What is MetaGPT?

MetaGPT overview

MetaGPT is an open source multiagent framework orchestrating the use of human procedural knowledge and AI agents powered by large language models (LLMs) to develop a diverse range of software solutions. Acting as an AI software company, MetaGPT features specialized AI agents that perform roles like those in traditional software companies.

MetaGPT is the proprietary technology of DeepWisdom founded by Chenglin Wu. The framework gained popularity quickly on GitHub, garnering interest around MetaGPT's underlying goal to propel the advancement of natural language programming using multiagent systems (MAS).

The popular research paper "MetaGPT: Metaprogramming for A Multi-Agent Collaborative Framework" explains the framework's orchestration of agent collaboration. As research surrounding LLM orchestration and AI agents continues, MetaGPT is an early contributor offering a straightforward approach to multiagent collaboration by using familiar workflows.

Your subscription will be delivered in English. You will find an unsubscribe link in every newsletter. You can manage your subscriptions or unsubscribe here. Refer to our IBM Privacy Statement for more information.

What is multiagent collaboration?

Agents acting within a multiagent system (MAS) remain autonomous but also cooperate and coordinate in agentic structures, or frameworks such as MetaGPT.[1] A MAS coordinates individual agents to operate and interact within their environment to complete complex tasks to achieve a common goal. This idea is a part of collective intelligence, a popular concept sometimes related to AGI.

A multiagent system consists of a network of problem-solving agents that work together to tackle challenges beyond the capacity of any single agent.[2] One of the key challenges in designing an effective MAS is coordinating the agents to ensure their collaboration on a shared objective. Successful collaboration relies on each agent's ability to solve individual tasks, as well as their ability to interact effectively with one another.[3]

The considerations in planning a MAS environment include:

- The limitations that other agents' activities create for an agent's decision-making,
- The constraints that arise from an agent's commitments to others regarding its available actions,
- And the unpredictable evolution of the world that is caused by external, unmodeled agents.[4]

One approach to this design challenge is the orchestration of agent collaboration by explicitly modeling teamwork. This multicollaborative design pattern breaks down a complex prompt and delegates the abstracted tasks to agents who execute them based on their specialized roles.[5]

Within MetaGPT's multiagent collaborative framework, a team of AI agents operates according to a structured workflow guided by standard operating procedures (SOPs) that define distinct roles and instructions. By integrating communication protocols and streamlined workflows within LLM-driven multiagent systems, users can create applications with a single line of input.

How does MetaGPT work?

MetaGPT acts as a multiagent collaborative framework capable of solving complex tasks. The multiagent collaboration framework works by simulating an entire software company using specialized agents that interact based on standard operating procedures and an assembly-line paradigm to break up tasks. Each agent plays a specialized role according to its functions within the software company. For example, the framework includes agents that act as product managers, architects, project managers and engineers that provide the entire process of a software company.

These agents in their different roles act within a virtual workspace to take a one-line requirement as input and output from various software solutions. This input includes data structures, application programming interfaces (APIs), competitive analysis and documents. Agents communicate through structured outputs based on requirements set by an assembly-line paradigm. Each agent generates the necessary information to prompt the next agent to complete the collective goal: producing a software application based on the user prompt. This incremental development process is due to how MetaGPT integrates human workflows to streamline coordination on a complex task.

Human procedural knowledge for multiagent collaboration

Standard operating procedures have been adopted across various domains as they play a critical role in supporting task decomposition and effective coordination.[6] In software development, SOPs promote collaboration among teams by deconstructing project tasks into smaller, actional procedures that are then assigned to specific, specialized roles. MetaGPT encodes SOPs into prompt sequences to organize and promote multiagent workflows. SOP workflows introduce structure and opportunities for refinement.

For example, the project manager agent is prompted to create a product requirement document (PRD). The product manager agent is prompted once and then asked to refine certain elements of the PRD due to the incremental development of the project. For example, the agent is instructed to create a section within the PRD for "product goals." It is then prompted again to refine those goals by updating the original product goals to ensure that they align with the current project direction.[7] This refinement method is employed to ensure that the global task is recognized and cohesive across all agents.

