
AGENT DESIGN PATTERN CATALOGUE: A COLLECTION OF ARCHITECTURAL PATTERNS FOR FOUNDATION MODEL BASED AGENTS

Yue Liu, Sin Kit Lo, Qinghua Lu, Liming Zhu, Dehai Zhao, Xiwei Xu, Stefan Harrer, Jon Whittle

Data61, CSIRO, Australia

Email: *firstname.lastname@data61.csiro.au*

November 7, 2024

Foundation model-enabled generative artificial intelligence facilitates the development and implementation of agents, which can leverage distinguished reasoning and language processing capabilities to take a proactive, autonomous role to pursue users' goals. Nevertheless, there is a lack of systematic knowledge to guide practitioners in designing the agents considering challenges of goal-seeking (including generating instrumental goals and plans), such as hallucinations inherent in foundation models, explainability of reasoning process, complex accountability, etc. To address this issue, we have performed a systematic literature review to understand the state-of-the-art foundation model-based agents and the broader ecosystem. In this paper, we present a pattern catalogue consisting of 18 architectural patterns with analyses of the context, forces, and trade-offs as the outcomes from the previous literature review. We propose a decision model for selecting the patterns. The proposed catalogue can provide holistic guidance for the effective use of patterns, and support the architecture design of foundation model-based agents by facilitating goal-seeking and plan generation.

1 Introduction

Being the technical backbones of the highly disruptive generative artificial intelligence (GenAI) technologies, foundation models (FMs) have received a vast amount of attention from academia and industries [1]. Specifically, the emergence of large language models (LLMs) with their remarkable capabilities to understand and generate human-like reasoning and content has sparked the growth of a diverse range of downstream tasks using language models. Subsequently, there is a rapidly growing interest in the development of FM-based autonomous agents, e.g., AutoGPT¹ and BabyAGI², which can take a proactive, autonomous role to pursue users' goals. This goal could be broad given by human, necessitating the agents to derive their autonomy from the capabilities of FMs, enabling them to segregate the goal into a set of executable tasks and orchestrate task execution to fulfill the goal. During the reasoning process, humans can also provide feedback on instrumental goals, revise a multi-step plan derived by the agent, correct intermediate results, or even refine a plan/goal during execution based on early outcomes.

While huge efforts have been put into this merging field, there is a steep learning curve for practitioners to build and implement FM-based agents. We noticed that there are a series of reusable solutions that can be grouped into patterns to address the diverse challenges in designing FM-based agents, however, the architecture design and architectural patterns collection of the agents have not been systematically explored and formulated. Furthermore, the design of systems that integrate agents is non-trivial and complex, especially in how to select appropriate design decisions to fulfill different software quality requirements and design constraints. Further, multi-agent systems may require additional considerations on the coordination and interactions of agents, for instance, collusion between agents, and correlated failures [2]. We list several challenges in developing and implementing FM-based agents as follows:

- Agents often struggle to fully comprehend and execute complex tasks, leading to the potential for inaccurate responses. This challenge may be intensified by the inherent reasoning uncertainties during plan generation

¹<https://github.com/Significant-Gravitas/AutoGPT>

²<https://github.com/yoheinakajima/babyagi>

and action procedures. For instance, across a long-term planning, the included steps may depend on each other, even slight deviation to a few steps can significantly impact the overall success rate.

- Agents should not be entirely blamed for inaccurate response, since users may provide limited context, ambiguous goals or unclear instructions during the interaction with agents, which will result in underspecification [3,4] in the reasoning process and response generation of agents.
- The sophisticated internal architecture of agents and foundation models results in limited explainability, making them “black boxes” to stakeholders. Consequently, agents often struggle to interpret their reasoning steps, which can affect the reliability, robustness, and overall trustworthiness of agent systems.
- The accountability process is complicated due to the interactions between various stakeholders, FM-based agents, non-agent AI models, and non-AI software applications within the overall ecosystem. Highly autonomous agents may delegate or even create other agents or tools for certain tasks. In this circumstance, responsibility and accountability may be intertwined among multiple entities.

In this regard, we present a catalogue of patterns for foundation model-based agents in this paper, aiming to address the identified issues via providing a holistic guidance to the design and development of different types of agents, and specifying different collaboration schemes between these agents. For instance, the goal creator patterns can clarify users’ intentions and instructions to avoid underspecification. A series of patterns for reflection can help identify and mitigate the uncertainties in agent-generated plans, while the explainability of agent reasoning process is improved by requesting an agent to reflect on its generated plan. Accountability can be preserved when agents participate in a vote where their identities and operations are all logged. Please note that “agent” can be referred to i) AI acting on behalf of another entity, or; ii) AI that can take active roles or produces effect to achieve users’ goals. The former circumstance requires thorough analysis on governance perspective, while hereby, we claim that in this study, we focus on the second concept of “agents” that are capable of goal-seeking and plan generation. In software engineering, an architectural pattern is a reusable solution to a problem that occurs commonly within a given context in software design. Our pattern catalogue includes 18 patterns that were identified based on the study conducted by Lu et al. [5]. The intended audience of collected patterns is software architects and developers who are interested in FM-based agent design and implementation. The contributions of this paper include:

- The collection of architectural patterns provides a design solution pool for practitioners to select from for real-world agent implementations. For instance, architects can apply *passive goal creator* or *proactive goal creator* considering the application scenarios and the requirements for accessibility.
- The FM-based agent ecosystem with architectural pattern annotations, serving as a guide for the design and development of FM-based agents. In particular, an agent can request feedback from both human and other agents for reasoning certainty and improved explainability, and there are three cooperation schemes for multiple-agent systems with different accountability processes.
- The curated analysis of each included pattern regarding the application context, addressed issues, consequent benefits and trade-offs on software quality attributes, real-world known uses, and the relationship with other patterns.
- A decision model that can help architects structure the included patterns and make rational design decisions on foundation model based agents. We also share the experiences on pattern application in different research projects.

The remainder of this paper is organised as follows. Section 2 introduces background knowledge and discusses related work. The methodology of this research is introduced in Section 3 and Section 4 presents each pattern in detail with our extended pattern template. Section 5 illustrates a decision model for pattern selection, and discusses the insights we obtained in this research project, while Section 6 concludes the paper.

2 Background & Related Work

The introduction of OpenAI’s ChatGPT [6] in November 2022 has gathered over 100 million users in two months upon its release, becoming the fastest-growing consumer internet app of all time [7]. This has also initiated the race among big tech arms in the development of FM and GenAI products. For instance, Google rolled out its own GenAI product, the Bard models ³, then released the Gemini ⁴ as the updated version. Anthropic has also emerged as one of the major

³<https://ai.google/static/documents/google-about-bard.pdf>

⁴<https://gemini.google.com/>

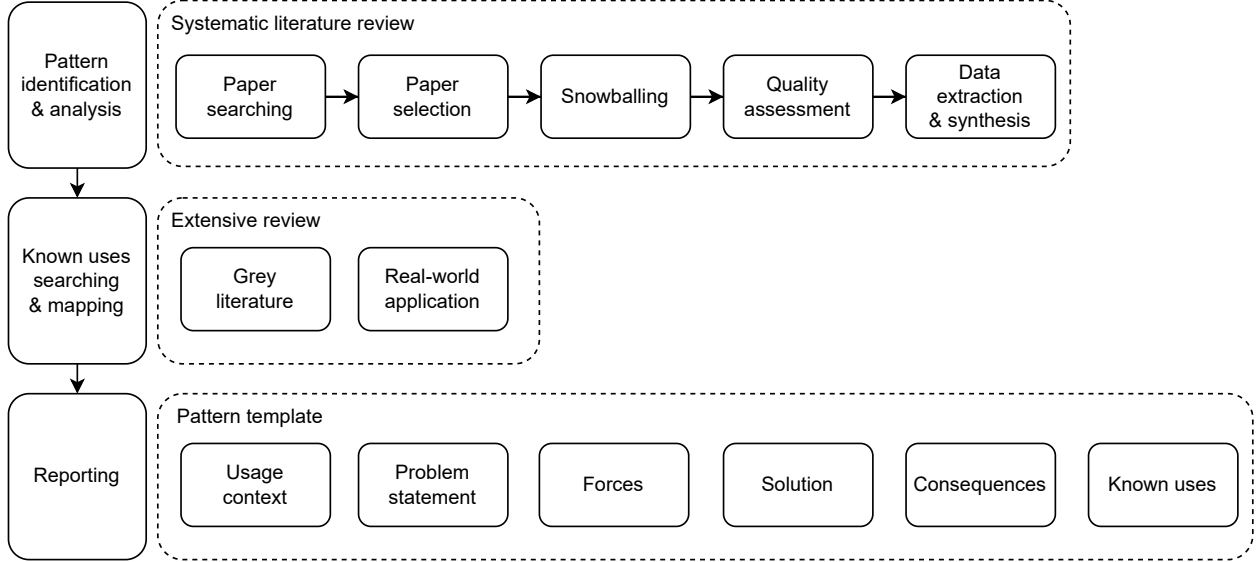


Figure 1: Methodology.

FM providers since the launch of their Claude models⁵ There are also many open-sourced FMs, such as Llama⁶ and Mistral⁷. Schulhoff et al. [8] performed a systematic literature review and proposed a taxonomy of diverse prompting techniques for foundation models.

With the explosive growth of FMs, it is highly notable that FM-based agents come into the picture. AI agents are typically designed to operate a particular software environment. A single agent is able to take actions in a variety of three-dimensional virtual worlds [9]. Recently, there have been a lot of studies that present the architecture of their agents and broader AI systems [10, 11]. LangChain analysed the cognitive architecture of agents [12]. However, these architectures often focus only on certain components or schemes. For instance, Packer et al. [10] explicitly covered the memory design of the agent. Andrew Ng discussed reflection, tool use, planning and multi-agent collaboration [9] and provided a corresponding course⁸. Jain presented four design patterns for compound AI systems, including retrieval augmented generation, conversational AI, multi-agent communication, and co-pilot [13]. Zhou et al. [14] proposed a four-tiered hierarchical artificial society model for the “cyber-physical-social” aspects of adaptive AI agents: i) an autonomous layer for agent memory, behavior and decision, ii) an evolutionary layer for agent learning and heterogeneity, iii) an interactive layer for agent collaboration, competition and topological structure, and iv) an emergent layer for the overall environment, feedback, and intervention. Yan et al. [15] shared their experience of building products with LLMs and the best practices they summarised during this process. Hassan et al. [16] demonstrated a high-level structure of FMware, consisting of Agentware and Promptware, and identified the challenges of software engineering for FMware. Gao et al. [17] proposed the roadmap for designing biomedical AI agents, and illustrated the required components.

We noticed that there is a lack of a holistic view of architecture design, making it challenging to develop FM-based agents. Moreover, software architecture comprises software elements, relations among them, and properties of both [18]. With the increase in the incorporation of machine learning into software and systems, methods to identify the impact on the reliability of machine learning are essential to ensure the reliability of the software and systems in which these algorithms reside [19]. Hence, frameworks that only list the high-level components supporting their functionality usually lack system-level thinking, with no explicit identification of software components, relationships among them, and their properties [5]. Our work presents 18 design patterns regarding agent goal-seeking and plan generation. The pattern catalogue can provide a holistic guidance to practitioners on the trade-off analysis for the design of foundation model-based agents.

