent_output': 'Analyzed
information ...'}, {'name':
↳ 'Market Research Agent',
↳ 'agent_input': 'Market
research request ...',
↳ 'agent_output': 'Market
analysis ...'}],
'agent_instructions'=
'You are a Financial
↳ Analysis Orchestrator
↳ responsible for
↳ coordinating
↳ comprehensive financial
↳ analysis by leveraging
↳ both document analysis
↳ and market research
↳ capabilities. Your
↳ workflow: ...')
```

```
[ERROR] Failed to create agent.
↳ Error: File
↳ "/workplace/metachain/metachain/
agents/financial_analysis_orchestrator.py",
↳ line 32
def
↳ transfer_to_document_manager_agent(Document
↳ analysis request including
↳ directory paths and analysis
↳ requirements: str):
```


^^^^^^^^

SyntaxError: invalid syntax

 **Agent Editor**  
**Agent**

```
create_orchestrator_agent
('agent_name'=
'Financial Analysis
↳ Orchestrator',
'agent_description'=
'Orchestrator agent that
↳ coordinates between
↳ document management and
↳ market research to
↳ provide comprehensive
↳ financial analysis.',
'sub_agents'=[{'name':
↳ 'Document Manager
↳ Agent', 'agent_input':
↳ 'doc_request',
↳ 'agent_output':
↳ 'doc_response'},
↳ {'name': 'Market
↳ Research Agent',
↳ 'agent_input':
↳ 'market_request',
↳ 'agent_output':
↳ 'market_response'}]],
'agent_instructions'=
'You are a Financial
↳ Analysis Orchestrator
↳ responsible for
↳ coordinating
↳ comprehensive financial
↳ analysis by leveraging
↳ both document analysis
↳ and market research
↳ capabilities. Your
↳ workflow: ...')
```

Successfully created agent:  
↳ Financial Analysis Orchestrator  
↳ in /workplace/metachain/metachain/  
agents/financial\_analysis\_orchestrator.py

 Agent Editor  
Agent

```
run_agent(  
  'agent_name'=  
  'Financial Analysis  
  ↳ Orchestrator',  
  'query'=  
  "Based_on_the_10-K_reports_  
  ↳ of_AAPL_and_MSFT_from_  
  ↳ the_past_5_years_in_the_  
  ↳ docs_directory`docs/  
  aapl-2020-2024-10K/`and_  
  ↳ `docs/  
  msft-2020-2024-10K/`,_along_  
  ↳ with_AAPL's_other_  
  ↳ reports`docs/  
  aapl-other-report/`and_  
  ↳ available_data,_conduct_  
  ↳ a_comprehensive_  
  ↳ horizontal_comparison,_  
  ↳ create_a_comparative_  
  ↳ analysis_report,_and_  
  ↳ provide_constructive_  
  ↳ investment_advice_for_  
  ↳ investing_in_them_in_  
  ↳ 2025.",  
  'ctx_vars'={'{}'})
```

```
[The output of the tool `run_agent`  
↳ showing in the interactive  
↳ terminal]  
Terminal viewport position: Showing  
↳ page 1 of 35.  
[NOTE] The output of the tool  
`run_agent`, you can use  
`terminal_page_up` to scroll up and  
`terminal_page_down` to scroll down.  
If there are many pages with  
meaningless content like progress bar  
or output of generating directory  
structure when there are many datasets  
in the directory, you can use  
`terminal_page_to` to move the  
viewport to the end of terminal where  
the meaningful content is.  
=====
```

```
[SUCCESS]  
**** Receive Task ****  
Receiving the task:  
Based on the 10-K reports of AAPL  
↳ and MSFT from the past 5 years in  
↳ the docs  
directory ...  
**** Assistant Message ****  
Financial Analysis Orchestrator:  
↳ I'll help coordinate a  
↳ comprehensive comparative  
↳ analysis between Apple (AAPL)  
and Microsoft (MSFT) using both  
↳ historical document analysis and  
↳ current market  
research. Let's break this down into  
↳ parallel workflows to gather all  
↳ necessary  
information.  
  
1. First, let's analyze the  
↳ historical documents (10-K  
↳ reports and other  
reports):  
**** Tool Calls ****  
[2025-01-22 05:58:48]  
transfer_to_document_manager_agent  
(`doc_request`= '..')  
**** Tool Execution ****  
[2025-01-22 05:58:48]  
Tool Execution:  
↳ transfer_to_document_manager_agent  
Result:  
---  
Please analyze the following  
↳ documents:  
...  
  
**** Assistant Message ****  
Document Manager Agent: ...  
=====
```

👤: Agent Editor  
Agent

terminal\_page\_to  
( 'page\_idx'=35)

```
[The output of the tool `run_agent`  
↳ showing in the interactive  
↳ terminal]  
Terminal viewport position: Showing  
↳ page 35 of 35.  
[NOTE] The output of the tool  
'run_agent', you can use  
'terminal_page_up' to scroll up and  
'terminal_page_down' to scroll down.  
If there are many pages with  
meaningless content like progress bar  
or output of generating directory  
structure when there are many datasets  
in the directory, you can use  
'terminal_page_to' to move the  
viewport to the end of terminal where  
the meaningful content is.  
=====
```

\*\*\* Assistant Message \*\*\*

Financial Analysis Orchestrator:  
↳ Based on the comprehensive  
↳ analysis of both historical  
↳ performance from 10-K  
reports and current financial data,  
↳ here is the final investment  
↳ recommendation  
for 2025:

Investment Recommendation Summary  
↳ for 2025:

1. Portfolio Allocation Strategy:
  - Microsoft (MSFT): 60% of  
↳ technology allocation
  - Apple (AAPL): 40% of technology  
↳ allocation
2. Investment Rationale:

Microsoft (MSFT) - Primary Position  
↳ (60%):  
...  
=====

🤖: Agent Editor Agent      Output

I have successfully created and  
↳ tested the following agents:

1. Document Manager Agent - for  
↳ managing and analyzing private  
↳ financial documents
2. Market Research Agent - for  
↳ retrieving and analyzing online  
↳ financial information
3. Financial Analysis Orchestrator -  
↳ coordinates between the two  
↳ agents for comprehensive analysis

The agents were tested with a  
↳ complex task of analyzing AAPL  
↳ and MSFT's financial documents  
↳ and providing investment advice.  
↳ The orchestrator successfully:

1. Processed 10-K reports from both  
↳ companies
2. Analyzed financial metrics and  
↳ trends
3. Generated a comprehensive  
↳ investment recommendation with:
  - Portfolio allocation strategy
  - Investment rationale for each  
↳ company
  - Key monitoring points
  - Risk management strategy
  - Implementation approach

The test demonstrated that the  
↳ agents can effectively work  
↳ together to provide detailed  
↳ financial analysis and actionable  
↳ investment advice.

## **A.8 LLM usage**

In the writing of this paper, the LLM was used only for correcting word choice and grammar. The LLM was employed exclusively as a research subject in the development of agents and was not involved in any critical stages of human-driven research, including idea generation, data analysis, or other core research activities.