In the example below, the product manager agent is prompted to generate a requirement analysis. After this initial prompt, the agent is prompted to generate a refined analysis that takes any new features and enhancements from ongoing collaboration into consideration.

Requirement analysis prompt instruction: *"Provide a detailed analysis of the requirements."*[8]

Refined analysis prompt instruction: *"Review and refine the existing requirement analysis into a string list to align with the evolving needs of the project due to incremental development. Ensure that the analysis comprehensively covers the new features and enhancements required for the refined project scope."*[9]

This example demonstrates a PRD generated as standardized output that is passed from the product manager agent to the architect agent. The SOP-based prompt for each generated output is listed beside them. Notice that the prompt is similarly based on the SOP workflow, including requirement definitions, methods for task completion and the order to complete them:

User prompt: *"Build me a React based app-based web application for wealth managers to use to review their clients' portfolios and make recommendations of appropriate funds for them to use. This application needs to work in America, the United Kingdom and Spain."*

Original requirements
Prompt instruction: *"Place the original user's requirements here."*[10]

"Build me a React based app-based web application for wealth managers to use to review their clients' portfolios and make recommendations of appropriate funds for them to use. This application needs to work in America, the United Kingdom and Spain."

Product goals
Prompt instruction: *"Provide up to three clear, orthogonal product goals."*[11]

1. Develop a secure and reliable web application for wealth managers.
2. Ensure multilanguage support for English, Spanish and regional variations.
3. Create an intuitive and user-friendly interface for portfolio management and recommendation.

User stories
Prompt instruction: *"Provide up to 3 to 5 scenario-based user stories."*[12]

1. As a wealth manager, I want to review my clients' portfolios and provide personalized advice.
2. As a wealth manager, I want to recommend funds to my clients based on their investment goals and risk tolerance.
3. As a wealth manager, I want to access the application in English or Spanish depending on my location.

Competitive analysis
Prompt instruction: *"Provide 5 to 7 competitive products."*[13]

1. "Wealthfront: Automated investment service with a clean interface, but lacks personalized fund recommendations."
2. "Personal Capital: Offers comprehensive wealth management tools, but can be overwhelming for new users."

3. “Betterment: Focuses on automated investing with limited customization for wealth managers."

Requirement analysis
Prompt instruction: *“Provide a detailed analysis of the requirements.”*[14]

The application must be built using React, ensuring it is scalable and maintainable. It should have a multilanguage interface to cater to users in America, the United Kingdom and Spain. The application must be secure to protect sensitive financial data and comply with regional regulations. It should also integrate with various financial data sources to provide real-time portfolio analysis and fund recommendations.

Requirement pool
Prompt instruction: *“List down the top 5 to 7 requirements with their priority (P0, P1, P2).”, “Cover both legacy content and incremental content. Retain content unrelated to incremental development.”*[15]

- (“Secure user authentication and data encryption to protect client information.”, “P0”),
- (“Multilanguage support for English and Spanish, with regional variations.”, “P0”),
- (“Real-time integration with financial data sources for portfolio management.”, “P0”),
- (“User-friendly dashboard for reviewing and managing client portfolios.”, “P1”),
- (“Recommendation engine to suggest appropriate funds based on client profiles.”, “P2”)

UI design draft
Prompt instruction: *“Provide a simple description of the UI elements, functions, style and layout.”*[16]

The UI will feature a clean and modern design with a dashboard that provides an overview of the client portfolios. It will include interactive charts for portfolio analysis, a recommendation section for fund selection, and easy toggling between English and Spanish languages.

Anything UNCLEAR
Prompt instruction: *“Mention and aspects of the project that are unclear and try to clarify them.”*[17]

There are no unclear points.