⁵<https://www.anthropic.com/claude>

⁶<https://llama.meta.com/>

⁷<https://mistral.ai/>

⁸<https://www.deeplearning.ai/short-courses/ai-agentic-design-patterns-with-autogen/>

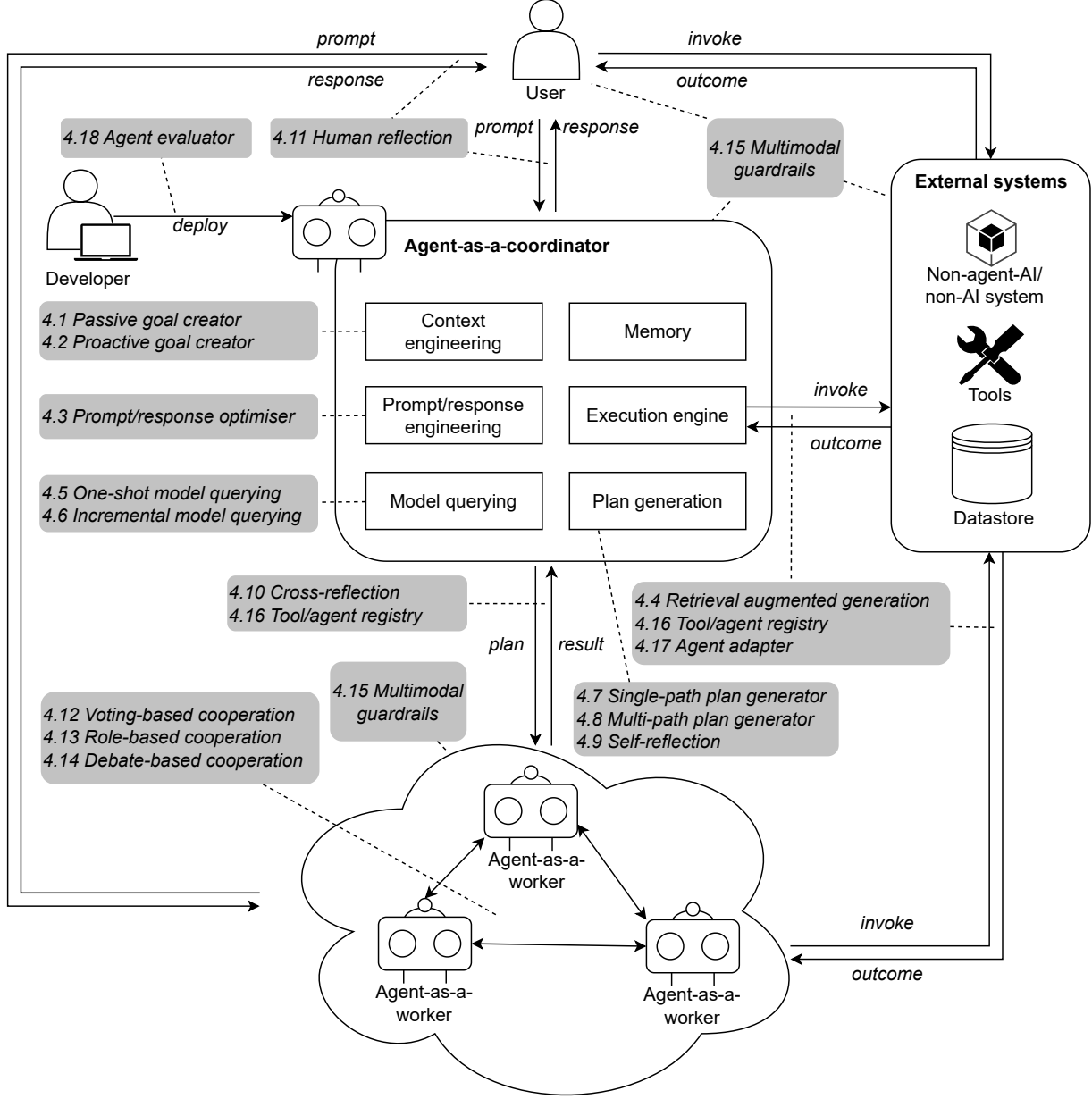


Figure 2: Ecosystem of FM-based agent systems annotated with architectural patterns in gray boxes.

3 Methodology

Fig. 1 illustrates the pattern extraction and collection process. First, we conducted a systematic literature review (SLR) on FM-based agents [5]. We focused on available materials and research works that are highly academic-based. We selected relevant papers based on a series of preset criteria and conducted both forward and backward snowballing processes to identify materials that were left out. After finalising the paper pool, we performed quality assessments on the selected materials to ensure the quality of the work. Finally, 57 studies were included for data extraction and synthesis. A pattern-oriented reference architecture for foundation model-based agents was proposed based on the findings.

Based on the reported findings, we delved into the analysis of identified patterns for building integrating FM-based agents. Through the SLR, we have identified a series of architectural design challenges in the development and implementation of systems with the integration of agents. We then further conducted extensive review on this topic,

Table 1: Foundation model based agent design pattern catalogue overview.

Pattern	Summary
Passive goal creator	Analyse users’ articulated prompts through the dialogue interface to preserve <i>interactivity</i> , <i>goal-seeking</i> and <i>efficiency</i> .
Proactive goal creator	Anticipate users’ goals by understanding human interactions and capturing the context via relevant tools, to enhance <i>interactivity</i> , <i>goal-seeking</i> and <i>accessibility</i> .
Prompt/response optimiser	Optimise the prompts/responses according to the desired input or output content and format to provide <i>standardisation</i> , <i>goal alignment</i> , <i>interoperability</i> and <i>adaptability</i> .
Retrieval augmented generation	Enhance the knowledge <i>updatability</i> of the agents while maintaining <i>data privacy</i> of on-premise foundation model-based agents/systems implementations.
One-shot model querying	Access the foundation model in a single instance to generate all necessary steps for the plan for <i>cost efficiency</i> and <i>simplicity</i> .
Incremental model querying	Access the foundation model at each step of the plan generation process to provide <i>supplementary context</i> , improve <i>reasoning certainty</i> and <i>explainability</i> .
Single-path plan generator	Orchestrate the generation of intermediate steps leading to the achievement of the user’s goal to improve <i>reasoning certainty</i> , <i>coherence</i> and <i>efficiency</i> .
Multi-path plan generator	Allow multiple choice creation at each intermediate step leading to achieving users’ goals to enhance <i>reasoning certainty</i> , <i>coherence</i> , <i>alignment to human preference</i> and <i>inclusiveness</i> .
Self-reflection	Enable the agent to generate feedback on the plan and reasoning process and provide refinement guidance from themselves to improve <i>reasoning certainty</i> , <i>explainability</i> , <i>continuous improvement</i> and <i>efficiency</i> .
Cross-reflection	Use different agents or foundation models to provide feedback and refine the generated plan and reasoning process for better <i>reasoning certainty</i> , <i>explainability</i> , <i>inclusiveness</i> and <i>scalability</i> .
Human reflection	Collect feedback from humans to refine the plan and reasoning process, to effectively align with <i>human preference</i> , improving <i>contestability</i> and <i>effectiveness</i> .
Voting-based cooperation	Enable free opinions expression across agents and reach consensus by submitting their votes to preserve <i>fairness</i> , <i>accountability</i> and <i>collective intelligence</i> .
Role-based cooperation	Assign assorted roles, and finalise decisions in accordance with the roles of agents for to facilitate <i>division of labor</i> , <i>fault tolerance</i> , <i>scalability</i> and <i>accountability</i> .
Debate-based cooperation	Provide and receive feedback across multiple agents adjusts the thoughts and behaviors during the debate with other agents until a consensus is reached to improve <i>adaptability</i> , <i>explainability</i> and <i>critical thinking</i> .
Multimodal guardrails	Control the inputs and outputs of foundation models to meet specific requirements such as user requirements, ethical standards, and laws to enhance <i>robustness</i> , <i>safety</i> , <i>standard alignment</i> , and <i>adaptability</i> .
Tool/agent registry	Maintain a unified and convenient source to select diverse agents and tools to improve <i>discoverability</i> , <i>efficiency</i> , <i>tool appropriateness</i> and <i>scalability</i> .
Agent adapter	Provide interface to connect the agent and external tools for task completion, ensuring <i>interoperability</i> and <i>adaptability</i> , and <i>reduce development cost</i> .
Agent evaluator	Perform testing to assess the agent regarding diverse requirements and metrics, ensuring the <i>functional suitability</i> , <i>adaptability</i> with improved <i>flexibility</i> .

which includes grey literature, and real-world applications (through scrutinising official websites and documents available), for identifying the known uses as the implementation of our included patterns. Combining the findings of both SLR and the additional review, in this paper, we report our findings on 18 extracted patterns.

4 Pattern Catalogue for Foundation Model-based Agents

In this section, we present a pattern catalogue for FM-based agents by adopting the extended pattern template in [20]. It includes the pattern name, a short summary, usage context, a problem statement, a discussion on the forces leading to the problem difficulty, the solution and its consequences, and several examples of real-world known uses of the pattern. Please note that for each included pattern, we provide a simplified graphical representation that highlights only the

essential components necessary to explain pattern application. Detailed interactions between all agent components have been omitted for clarity. Table 1 offers an overview of the collected patterns.

Fig. 2 illustrates the ecosystem of foundation model-based agents, the agent components and interactions between different entities are annotated with the relevant patterns. When users interact with the agent, *passive goal creator* and *proactive goal creator* can help comprehend users’ intentions and environmental information, and formalised the eventual goals in context engineering, while *prompt/response optimiser* refines the prompts or instructions to other agents/tools based on the predefined templates for certain format or content requirements. Given users’ input, the agent fetches additional context information from the knowledge base via *retrieval augmented generation*. Then, it constructs plans to decompose the ultimate goals into actionable tasks through *single-path plan generator* and *multi-path plan generator*. In this process, *one-shot model querying* and *incremental model querying* may be carried out.

A generated plan should be reviewed to ensure its accuracy, usability, completeness, etc. *Self-reflection*, *cross-reflection*, and *human reflection* can help the agent to collect feedback from different reflective entities, and refine the plan and reasoning steps accordingly. Afterwards, the agent can assign tasks to other narrow AI-based or non-AI systems, invoke external tools, and employ a set of agents for goal achievement by *tool/agent registry*. In particular, agents can work on the same task and finalise the results with *voting-based*, *role-based*, or *debate-based cooperation*. For instance, agents can act as different roles such as coordinator and worker. *Agent adapter* keeps learning the interfaces of different tools, and convert them into FM-friendly environment. *Multimodal guardrails* can be applied to manage and control the inputs/outputs of foundation models. Meanwhile, the employed agents will conduct respective reasoning, planning and execution process, which may require external systems via *retrieval augmented generation*, *tool/agent registry*, and *agent adapter* either. Please note that we omit the detailed architecture of agent-as-a-worker, and pattern application in several interactions for the clarity of this diagram, for instance, each agent-as-a-worker has its *passive/proactive goal creator*, *prompt/response optimiser*, and *single/multi-path plan generator*, etc. Finally, we claim that developers can evaluate the performance of agents at both design-time and runtime via *agent evaluator*.

4.1 Passive Goal Creator

Summary: Passive goal creator analyses users’ articulated goals through the dialogue interface.

Context: When querying agents to address certain issues, users provide related context and explain the goals in prompts.

Problem: Users may lack expertise of interacting with agents, and the provided information can be ambiguous for goal achievement.

Forces:

- *Underspecification.* Users may not be able to provide thorough context information and specify precise goals to agents.
- *Efficiency.* Users expect quick responses from agents.

Solution: Fig. 3 illustrates a simple graphical representation of *passive goal creator*. A foundation model-based agent provides a dialogue interface where users can directly specify the context and problems, which are transferred to *passive goal creator* for goal determination. Meanwhile, the *passive goal creator* can also retrieve related information from memory, including the repository of artefacts being worked on, relevant tools used in recent tasks, conversation histories, and the positive and negative examples, which are appended to the user’s prompt for goal-seeking. The generated goals are sent to other components for further task decomposition and completion. In this case, the agent passively receives input from users and generates the strategies to refine and clarify users’ goals, as it only receives the context information directly provided by users. Please note that in multi-agent systems, an agent can send prompts by invoking the API of another agent to assign specific task, while the latter agent analyses the received information and determine the goal.