The preceding example is a handover to the architect agent for design, here is the SOP-based prompt given along with the PRD:

"Here is the product requirement document (PRD): based on this PRD review whether this API design meets the requirements of the PRD, and whether it complies with good design practices.”[18]

The architect follows its own SOP workflow to design the system for the application.

An SOP outlines the responsibilities of each agents’ role and establishes standards for intermediate outputs. These SOPs are then encoded to influence the agent’s behavior. This practice allows agents to generate structured outputs, such as high-quality requirements documents, design artifacts, flowcharts and interface specifications. All handovers between agents must comply with certain established standards that reduce the risk of hallucination caused by idle chatter between LLMs. The use of structured outputs greatly enhances the success rate of target code generation.[19]

How MetaGPT uses SOPs to prompt agents

All agents within MetaGPT act as employees that must follow a strict and streamlined workflow. There are two main parts within the SOP that define the agents' behavior: role specialization and workflow across agents.

Agent role specialization: MetaGPT defines five roles within the software company: product manager, architect, project manager, engineer and QA engineer. Each agent's profile is initialized with specific information such as the agent name, profile, goal and the constraints for each role as well as the specific context and skills for each role.[20] Each agent is like a digital organism acting within an environment.[21] The concept of predefined roles is dissimilar to multiagent frameworks like CrewAI that allow users to define the agent's function within a crew for more general-purpose use cases.

Workflow across agents: Defining the agents' roles and operational skills establishes a basic workflow for agents to follow within the software development process. Agents work within a sequential order or an assembly line paradigm to break down complex tasks for greater efficiency among the team.

How do agents interact in MetaGPT?

MetaGPT posits that meaningful collaboration requires effective, coherent and accurate problem-solving processes. AI agents are complex systems; however, their processes don't have to be, at least not according to MetaGPT and its inheritance of straightforward human workflows.

Communication protocol

Agents interact within a structured communication interface referred to as the communication protocol. MetaGPT differs from most LLM-based multiagent frameworks because it does not use unconstrained natural language as a communication interface and instead proposes the use of structured communication to formulate the interaction between agents. For example, in ChatDev, another multiagent collaboration framework, agents communicate through dialogue whereas agents in MetaGPT communicate through structured outputs like documents and diagrams.

To facilitate this communication protocol, MetaGPT establishes a schema and format for each agent and requests that each different role provides the necessary outputs based on their specific goal and context.[22] In an example, the architect agent generates two outputs: the system interface design and a sequence flow diagram. Both outputs contain system module design and interaction sequences that serve as paramount deliverables for engineer agents.[23]

Publish and subscribe mechanism

To enhance communication efficiency, MetaGPT uses a global message pool to store information that allows agents to exchange messages directly. Agents publish their structured messages in the pool and can access messages from other agents transparently. This approach enables agents to directly access the necessary information from the shared pool, removing the need to ask other agents and wait for their replies.[24]

MetaGPT development process

The development process begins with a user's input command and ends with software designed according to the user's specifications. For example, the user input might be "Write a Python GUI application such that you can draw an image with it." The user prompt is passed on to the software company, a team of role-playing agents: product manager, architect, project manager, engineer, QA engineer

After receiving the user prompt to build a specific software application, the product manager generates a PRD that encompasses goals, user stories, competitive analysis, requirement analysis and requirement pool. In addition, the product manager agent also generates a competitive quadrant chart based on the specifications on the application. These documents and charts are handed over to the architect agent for system design.

The architect agent devises technical specifications based on the requirements in the PRD. Specifications include system architecture diagrams and interface definitions for the overall technical trajectory of the project. The project's architecture including files, classes and the sequence flow chart is designed based on the architect agent's technical definitions. The architect's generated documentation is then given to the project manager for task allocation and execution.

The project manager breaks down the project in a task list. Each code file is analyzed based on its intended functionality and then treated as a separate task assigned to engineers.

The engineer agent generates the code needed with fundamental development skills to complete the development tasks. Upon receiving the code output from the engineer agent, the QA engineer agent generates unit test code and reviews it to identify and fix any bugs.