Consequences:

Benefits:

- *Interactivity.* Users or other agents can interact with an agent via a dialogue interface or related APIs.
- *Goal-seeking.* The agent can analyse user-provided context and retrieve related information from memory, to identify and determine the objectives and create corresponding strategies.
- *Efficiency.* Users can directly send prompts to the agent through the dialogue interface, which is intuitive and easy to use.

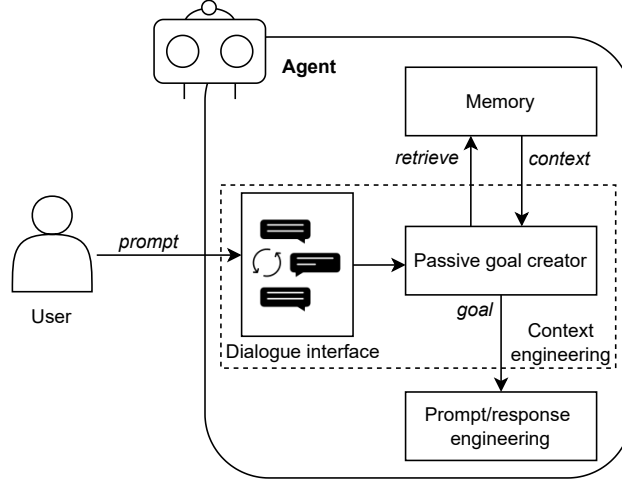


Figure 3: Passive goal creator.

Drawbacks:

- *Reasoning uncertainty.* Users may have assorted backgrounds and experiences. Unclear or ambiguous context information may intensify the reasoning uncertainties, especially considering there are no standard prompt requirements.

Known uses:

- Liu et al. [21] designed an agent that can communicate with users and help refine research questions via a dialogue interface.
- Kannan et al. [22] proposed an agent for users to decompose and allocate tasks to robots through a dialogue interface.
- *HuggingGPT*⁹. HuggingGPT can generate responses to address user requests via a chatbot. Users' requests including complex intents can be interpreted as their intended goals.

Related patterns:

- *Proactive goal creator.* *Proactive goal creator* can be regarded an alternative of *passive goal creator* enabling multimodal context injection.
- *Prompt/response optimiser.*
- *Passive goal creator* can first handle users' inputs and transfer the goals and relevant context information to *prompt/response optimiser* for prompt refinement.

4.2 Proactive Goal Creator

Summary: Proactive goal creator anticipates users' goals by understanding human interactions and capturing the context via relevant tools.

Context: Users explain the goals that the agent is expected to achieve in the prompt.

Problem: The context information collected via solely a dialogue interface may be limited, and result in inaccurate responses to users' goals.

Forces:

⁹<https://huggingface.co/spaces/microsoft/HuggingGPT>

- *Underspecification.* i) Users may not be able to provide thorough context information and specify precise goals to agents. ii) Agents can only retrieve limited information from the memory.
- *Accessibility.* Users with specified disabilities may not be able to directly interoperate with the agent via *passive goal creator*.

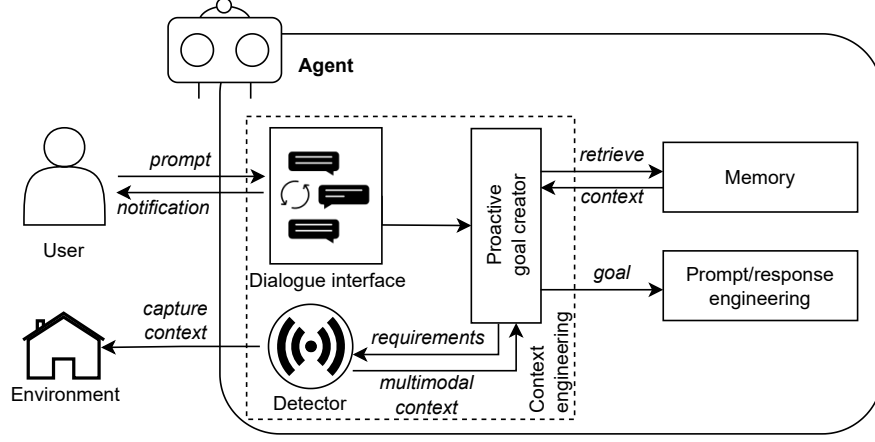


Figure 4: Proactive goal creator.

Solution: Fig. 4 illustrates a simple graphical representation of *proactive goal creator*. In addition to the prompts received from dialogue interface, and relevant context retrieved from memory, the *proactive goal creator* can anticipate users’ goals by sending requirements to detectors, which will then capture and return the user’s surroundings with multimodal context information for further analysis and comprehension to generate the goals, for instance, identifying the user’s gestures through cameras, recognising application UI layout via screenshots, etc. Please note the *proactive goal creator* should notify users about context capturing and other relevant issues with a low false positive rate, to avoid unnecessary interruptions. In addition, the captured environment information can be stored in the agent’s memory (or knowledge base) to establish “world models” [23, 24] to continuously improve its ability to comprehend the real world.

Consequences:

Benefits:

- *Interactivity.* An agent can interact with users or other agents by anticipating their decisions proactively with captured multimodal context information.
- *Goal-seeking.* The multimodal input can provide more detailed information for the agent to understand users’ goals, and increase the accuracy and completeness of goal achievement.
- *Accessibility.* Additional tools can help capture the sentiments and other context information from disabled users, ensuring accessibility and broadening the human values of foundation model-based agents.

Drawbacks:

- *Overhead.* i) *Proactive goal creator* is enabled by the multimodal context information captured by relevant tools, which may increase the cost of the agent. ii) Limited context information may increase the communication overhead between users and agents.

Known uses:

- *GestureGPT* [25]. *GestureGPT* can decipher users’ hand gesture descriptions and hence comprehend users’ intents.
- Zhao et al. [26] proposed a programming screencast analysis tool that can extract the coding steps and code snippets.
- *ProAgent* [27]. *ProAgent* can observe the behaviours of other teammate agents, deduce their intentions, and adjust the planning accordingly.

Related patterns:

- *Passive goal creator.* *Proactive goal creator* can be regarded an alternative of *passive goal creator* enabling multimodal context injection.
- *Prompt/response optimiser.* *Proactive goal creator* can first handle users' inputs and transfer the goals and relevant context information to *prompt/response optimiser* for prompt refinement.
- *Multimodal guardrails.* Multimodal guardrails can help process the multimodal data captured by proactive goal creator.

4.3 Prompt/Response Optimiser

Summary: Prompt/response optimiser refines the prompts/responses according to the desired input or output content and format.

Context: Users may struggle with writing effective prompts, especially considering the injection of comprehensive context. Similarly, it may be difficult for users to understand the agent's outputs in certain cases.

Problem: How to generate effective prompts and standardised responses that are aligned with users' goals or objectives?

Forces:

- *Standardisation.* Prompts and responses may vary in structure, format, and content, which will lead to potential confusion or inconsistent behaviours of the agent.
- *Goal alignment.* Ensuring that prompts and responses are aligned with the ultimate goal or objective can facilitate the agent to achieve desired results.
- *Interoperability.* The generated prompts and responses may be directly input to other components, external tools or agents for completing further tasks.

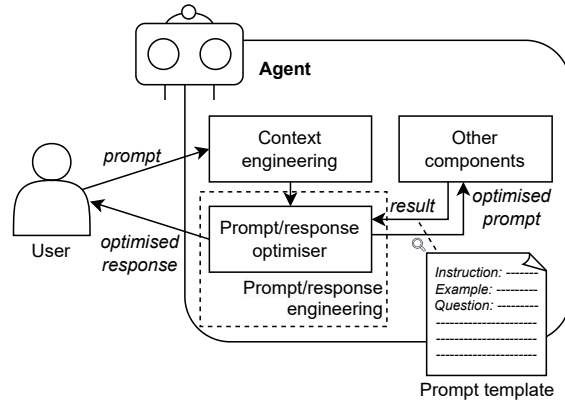


Figure 5: Prompt/response optimiser.

Solution: Fig. 5 illustrates a high-level graphical representation of *prompt/response optimiser*. A user may input initial prompts to the agent, however, such prompts may be ineffective due to the lack of relevant context, unintentional injection attacks, redundancy, etc. In this regard, *prompt/response optimiser* can construct refined prompts and responses adhering to predefined constraints and specifications. These constraints and specifications outline the desired content and format for the inputs and outputs, ensuring alignment with the ultimate goal. A prompt or response template is often used in the *prompt/response optimiser* as a factory for creating specific instances of prompts or responses [28, 29]. This template offers a structured approach to standardise the queries and responses, improving the accuracy of the responses and facilitate their interoperations with external tools or agents. For instance, a prompt template can contain the instructions to an agent, some examples for few-shot learning, and the question/goal for the agent to work.

Consequences:

Benefits:

- *Standardisation.* *Prompt/response optimiser* can create standardised prompts and responses regarding the requirements specified in the template.
- *Goal alignment.* The optimised prompts and responses adhere to user-defined conditions, hence they can achieve higher accuracy and relevance to the goals.
- *Interoperability.* Interoperability between agent and external tools is facilitated by *prompt/response optimiser*, which can provide consistent and well-defined prompts and responses for task execution.
- *Adaptability.* *Prompt/response optimiser* can accommodate different constraints, specifications, or domain-specific requirements by refining the template with a knowledge base.

Drawbacks:

- *Underspecification.* In certain cases, it may be difficult for *prompt/response optimiser* to capture and incorporate all relevant contextual information effectively, especially considering the ambiguity of users' input, and dependency on context engineering. Consequently, the optimiser may struggle to generate appropriate prompts or responses.
- *Maintenance overhead.* Updating and maintaining prompt or response templates may cause significant overhead. Changes in requirements may necessitate modifying multiple templates, which is time-consuming and error-prone.

Known uses:

- *LangChain*¹⁰. LangChain provides prompt templates for practitioners to develop custom foundation model-based agents.
- *Amazon Bedrock*¹¹. Users can configure prompt templates in Amazon Bedrock, defining how the agent should evaluate and use the prompts.
- *Dialogflow*¹². Dialogflow allows users to create generators to specify agent behaviours and responses at runtime.

Related patterns:

- *Passive goal creator* and *proactive goal creator* can first handle users' inputs and transfer the goals and relevant context information to *prompt/response optimiser* for prompt refinement.
- *Self-reflection*, *cross-reflection*, and *human-reflection*. The reflection patterns can be applied to assess and refine the output of *prompt/response optimiser*.
- *Agent adapter*. *Prompt/response optimiser* can improve users' inputs, and the optimised prompts can be sent to other agents for goal achievement, while *agent adapter* focuses more on the utilisation of external tools.

4.4 Retrieval Augmented Generation (RAG)

Summary: Retrieval augmented generation techniques enhance the knowledge updatability of agents for goal achievement, and maintain data privacy of on-premise foundation model-based agents/systems implementations.

Context: Large foundational model-based agents are not equipped with knowledge related to explicitly specific domains, especially on highly confidential and privacy-sensitive local data, unless they are fine-tuned for pre-trained using domain data.

Problem: Given a task, how can agents conduct reasoning with data/knowledge that are not learned by the foundation models through model training?

Forces:

¹⁰https://api.python.langchain.com/en/latest/prompts/langchain_core.prompts.prompt.PromptTemplate.html

¹¹<https://docs.aws.amazon.com/bedrock/latest/userguide/advanced-prompts-configure.html>

¹²<https://cloud.google.com/dialogflow>

- *Lack of knowledge.* The reasoning process may be unreliable when the agent is required to accomplish domain-specific tasks that the agent has no such knowledge reserve.
- *Overhead.* Fine-tuning large foundation model using local data or training a large foundation model locally consumes high amount of computation and resource costs.
- *Data Privacy.* Local data are confidential to be used to train or fine-tune the models.