LLM integration in MetaGPT

MetaGPT's agents are based on OpenAI's series of GPT (generative pre-trained transformer) models GPT-3.5 and GPT-4. However, MetaGPT and its open source community have contributed to the addition of several other models that can be initialized through LLM API configuration. MetaGPT offers a tutorial that facilitates the exploration of how to integrate with open source LLMs on its GitHub docs site. The first step to integrating an LLM is to set up an inference repository (repo) such as LLaMA-Factory, FastChat, Ollama and so on. This repository enables deployment of the corresponding LLM model that is configured through its API credentials. All supported inference repos except Ollama support publishing OpenAI-compatible interfaces. MetaGPT seeks to support the Ollama interface in the future.

Can AI agents excel at metaprogramming?

Metaprogramming frameworks provide the functionality to create programs that can write, manipulate and analyze other programs as well as the program itself. MetaGPT models this purpose by incorporating human workflows to power generative AI using agent-based techniques to enhance metaprogramming.

LLM-based agents contain several core capabilities that have advanced automatic programming tasks.[25] Among those advancements are ReAct and Reflexion, reasoning paradigms used by agents that employ a chain of thought prompt to generate reasoning trajectories and action plans with LLMs.[26]

The ReAct agent design loop demonstrates an effective process for automatic programming because of its iterative design loop that empowers agents to reason, act and observe.[27] Reflexion reinforces language agents through linguistic feedback, following a similar iterative design loop to induce better decision-making.[28] Both paradigm designs allow agents to continuously learn and improve their workflow.

Traditional LLMs, such as OpenAI's GPT-3 model, Meta's Llama models and the IBM Granite™ models are limited in their knowledge and reasoning. They generate responses based on the training data, which may frequently include outdated information. In contrast, agentic technology leverages backend tool-calling to access current information, streamline workflows and autonomously create specific tasks to accomplish complex objectives. During this process, the autonomous agent learns to adjust to user expectations over time, providing a tailored experience and more thorough responses. This tool-calling can occur without human intervention, expanding the potential real-world applications for these AI systems.

How MetaGPT uses metaprogramming agents

MetaGPT's engineer agents participate in iterative programming with executable feedback. The processes of debugging and optimization are important in everyday programming tasks. Other implementations lack agent self-correction mechanisms that lead to unwanted outcomes such as code hallucinations or nonfunctioning code. To mitigate these risks, MetaGPT introduces an executable feedback mechanism to improve code upon each iteration.

The engineer agent writes code based on the original product requirements and design. This allows the agent to continuously improve code using its own historical execution and debugging memory. The engineer obtains more information to improve the code by writing and executing the corresponding unit test cases. After receiving the results, the engineer either proceeds with additional development tasks or debugs the code before resuming programming. This iterative process continues until the test is passed or a maximum of 3 retries is reached.[29]

Other multiagent frameworks

crewAI – CrewAI is an open source, python-based multiagent framework leveraging role-playing autonomous agents that work together as a crew to complete tasks. Users can build and customize agents based on their application needs or use. Use cases include generalized purposes such as content planning and creation, data analysis and automation tasks. CrewAI features integration with IBM’s watsonx.ai™ and offers several LLM integrations and compatibility with Ollama.[30]

ChatDev – ChatDev is an open source multiagent framework that simulates a virtual software company operating through various intelligent agents holding different organizational roles. Agents cooperate through dialogue to produce a software product including executable code and documentation. ChatDev supports OpenAI’s GPT-3.5-turbo and GPT-4 models to power their intelligent agents.[31]

AutoGPT – AutoGPT is an open source multiagent framework that applies natural language processing (NLP) to understand user goals and complex task decomposition. Agents cooperate in an automated workflow to process a high-level user prompt by breaking down each task into a sequence of smaller subtasks. Use cases include generalized solutions such as market research and analysis, product development, virtual assistance and more. This framework is built with AI agents based on OpenAI’s GPT-4.[32]