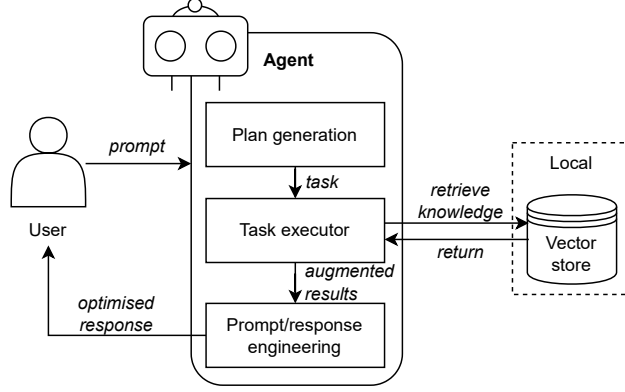


Figure 6: Retrieval augmented generation.

Solution: Fig. 6 illustrates a high-level graphical representation of *retrieval augmented generation*. RAG is a technique for enhancing the accuracy and reliability of agents with facts retrieved from other sources (internal or online data). The knowledge gaps where the agents are lacking in memory are filled using the parameterized knowledge generated in vector databases. For instance, after plan generation, specific tasks may require information that is not within the original agent memory. The agent can hence retrieve information from the parameterised knowledge and use for task completion, while the augmented responses will be returned back to the user after optimisation. In particular, the implementation of RAG consists of the following steps: i) Determine the data sources. ii) Define the data structure for indexing raw data (i.e., text, image, video and audio) into embeddings or knowledge graphs. iii) Given a certain query, task executor encodes this query and searches the knowledge base (e.g., vector store) to retrieve the most relevant information. iv) Task executor processes the obtained data through reranking and filtering to produce a more informed and accurate response.

The retrieval process requires zero pretraining or fine-tuning of the model served by the agent which preserves the data privacy of local data, reduces training and computation costs, and also provides up-to-date and more precise information required. The retrieved local data can be sent back to the agent via prompts (need to consider the context window size), whereas the agent is able to process the information and generate plans via in-context learning. Currently there is a cluster of RAG techniques focusing on various enhancement aspects, data sources and applications [30], for instance, federated RAG [31], graph RAG [32], etc. Further, Retrieval Interleaved Generation can be considered a related technique of RAG where the agent can dynamically access external knowledge throughout the response generation phase.

Consequences:

Benefits:

- *Knowledge retrieval.* Agents can search and retrieve knowledge related to the given tasks, which ensures the reliability of reasoning steps.
- *Updatability.* The prompts/responses generated using RAG by the agent on internal or online data are updatable by the complimentary parameterized knowledge.
- *Data privacy.* The agent can retrieve additional knowledge from local datastores, which ensures data privacy and security.
- *Cost-efficiency.* Under the data privacy constraint, RAG can provide essential knowledge to the agent without training a new foundation model entirely. This reduced the training costs.

Drawbacks:

- *Maintenance overhead.* Maintenance and update of the parameterized knowledge in the vector store requires additional computation and storage costs.
- *Data limitation.* The agents still mainly rely on the data it has been trained on to generate prompts. This can impact the quality and accuracy of the generated content in those specific domains.

Known uses:

- *LinkedIn¹³.* LinkedIn applies RAG to construct the pipeline of foundation model based agents, which can search appropriate case studies to respond users.
- Yan et al. [33] devise a retrieval evaluator which can output a confidence degree after assessing the quality of retrieved data. The solution can improve the robustness and overall performance of RAG for agents.
- Levonian et al. [34] apply RAG with GPT-3.5, developing an agent that can retrieve the contents of a high-quality open-source math textbook to generate responses to students.

Related patterns: *Retrieval augmented generation* can complement all other patterns by providing extra context information from the local datastore.

4.5 One-Shot Model Querying

Summary: The foundation model is accessed in a single instance to generate all necessary steps for the plan.

Context: When users interact with the agent for specific goals, the included foundation model is queried for plan generation.

Problem: How can the agent generate the steps for a plan efficiently?

Forces:

- *Efficiency.* For certain pressing tasks, the agent should be able to conduct planning and respond in a short amount of time.
- *Overhead.* Users need to pay for each interaction with commercial foundation models.

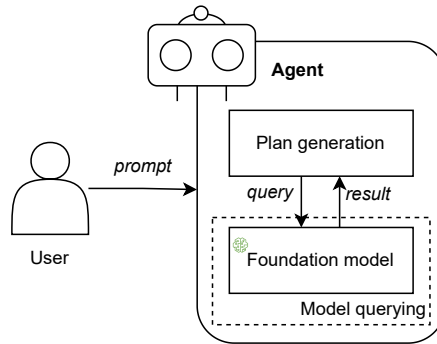


Figure 7: One-shot model querying.

Solution: Fig. 7 illustrates interactions between user and agent within *one-shot model query*. In this scenario, the agent queries the incorporated foundation model to generate a corresponding plan based on user-specified goals and constraints. The foundation model is only queried for once in regard to the user’s requirements (e.g. limited budget), to comprehend the provided inputs. In this manner, the agent can devise a multi-step plan to achieve a broad goal, and provide a holistic explanation for this plan without delving into detailed reasoning steps. Please note that this pattern is applicable when other components query the integrated foundation model.

Consequences:

¹³<https://www.linkedin.com/blog/engineering/generative-ai/musings-on-building-a-generative-ai-product>

Benefits:

- *Efficiency.* The agent can generate a plan to achieve users’ goals by querying the underlying foundation model only once, which saves consumed time.
- *Cost-efficiency.* Users’ expenses can be reduced since the foundation model is queried for one time.
- *Simplicity.* *One-shot model querying* can satisfy the tasks that do not require complex action plans.

Drawbacks:

- *Oversimplification.* For complex tasks, *one-shot model querying* may not be able to fully capture all requirements at one time, hence oversimplifying the tasks and cannot return a correct response.
- *Lack of explainability.* *One-shot model querying* may suffer the lack of explainability as the incorporated foundation model is queried only once, which may not provide detailed reasoning steps for plan generation.
- *Size of the context window.* The response quality may be constrained considering the foundation models’ current capability of handling long conversational contexts and the token limits.

Known uses: *One-shot model querying* can be considered configuration or use by default when a user is leveraging a foundation model, while CoT and Zero-shot-CoT both exemplify this pattern [35,36].

Related patterns:

- *Incremental model querying.* *Incremental model querying* can be regarded an alternative of *one-shot model querying* with iteration.
- *Single-path plan generator.* *One-shot model querying* enables the generation of single-path plans by only querying the foundation model for one time.
- *Multimodal guardrails.* Multimodal guardrails serve as an intermediate layer, managing the inputs and outputs of model querying.

4.6 Incremental Model Querying

Summary: Incremental model querying involves accessing the foundation model at each step of the plan generation process.

Context: When users interact with the agent for specific goals, the included foundation model is queried for plan generation.

Problem: The foundation model may struggle to generate the correct response at the first attempt. How can the agent conduct an accurate reasoning process?

Forces:

- *Size of the context window.* The context window of a foundation model may be limited, hence users may not be able to provide a complete and comprehensive prompt.
- *Oversimplification.* The reasoning process may be oversimplified and hence endure uncertainties with only one attempt of model querying.
- *Lack of explainability.* The generated responses of foundation models require detailed reasoning process to preserve explainability and eventual trustworthiness.

Solution: Fig. 8 illustrates interactions between the plan generation component and integrated foundation model with *incremental model querying*. The agent could engage in a step-by-step reasoning process to develop the plan for goal achievement with multiple queries to the foundation model. Meanwhile, human feedback can be provided at any time to both the reasoning process and generated plan, and adjustments can be made accordingly during model querying. The number of queries can be predefined in agent configuration or determined in user prompts. Please note that *incremental model querying* can rely on a reusable template, which guides the process through context injection or an explicit workflow/plan repository and management system. This pattern is applicable when other components query the integrated foundation model.

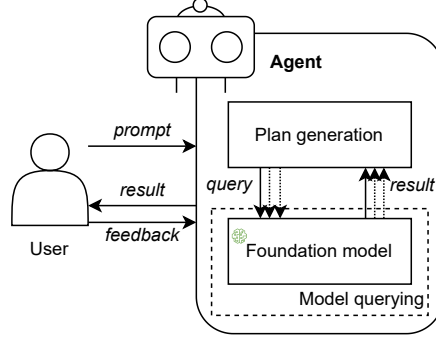


Figure 8: Incremental model querying.

Consequences:**Benefits:**

- *Supplementary context.* *Incremental model querying* allows users to split the context in multiple prompts to address the issue of limited context window.
- *Reasoning certainty.* Foundation models will iteratively refine the reasoning steps by self-checking or feedback from users.
- *Explainability.* Users can query the foundation model to provide detailed reasoning steps through *incremental model querying*.

Drawbacks:

- *Overhead.* i) *Incremental model querying* requires multiple interactions with the foundation model, which may increase the time consumption for planning determination. ii) The high volume of user queries may be cost-intensive when utilising commercial foundation models.

Known uses:

- *HuggingGPT⁹.* The underlying foundation model of HuggingGPT is queried multiple times to decompose users' requests into fine-grained tasks, and then determine the dependencies and execution orders of tasks [37].
- *EcoAssistant* [38]. EcoAssistant applies a code executor interacting with the foundation model to iteratively refine code.
- *ReWOO* [39]. ReWOO queries the foundation model to i) generate a list of interdependent plans, and; ii) combine the observation evidence fetched from tools with the corresponding task.

Related patterns:

- *One-shot model querying.* *Incremental model querying* can be regarded an alternative of *one-shot model querying* with iteration.
- *Multi-path plan generator.* The agent can capture users' preferences at each step and generate multi-path plans by iteratively querying the foundation model.
- *Self-reflection.* *Self-reflection* requires agents to query their incorporated foundation model multiple times for response review and evaluation.
- *Human-reflection.* *Human-reflection* is enabled by *incremental model querying* for iterative communication between users/experts and the agent.
- *Multimodal guardrails.* Multimodal guardrails serve as an intermediate layer, managing the inputs and outputs of model querying.

4.7 Single-Path Plan Generator

Summary: Single-path plan generator orchestrates the generation of intermediate steps leading to the achievement of the user's goal.

Context: A agent is considered “black box” to users, while users may care about the process of how an agent achieve users’ goals.

Problem: How can an agent efficiently formulate the strategies to achieve users’ goals?

Forces:

- *Underspecification.* Users may assign tasks with high-level abstraction, which may be challenging for agents to handle the uncertainty or ambiguity in the provided context.
- *Coherence.* Users and other interacting tools/agents will expect coherent responses or guidelines for achieving certain goals.
- *Efficiency.* Uncertain decisions may affect the efficiency of an agent, which will result in reduced user satisfaction.

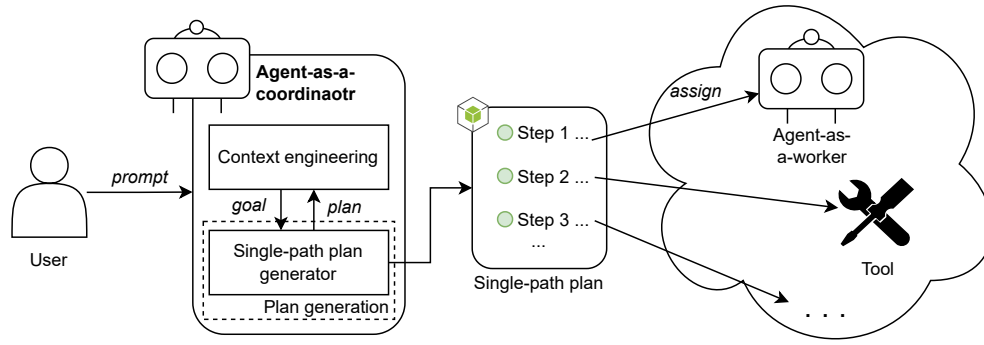


Figure 9: Single-path plan generator.

Solution: Fig. 9 illustrates a high-level graphical representation of *single-path plan generator*. After receiving and comprehending users’ goals, the *single-path plan generator* can coordinate the creation of a plan for other agents or tools and prioritise the tasks, to progressively lead towards goal accomplishment. Specifically, the plan generation process requires inference and reasoning that whether intermediate steps are actionable and optimal. Each step in this process is designed to have only a single subsequent step, forming a linear and direct plan, such as Chain-of-Thought (CoT) [40]. Self-consistency is employed to confirm with the foundation model several times and select the most consistent answer as the final decision [41]. Please note that the generated plan may have different granularity based on the given goal that complex plans may incorporate multiple workflows, processes, tasks and fine-grained steps.

Consequences:

Benefits:

- *Reasoning certainty.* *Single-path plan generator* generates a multi-step plan, which can reflect the reasoning process and mitigate the uncertainty or ambiguity for achieving users’ goals.
- *Coherence.* The interacting users, agents and tools are provided a clear and coherent path towards the ultimate goals.
- *Efficiency.* *Single-path plan generator* can increase efficiency in agents via pruning unnecessary steps or distractions.

Drawbacks:

- *Flexibility.* A single-path plan may result in limited flexibility to accommodate diverse user preferences or application scenarios, hence users cannot customise their solutions.
- *Oversimplification.* The agent may oversimplify the generated plan which requires multi-faceted approaches.

Known uses:

- *LlamaIndex*¹⁴. LlamaIndex fine-tunes a ReAct Agent to achieve better performance with *single-path plan generator* via CoT.
- *ThinkGPT*¹⁵. ThinkGPT provides a toolkit to facilitate the implementation of *single-path plan generator* pattern.
- Zhang et al. [42] promote the implementation by elucidating the basic mechanisms and paradigm shift of CoT.

Related patterns:

- *One-shot model querying*. *One-shot model querying* enables the generation of single-path plans by only querying the foundation model for one time.
- *Multi-path plan generator*. *Multi-path plan generator* can be regarded an alternative of *single-path plan generator* for customised strategy.
- *Self-reflection*. *Single-path plan generator* and *self-reflection* both contribute to self-Consistency with Chain of Thought.

4.8 Multi-Path Plan Generator

Summary: Multi-path plan generator allows for creating multiple choices at each intermediate step leading to achieving users' goals.

Context: A agent is considered “black box” to users, while users may care about the process of how an agent achieve users' goals.

Problem: How can an agent generate a high-quality, coherent, and efficient solution considering inclusiveness and diversity when presented with a complex task or problem?

Forces:

- *Underspecification*. Users may assign tasks with high-level abstraction, which may be challenging for agents to handle the uncertainty or ambiguity in the provided context.
- *Coherence*. Users and other interacting tools/agents will expect coherent responses or guidelines for achieving certain goals.
- *Alignment to human preference*. Certain goals require agents to capture users' preferences, to provide customised solutions.
- *Oversimplification*. For particular complex tasks, agents may oversimplify the reasoning process, hence the provided solutions cannot satisfy users' requirements.

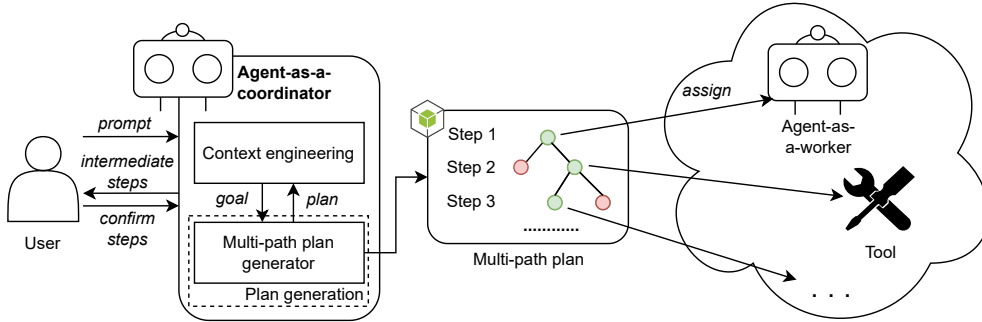


Figure 10: Multi-path plan generator.

¹⁴https://docs.llamaindex.ai/en/stable/examples/finetuning/react_agent/react_agent_finetune/

¹⁵<https://github.com/jina-ai/thinkgpt>

Solution: Fig. 10 illustrates a high-level graphical representation of *multi-path plan generator*. Based on *single-path plan generator*, *multi-path plan generator* can create multiple choices at each step towards the achievement of goals, which requires the underlying foundation model to tease out the eligible and actionable activities for each choice in the previous step. Specifically, users’ preferences may influence the subsequent intermediate steps, leading to different eventual plans. The employment of involved agents and tools will be adjusted accordingly. Tree-of-Thoughts [43] exemplifies this design pattern.

Consequences:

Benefits:

- *Reasoning certainty.* *Multi-path plan generator* can generate a plan with multiple choices of intermediate steps to resolve the uncertainty or ambiguity within reasoning process.
- *Coherence.* The interacting users, agents and tools are provided a clear and coherent path towards the ultimate goals.
- *Alignment to human preference.* Users can confirm each intermediate step to finalise the planning, hence human preferences are absorbed in the generated customised strategy.
- *Inclusiveness.* The agent can specify multiple directions in the reasoning process for complex tasks.

Drawbacks:

- *Overhead.* Task decomposition and multi-plan generation may increase the communication overhead between the user and agent.

Known uses:

- *AutoGPT*¹. AutoGPT can make informed decisions by incorporating Tree-of-Thoughts as the *multi-path plan generator*.
- *Gemini*¹⁶. For a task, Gemini can generate multiple choices for users to decide. Upon receiving users’ responses, Gemini will provide multiple choices for the next step.
- *Open AI*¹⁷. GPT-4 was leveraged to implement a *multi-path plan generator* based on Tree-of-Thoughts.

Related patterns:

- *Incremental model querying.* The agent can capture users’ preferences at each step and generate multi-path plans by iteratively querying the foundation model.
- *Single-path plan generator.* *Multi-path plan generator* can be regarded an alternative of *single-path plan generator* for customised strategy.
- *Human-reflection.* *Multi-plan generator* creates plans with various directions, and *human-reflection* can help finalise the plan with user feedback to determine the choice of each intermediate step.

4.9 Self-Reflection

Summary: Self-reflection enables the agent to generate feedback on the plan and reasoning process and provide refinement guidance from themselves.

Context: Given users’ goals and requirements, the agent will generate a plan to decompose the goals into a set of tasks for achieving the goals.

Problem: A generated plan may be affected by hallucinations of the foundation model, how to review the plan and reasoning steps and incorporate feedback efficiently?

Forces:

¹⁶https://www.youtube.com/watch?v=v5tRc_5-8G4

¹⁷<https://github.com/princeton-nlp/tree-of-thought-11m>

- *Reasoning uncertainty.* There may be inconsistencies or uncertainties embedded in the agent’s reasoning process, affecting the task success rate and response accuracy.
- *Lack of explainability.* The trustworthiness of the agent can be disturbed by the issue of transparency and explainability of how the plan is generated.
- *Efficiency.* Certain goals require the plan to be finalised within a specific time period.

Solution: Fig. 11 depicts a high-level graphical representation of *self-reflection*. In particular, reflection is an optimisation process formalised to iteratively review and refine the reasoning process and generated contents of the agent. The user prompts specific goals to the agent, which then generates a plan to accomplish users’ requirements. Subsequently, the user can instruct the agent to reflect on the plan and the corresponding reasoning process. The agent will backtrack the inference process to verify whether certain intermediate results are incorrect and hence misleading all subsequent steps, then adjust and align its reasoning process to create a refined plan accordingly. Such reflection processes and results can be saved in the agent’s memory for continuous learning. The finalised plan will be carried out step by step. Self-consistency [44] exemplifies this pattern.

Consequences:

Benefits:

- *Reasoning certainty.* Agents can evaluate their own responses and reasoning procedure to check whether there are any errors or inappropriate outputs, and make refinement accordingly.
- *Explainability.* *Self-reflection* allows the agent to review and explain its reasoning process to users, facilitating better comprehension of the agent’s decision-making process.
- *Continuous improvement.* The agent can continuously update the memory or knowledge base and the manner of formalising the prompts and knowledge, to provide more reliable and coherent output to users without or with fewer reflection steps.
- *Efficiency.* On one hand, it is time-saving for the agent to self-evaluate its response, as no additional communication overhead is cost compared to other reflection patterns. On the other hand, the agent can provide more accurate responses in the future to reduce the overall reasoning time consumption considering the continuous improvement.

Drawbacks:

- *Reasoning uncertainty.* The evaluation result is dependent on the complexity of *self-reflection* and the agent’s competence in assessing its generated responses.
- *Overhead.* i) *Self-reflection* can increase the complexity of an agent, which may affect the overall performance. ii) Refining and maintaining agents with self-reflection capabilities requires specialised expertise and development process.

Known uses:

- *Reflexion* [45]. Reflexion employs a *self-reflection* model which can generate nuanced and concrete feedback based on the success status, current trajectory, and persistent memory.
- *Bidder agent* [46]. A replanning module in this agent utilises *self-reflection* to create new textual plans based on the auction’s status and new context information.
- *Generative agents* [47]. Agents perform reflection two or three times a day, by first determining the objective of reflection according to the recent activities, then generating a reflection which will be stored in the memory stream.

Related patterns:

- *Prompt/response optimiser.* *Self-reflection* can be applied to assess and refine the output of *prompt/response optimiser*.
- *Incremental model query.* *Self-reflection* requires agents to query their incorporated foundation model multiple times for response review and evaluation.
- *Single-path plan generator.* *Single-path plan generator* and *self-reflection* both contribute to self-Consistency with Chain of Thought.

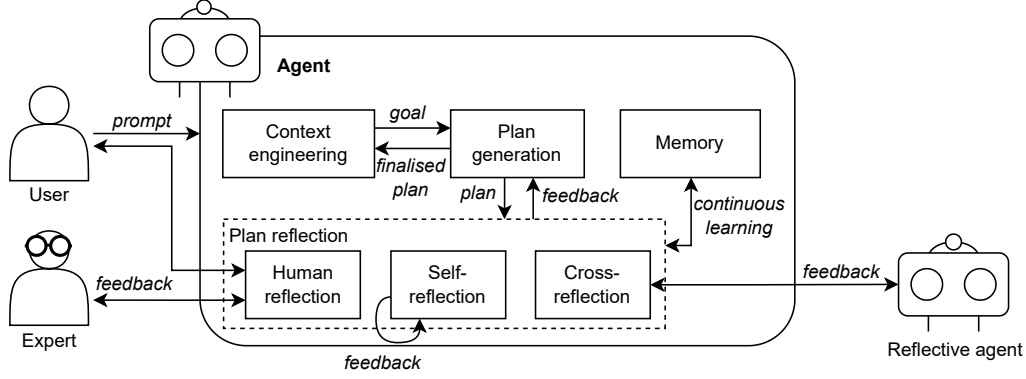


Figure 11: Plan reflection pattern.

4.10 Cross-Reflection

Summary: Cross-reflection uses different agents or foundation models to provide feedback and refine the generated plan and corresponding reasoning procedure.

Context: The agent generates a plan to achieve users' goals, while the quality of this devised plan should be assessed.

Problem: When an agent has limited capability and cannot conduct reflection with satisfying performance, how to evaluate the output and reasoning steps of this agent?

Forces:

- *Reasoning uncertainty.* The inconsistencies and errors in the agent's reasoning process may reduce response accuracy and affect the overall trustworthiness.
- *Lack of explainability.* The trustworthiness of the agent can be disturbed by the issue of transparency and explainability of how the plan is generated.
- *Limited capability.* An agent may not be able to perform reflection well due to its limited capability and the complexity of self-reflection.

Solution: Fig. 11 includes a high-level graphical representation of *cross-reflection*. If an agent cannot generate accurate results or precise planning steps via reflecting its outputs, users can prompt the agent to query another agent which is specialised in reflection. The latter agent can review and evaluate the logged outputs and relevant reasoning steps of the original agent, and provide refinement suggestions. This process can be iterative until the reflective agent confirms the plan. In addition, multiple agents can be queried for reflection to generate comprehensive responses.

Consequences:

Benefits:

- *Reasoning certainty.* The agent's outputs and respective methodology are assessed and refined by other agents to ensure the reasoning certainty and response accuracy.
- *Explainability.* Multiple agents can be employed to review the reasoning process of the original agent, providing thorough explanations to the user.
- *Inclusiveness.* The reflective feedback includes different reasoning outputs when multiple agents are queried, which can help formalise a comprehensive refinement suggestion.
- *Scalability.* *Cross-reflection* supports scalable agent-based systems as the reflective agents can be flexibly updated without disrupting the system operation.

Drawbacks:

- *Reasoning uncertainty.* The overall response quality and reliability are dependent on the performance of other reflective agents.

- *Fairness preservation.* When various agents participate in the reflection process, a critical issue would be how to preserve fairness among all the provided feedback.
- *Complex accountability.* If the cross-reflection feedback causes serious or harmful results, the accountability process may be complex when multiple agents are employed.
- *Overhead.* i) There will be communication overhead for the interactions between agents. ii) Users may need to pay for utilising the reflective agents.

Known uses:

- *XAgent*¹⁸. In XAgent, the tool agent can send feedback and reflection to the plan agent to indicate whether a task is completed, or pinpoint the refinements.
- Yao et al. [48] explore agents' capability of learning through communicating with each other. A thinker agent can provide suggestions to an actor agent, who is responsible for decision-making.
- Qian et al. [49] develop a virtual software development company based on agents, where the tester agents can detect bugs and report to programmer agents.
- Talebirad and Nadiri [50] analyse the inter-agent feedback which involves criticism of each other, which can help agents adapt their strategies.

Related patterns:

- *Prompt/response optimiser.* *Cross-reflection* can provide feedback to improve the output of *prompt/response optimiser*.
- *Voting-based, role-based, and debate-based cooperation.* Reflective agents can collaborate to evaluate an agent's outputs in different cooperation schemes.
- *Tool/agent registry.* The agent can search reflective agents for *cross-reflection* via *tool/agent registry*.

4.11 Human Reflection

Summary: The agent collects feedback from humans to refine the plan, to effectively align with the human preference.

Context: Agents create plans and strategies that decompose users' goals and requirements into a pool of tasks. The tasks will be completed by other tools and agents.

Problem: How to ensure human preference is fully and correctly captured and integrated into the reasoning process and generated plans?

Forces:

- *Alignment to human preference.* Agents are expected to achieve users' goals ultimately, consequently, it is critical for agents to comprehend users' preferences.
- *Contestability.* If the agent's outputs do not satisfy users' requirements and will cause negative impacts, there should be a timely process for users to contest the responses of agent.

Solution: Fig. 11 presents a high-level graphical representation of *human-reflection*. When a user prompts his/her goals and specified constraints, the agent first creates a plan consisting of a series of intermediate steps. The constructed plan and its reasoning process logs can be presented to the user for review, or sent to other human experts to validate the feasibility and usefulness. The user or expert can provide comments or suggestions to indicate which steps can be updated or replaced. The plan will be iteratively assessed and improved until it is approved by the user/expert.

Consequences:

Benefits:

¹⁸<https://github.com/OpenBMB/XAgent>

- *Alignment to human preference.* The agent can directly receive feedback from users or additional human experts to understand human preferences, and improve the outcomes or procedural fairness, diversity in the results, etc.
- *Contestability.* Users or human experts can challenge the agent’s outcomes immediately if abnormal behaviours or responses are found.
- *Effectiveness.* *Human-reflection* allows agents to include users’ perspectives for plan refinement, which can help formalise responses tailored to users’ specific needs and level of understanding. This can ensure the usability of strategies, and improve the effectiveness for achieving users’ goals.

Drawbacks:

- *Fairness preservation.* The agent may be affected by users who provide skewed information about the real world.
- *Limited capability.* Agents may still have limited capability to understand human emotions and experiences.
- *Underspecification.* Users may provide limited or ambiguous reflective feedback to agents.
- *Overhead.* Users may need to pay for the multiple rounds of communication with the agent.

Known uses:

- *Inner Monologue* [51]. Inner Monologue is implemented in a robotic system, which can decompose users’ instructions into actionable steps, and leverage human feedback for object recognition.
- Ma et al. [52] explore the deliberation between users and agents for decision-making. Users and agents both need to provide related evidence and arguments for their conflicting opinions.
- Wang et al. [53] incorporate human feedback for agents to capture the dynamic evolution of user interests and consequently provide more accurate recommendations.

Related patterns:

- *Prompt/response optimiser.* *Human-reflection* can provide human preference and suggestions to improve the generated prompts and responses.
- *Multi-path plan generator.* *Multi-plan generator* creates plans with various directions, and *human-reflection* can help finalise the plan with user feedback to determine the choice of each intermediate step.
- *Incremental model querying.* *Human-reflection* is enabled by *incremental model querying* for iterative communication between users/experts and the agent.

4.12 Voting-based Cooperation

Summary: Agents can freely provide their opinions and reach consensus through voting-based cooperation.

Context: Multiple agents can be leveraged within a compound AI system. Agents need to collaborate on the same task while having their own perspectives.

Problem: How to finalise the agents’ decisions properly to ensure fairness among different agents?

Forces:

- *Diversity.* The employed agents can have diverse opinions of how a plan is constructed or how a task should be completed.
- *Fairness.* Decision-making among agents should take their rights and responsibilities into consideration to preserve fairness.
- *Accountability.* The behaviours of agents should be recorded to enable future auditing if any violation is found in the collaboration outcomes.

Solution: Fig. 12 illustrates how agents can cooperate to finalise a decision via votes. Specifically, an agent can first generate a candidate response to the user’s prompts, then it holds a vote in which different reflective suggestions are

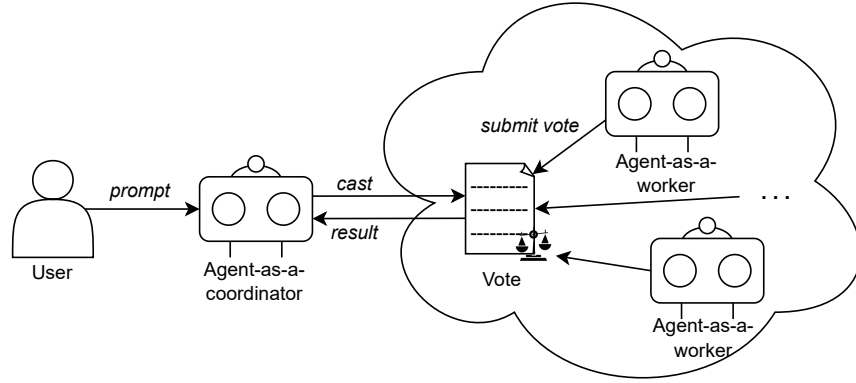


Figure 12: Voting-based cooperation.

presented as choices. Additional agents are requested to submit their votes to select the most appropriate feedback according to their capabilities and experiences. In this circumstance, agents communicate in a centralised manner that the original agent will act as a coordinator. The voting result will be formalised and sent back to the original agent, who can refine the response accordingly before answering the user. Please note that the voting process can be implemented through various methods, e.g., direct communications between the agent-as-a-coordinator and other agents, blockchain-based smart contracts, etc. Moreover, the identity management of all participating agents is significant to ensure the traceability and verifiability of votes.

Consequences:

Benefits:

- *Fairness.* Votes can be held in multiple ways to preserve fairness. For instance, counting heads to ensure agents' rights are equal, or weights can be distributed considering the roles of agents, etc.
- *Accountability.* The overall procedure and final results are recorded in the respective voting system. Stakeholders can trace back to identify the accountable agents selecting certain options.
- *Collective intelligence.* The finalised decisions after votes can leverage the strengths of multiple agents (e.g. comprehensive knowledge base), hence they are regarded as more accurate and reliable than the ones generated by a single agent.

Drawbacks:

- *Centralisation.* Specific agents may gain the majority of decision rights and hence have the ability to compromise the voting process.
- *Overhead.* Hosting a vote may increase the communication overhead for agents to examine and vote for the choices.

Known uses:

- Hamilton [54] utilises nine agents to simulate court where the agents need to vote for the received cases. Each case is determined by the dominant voting result.
- ChatEval [55]. Agents can reach consensus on users' prompts via voting, while the voting results can be totalled by calculating either the majority vote or the average score.
- Yang et al. [56] explore the alignment of agent voters based on GPT-4 and LLaMA-2 and human voters on 24 urban projects. The results indicate that agent voters tend to have uniform choices while human voters have diverse preferences.
- Li et al. [57] incrementally query a foundation model to generate N samples, and leverage multiple agents to select a finale response via majority voting.

Related patterns:

- *Cross-reflection.* An agent can query multiple agents to provide feedback, which can be determined via *voting-based cooperation* between the reflective agents.
- *Role-based and debate-based cooperation.* *Voting-based cooperation* can be regarded as an alternative to other cooperation patterns by hosting a vote between agents, whilst they can be applied together to complement each other.
- *Tool/agent registry.* Agents participating in the voting process can be employed via *tool/agent registry*.

4.13 Role-based Cooperation

Summary: Agents are assigned assorted roles and decisions are finalised in accordance with their roles.

Context: Multiple agents can be leveraged within a compound AI system. Agents need to collaborate on the same task while having their own perspectives.

Problem: How can agents cooperate on certain tasks considering their specialties?

Forces:

- *Diversity.* The employed agents can have diverse opinions of how a plan is constructed or how a task should be completed.
- *Division of labor.* As agents can be trained with different corpus for various purposes, their strengths and expertise should be taken into consideration for task completion.
- *Fault tolerance.* Agents may be unavailable during cooperation, which will affect the eventual task result.

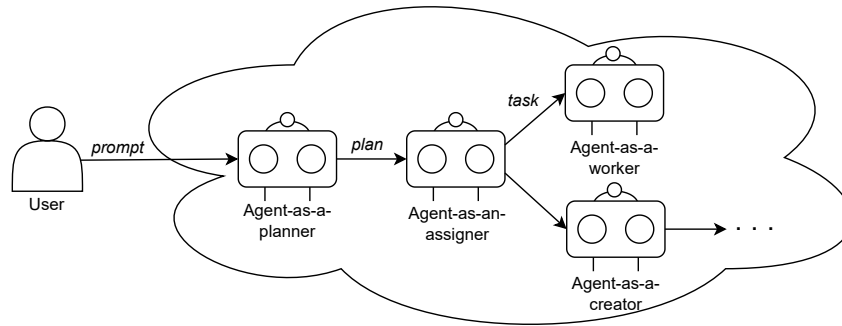


Figure 13: Role-based cooperation.

Solution: Fig. 13 illustrates a high-level graphical representation of *role-based cooperation*, where agents coordinate in a hierarchical scheme. In particular, agents can be assigned certain roles and hence establishing a workflow via persona specification, task definition, tool employment, and process orchestration. For example, an agent-as-a-planner can generate a multi-step plan by decomposing user’s goal into a chain of tasks. Subsequently, the agent-as-an-assigner can orchestrate task assignment, i.e., some tasks can be completed by the assigner itself, while other tasks can be delegated to certain agent-as-a-worker based on their domain-specific capabilities and expertise. In addition, if there is no available agent, agent-as-a-creator can be invoked to create a new agent with a specific role, by providing necessary resources, clear objectives and initial guidance to ensure a seamless transition of tasks and responsibilities. Please note that more elaborate roles can be defined and assigned to the agents.

Consequences:**Benefits:**

- *Division of labor.* Agents can simulate the division of labor in the real world according to their roles, which enables the observation of social phenomena.

- *Fault tolerance.* Since multiple agents are leveraged, the system can continue operation by replacing inactive agents with other agents of the same role.
- *Scalability.* Agents of new roles can be employed or created anytime to refine the task workflow and extend the capability of the whole system.
- *Accountability.* Accountability is facilitated as the responsibilities of agents are attributed clearly regarding their expected roles.

Drawbacks:

- *Overhead.* Cooperation between agents will increase communication overhead, while agent services with different roles may have different prices.

Known uses:

- *XAgent*¹⁸. XAgent consists of three main parts: planner agent for task generation, dispatcher agent for task assignment, and tool agent for task completion.
- *MetaGPT* [58]. MetaGPT utilises various agents acting as different roles (e.g., architect, project manager, engineer) in standardized operating procedures.
- *MedAgents* [59]. Agents are assigned roles as various domain experts (e.g. cardiology, surgery, gastroenterology) to provide specialised analysis and collaboratively work on healthcare issues.
- Wang et al. [60] propose Mixture-of-Agents where proposer agents provide useful reference responses to aggregator agents, and the aggregator agents are composed in layers to synthesise and refine the responses.

Related patterns:

- *Cross-reflection.* An agent can query multiple agents to provide feedback, which can be determined via *role-based cooperation* between the reflective agents.
- *Voting-based and debate-based cooperation.* *Role-based cooperation* can be regarded as an alternative of other cooperation patterns by clearly assigning roles to agents, which will then work and collaborate according to the given roles. Whilst, these patterns can be applied together to complement each other.
- *Tool/agent registry.* Agents with different roles can be searched and employed via *tool/agent registry*.

4.14 Debate-based Cooperation

Summary: An agent receives feedback from other agents, and adjusts the thoughts and behaviours during the debate with other agents until a consensus is reached.

Context: A compound AI system can integrate multiple agents to provide more comprehensive services. The included agents need to collaborate on the same task while having their own perspectives.

Problem: How to leverage multiple agents to create refined responses, while facilitating the evolution of agents.

Forces:

- *Diversity.* Different agents may have various opinions to help refine the generated responses to users.
- *Lack of adaptability.* An agent may exhibit limited creativity in reasoning and response generation when given new context or tasks.
- *Lack of explainability.* The interaction process of agents should be interpreted for auditing if violations are detected.

Solution: Fig. 14 depicts a high-level graphical representation of *debate-based cooperation*. A user can send queries to an agent, which will then share the questions with other agents. Given the shared question, each agent generates its own initial responses, and subsequently, a round of debate will start between the agents. Agents will propagate their initial response in a decentralised manner to each other for verification, while also providing instructions and potential directions to construct a more comprehensive response based on inclusive and collective outcomes. In addition,

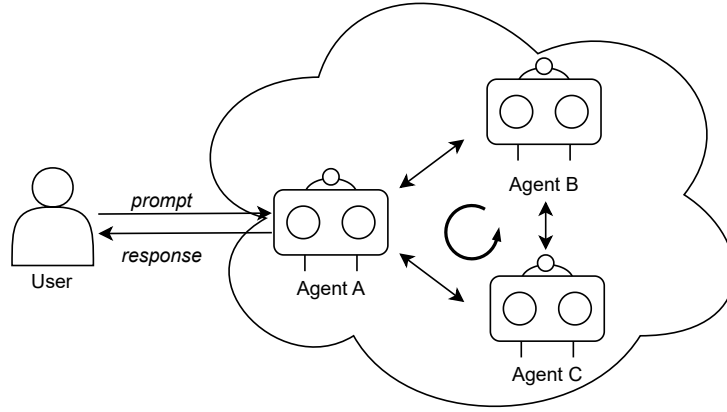


Figure 14: Debate-based cooperation.

agents may utilise a shared memory in certain circumstances, or allow each other to access the respective memory for facilitating the debate. This debate process can be iterative to enhance the performance of all participating agents. *Debate-based cooperation* can end according to a predefined number of debate rounds, or the agents will continue the procedure until a consensus answer is obtained.

Consequences:

Benefits:

- *Adaptability.* Agents can adapt to other agents during the debate procedure, achieving continuous learning and evolution.
- *Explainability.* *Debate-based cooperation* is structured with agents' arguments and presented evidence, preserving transparency and explainability of the whole procedure.
- *Critical thinking.* Arguing with other agents can help an agent develop the ability of critical thinking for future reasoning process.

Drawbacks:

- *Limited capability.* The effectiveness of *debate-based cooperation* relies on agents' capabilities of reasoning, argument, and evaluation of other agents' statement.
- *Data privacy.* Agents may need to withhold certain sensitive information, which can affect the debate process.
- *Overhead.* The complexity of debate may increase the communication and computation overhead.
- *Scalability preservation.* The system scalability may be affected as the number of participating agents increases. The coordination of agents and processing of their arguments may become complex.

Known uses:

- *crewAI*¹⁹. crewAI provides a multi-agent orchestration framework where multiple agents can be grouped for discussion on a given topic.
- Liang et al. [61] leverage multi-agent debate to address the issue of "Degeneration-of-Thought". Within the debate, an agent needs to persuade another and correct the mistakes.
- Du et al. [62] employ multiple agents to discuss the given user input, and the experiment results indicate that the agents can converge on a consensus answer after multiple rounds.
- Chen et al. [63] explore the negotiation process in a multi-agent system, where each agent can perceive the outcomes of other agents, and adjust its own strategies.

¹⁹<https://www.crewai.com/>

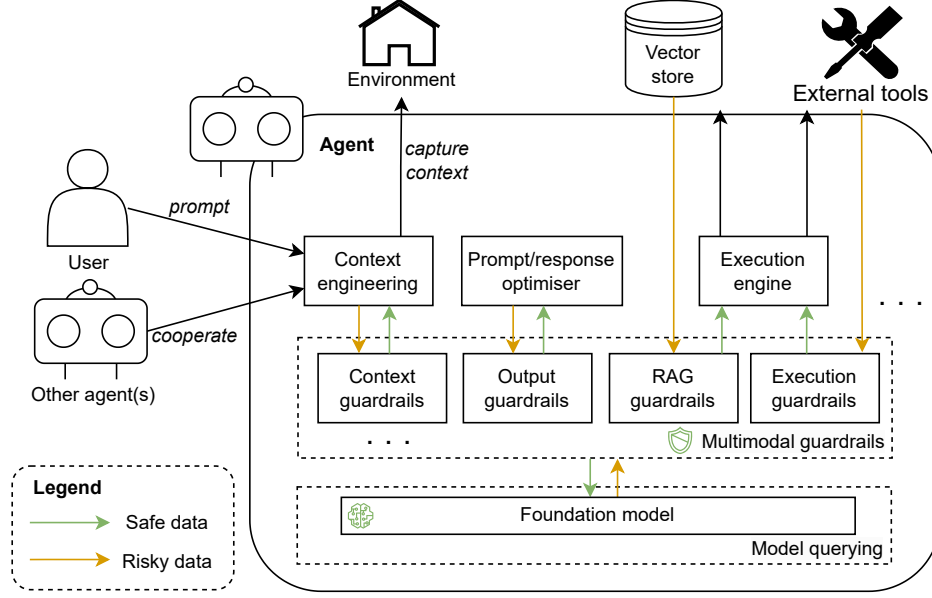


Figure 15: Multimodal guardrails.

- Li et al. [64] propose a framework including peer rank and discussion between agents, to mitigate the biases in automated evaluation process.

Related patterns:

- *Cross-reflection.* Agents can decide the reflective feedback to another agent via *debate-based cooperation*.
- *Voting-based and role-based cooperation.* *Debate-based cooperation* can be regarded as an alternative of other cooperation patterns by hosting a debate between agents, whilst they can be applied together to complement each other.
- *Tool/agent registry.* Agents participating in the debate process can be employed via *tool/agent registry*.

4.15 Multimodal Guardrails

Summary: Multimodal guardrails can control the inputs and outputs of foundation models to meet specific requirements such as user requirements, ethical standards, and laws.

Context: An agent consists of foundation model and other components. When users prompt specific goals to the agent, the underlying foundation model is queried for goal achievement.

Problem: How to prevent the foundation model from being influenced by adversarial inputs, or generate harmful or undesirable outputs to users and other components?

Forces:

- *Robustness.* Adversarial information may be sent to the foundation model, which will affect the model's memory and all subsequent reasoning processes and results.
- *Safety.* Foundation models may generate inappropriate responses due to hallucinations, which can be offensive to users, and disturb the operation of other components (e.g., other agents, external tools).
- *Standard alignment.* Agents and the underlying foundation models should align with the specific standards and requirements in industries and organisations.

Solution: Fig. 15 presents a simplified graphical representation of *multimodal guardrails*. Guardrails can be applied as an intermediate layer between the foundation model and all other components in a compound AI system. When

users input prompts or other components (e.g. memory) send any message to the foundation model, guardrails can first verify whether the information meets specific predefined requirements. Only valid information are delivered to the foundation model, while risky or sensitive data will be processed before being transferred. For instance, personally identifiable information should be treated with care or removed to protect privacy. Guardrails can evaluate the contents either relying on predefined examples, or in a “reference-free” manner. Equivalently, when the foundation model creates results, guardrails need to ensure that the responses do not include biased or irrespective information to users, or fulfil the particular requirements of other system components. Please note that a set of guardrails can be implemented where each of them is responsible for specialised interactions, e.g., information retrieval from datastore, validation of users’ input, external API invocation, etc. Meanwhile, guardrails are capable of processing multimodal data such as text, audio, video to provide comprehensive monitoring and control.

Consequences:

Benefits:

- *Robustness.* Guardrails preserve the robustness of foundation models by filtering the inappropriate context information.
- *Safety.* Guardrails serve as validators of foundation model outcomes, ensuring the generated responses do not harm agent users.
- *Standard alignment.* Guardrails can be configured referring to organisational policies and strategies, ethical standards, and legal requirements to regulate the behaviours of foundation models.
- *Adaptability.* Guardrails can be implemented across various foundation models and agents, and deployed with customised requirements.

Drawbacks:

- *Overhead.* i) Collecting diverse and high-quality corpus to develop *multimodal guardrails* may be resource-intensive. ii) Real-time processing multimodal data can increase the computational requirements and costs.
- *Lack of explainability.* The complexity of *multimodal guardrails* makes it difficult to explain the finalised outputs.

Known uses:

- *NeMo guardrails* [65]. NVIDIA released NeMo guardrails, which are specifically designed to ensure the coherency of dialogue between users and AI systems, and prevent negative impact of misinformation and sensitive topics.
- *Llama guard* [66]. Meta published Llama guard, a foundation model based safeguard model fine-tuned via a safety risk taxonomy. Llama guard can identify the potentially risky or violating content in users’ prompts and model outputs.
- *Guardrails AI*²⁰. Guardrails AI provides a hub, listing various validators for handling different risks in the inputs and outputs of foundation models.

Related patterns:

- *Proactive goal creator.* *Multimodal guardrails* can help process the multimodal data captured by *proactive goal creator*.
- *One-shot and incremental model querying.* *Multimodal guardrails* serve as an intermediate layer, managing the inputs and outputs of model querying.

4.16 Tool/Agent Registry

Summary: The tool/agent registry maintains a unified and convenient source to select diverse agents and tools.

Context: Within an agent, the task executor may cooperate with other agents or leverage external tools for expanded capabilities.

²⁰<https://www.guardrailsai.com/>

Problem: There are diverse agents and tools, how can the agent efficiently select the appropriate external agents and tools?

Forces:

- *Discoverability.* It may be difficult for users and agents to discover the available agents and tools considering the diversity.
- *Efficiency.* Users/agents need to finalise agent and tool selection within a certain time period.
- *Tool appropriateness.* Particular tasks may have specific requirements of agents/tools (e.g. certain capabilities).

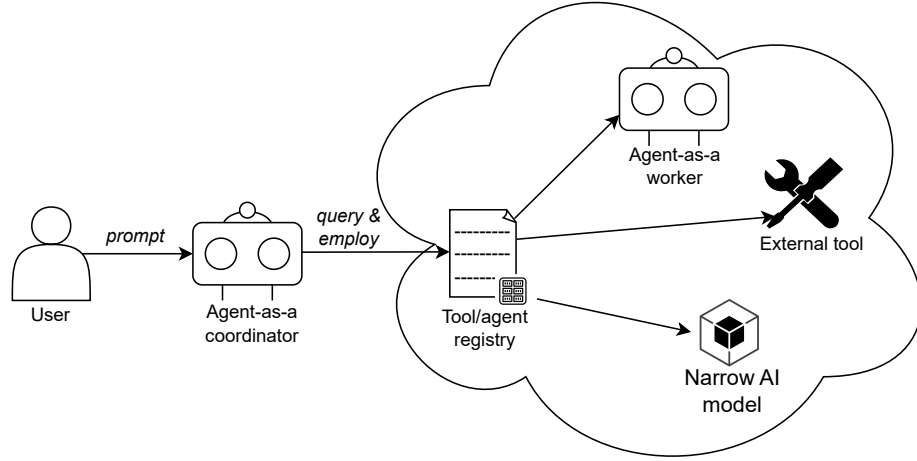


Figure 16: Tool/agent registry.

Solution: Fig. 16 depicts how an agent searches external agents and tools via a *tool/agent registry*. A user prompts goals to an agent, which then decomposes the goals into fine-grained tasks. The agent can query the *tool/agent registry*, which is the main entry point for collecting and categorising various tools and agents regarding a series of metrics (e.g., domain-specific capability, price, context window). Based on the returned information, the agent can employ and assign the tasks to respective tools and agents. Please note that a registry can be implemented in different manners, for instance, a coordinator agent with specific knowledge base, blockchain-based smart contract, etc., and a registry can be extended into a marketplace for tool/agent service trading.

Consequences:

Benefits:

- *Discoverability.* The registry provides a catalogue for users and agents to discover tools and agents with different capabilities.
- *Efficiency.* The registry offers an intuitive inventory listing the attributes (e.g., performance, price) of tools and agents, which saves time for comparison.
- *Tool appropriateness.* Given the task requirements and conditions, users and agents can select the most appropriate tools/agents according to the provided attributes.
- *Scalability.* The registry only stores certain metadata about tools and agents, hence the data structure is simple and lightweight, which ensures the scalability of the registry.

Drawbacks:

- *Centralisation.* The registry may become a vendor lock-in solution and cause single point of failure. It may be manipulated and compromised if it is maintained by external entities.
- *Overhead.* Implementing and maintaining a tool/agent registry can introduce additional complexity and overhead.

Known uses:

- *GPTStore*²¹. GPTStore provides a catalogue for searching ChatGPT-based agents.
- *TPTU* [67]. TPTU incorporates a toolset to broaden the capabilities of AI Agents.
- *VOYAGER* [68]. VOYAGER can store action programs and hence incrementally establish a skill library for reusability.
- *OpenAgents* [69]. An agent is specifically developed to manage the API invocation of plugins.

Related patterns:

- *Cross-reflection*. The agent can search reflective agents for *cross-reflection* via *tool/agent registry*.
- *Voting-based, role-based and debate-based cooperation*. *Tool/agent registry* can provide a source of agents for the cooperation patterns.
- *Agent adapter*. *Tool/agent registry* records the available external tools, while *agent adapter* can convert the interface of selected tools into agent-friendly format.

4.17 Agent Adapter

Summary: An agent adapter provides interface to connect the agent and external tools for task completion.

Context: An agent may leverage external tools to complete certain tasks for expanded capabilities.

Problem: The agent needs to deal with different interfaces of diverse tools, while certain interfaces might be incompatible or inefficient to interact for the agent. How can the agent assign tasks to external tools and process the results?

Forces:

- *Interoperability*. Certain tasks require external tools to complete, and the tools may need agents to process particular information during intermediate steps.
- *Adaptability*. Agents may employ new tools considering task complexity, tool capability, cost, etc.
- *Overhead*. Manual development of compatible interfaces for agents and external tools can be intensive and inefficient.

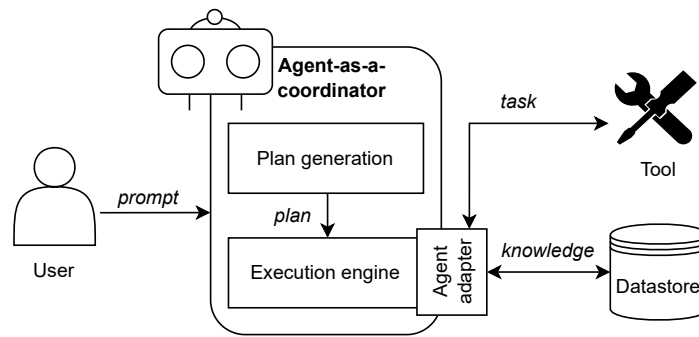


Figure 17: Agent adapter.

Solution: Fig. 17 demonstrates a simplified graphical representation of *agent adapter*. Given user’s instructions, the agent generates a plan consisting of a set of tasks to achieve the user’s goals. In particular, the agent may employ diverse external tools to complete different tasks. However, tools have respective interfaces, which can be of different abstraction levels for the agent to deal with, or have specific format requirements, etc. *Agent adapter* can help invoke and manage these interfaces by converting the agent messages into required format or content, and vice versa. In

²¹<https://gptstore.ai/>

particular, the adapter can retrieve tool manual or tutorial from datastore, to acquire available interfaces and learn the usage. It then transforms the agent outputs based on the interface requirements and invokes the service [70]. Please note that fine-grained interface description can enhance agent understanding and hence improve the performance. The adapter also receives execution results from tools, which will be sent to the underlying foundation model for further analysis (e.g. task assignment to other tools, self-reflection for tool employment). For instance, the adapter can translate tasks into system messages when interacting with local file system, or capture and operate graphical user interface when playing a video game.

Consequences:

Benefits:

- *Interoperability.* *Agent adapter* facilitates the interoperation between an agent and external tools.
- *Adaptability.* Agents can employ new tools via *agent adapter*, which can acquire and convert the tool API via corresponding manual or tutorial.
- *Reduced development cost.* *Agent adapter* enables autonomous conversion of interfaces, there is no need to develop compatible interfaces for different tools, hence the development cost is reduced.

Drawbacks:

- *Maintenance overhead.* i) *Agent adapter* itself requires proper maintenance and evaluation to ensure the correctness of outputs. ii) *Agent adapter* may need additional memory or external data store to record the historical tool interfaces.

Known uses:

- *AutoGen*²². Users can register different tools in the agent, specifying the usage description. Registered tools will be leveraged by the agent during a conversation with user.
- *Apple Intelligence*²³. *Apple Intelligence* can support writing, image generation, schedule management across different products and applications. For instance, it can capture the entities in users' photo library and create emoji.
- *Semantic Kernel*²⁴. *Semantic Kernel* can orchestrate agents and plugins to extend agents' skills. Plugins need to provide semantic description (e.g. input, output, side effects) for agents to understand.
- Yang et al. [71] devise SWE-agent that can provide agent-computer interfaces, enabling foundation model-based agents to process code commands and resolve software engineering tasks.

Related patterns:

- *Prompt/response optimiser.* *Prompt/response optimiser* can improve users' inputs, and the optimised prompts can be sent to other agents for goal achievement, while *agent adapter* focuses more on the utilisation of external tools.
- *Tool/agent registry.* *Tool/agent registry* records the available external tools, while *agent adapter* can convert the interface of selected tools into agent-friendly format.

4.18 Agent Evaluator

Summary: Agent evaluator can perform testing to assess the agent regarding diverse requirements and metrics.

Context: Within an agent, the underlying foundation model and a series of components coordinate to conduct reasoning and generate the responses given users' prompts.

Problem: How to assess the performance of agents to ensure they behave as intended?

Forces:

²²<https://microsoft.github.io/autogen/docs/tutorial/tool-use>

²³<https://www.apple.com/apple-intelligence/>

²⁴<https://learn.microsoft.com/en-us/semantic-kernel/agents/plugins/?tabs=Csharp>

- *Functional suitability guarantee.* Agent developers need to ensure that a deployed agent operates as intended, providing complete, correct, and appropriate services to users.
- *Adaptability improvement.* Agent developers need to understand and analyse the usage of agents in specific scenarios, to perform suitable adaptations.

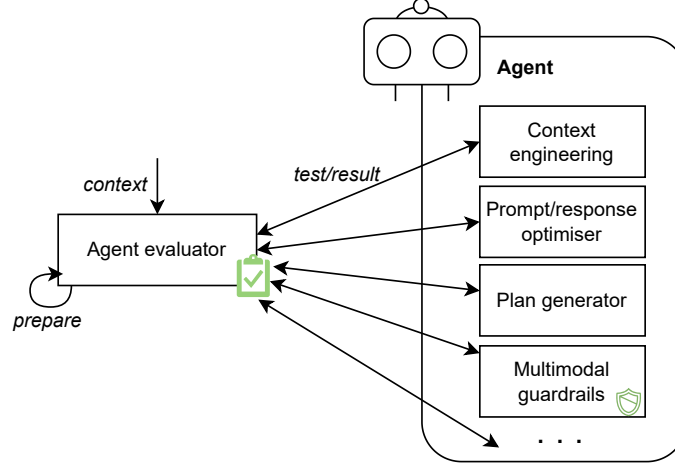


Figure 18: Agent evaluator.

Solution: Fig. 18 presents a simplified graphical representation of *agent evaluator*. Developers can deploy evaluator to assess the agent regarding responses and reasoning process at both design-time and runtime. Specifically, developers need to build up the evaluation pipeline, for instance, by defining specific scenario-based requirements, metrics and expected outputs from agents. Given particular context, the *agent evaluator* prepares context-specific test cases (either searching from external resources or generating by itself), and performs evaluation on the agent components respectively. The evaluation results provide valuable feedback such as boundary cases, near-misses, etc., while developers can further fine-tune the agent or employ corresponding risk mitigation solutions, and also upgrade the evaluator based on the results.

Consequences:

Benefits:

- *Functional suitability.* Agent developers can learn the agent’s behavior, and compare the actual responses with expected ones through the evaluation results.
- *Adaptability.* Agent developers can analyse the evaluation results regarding scenario-based requirements, and decide whether the agent should adapt to new requirements or test cases.
- *Flexibility.* Agent developers can define customised metrics and the expected outputs to test a specific aspect of the agent.

Drawbacks:

- *Metric quantification.* It is difficult to design quantified rubrics for the assessment of software quality attributes.
- *Quality of evaluation.* The evaluation quality is dependent on the prepared test cases.

Known uses:

- *Inspect*²⁵. UK AI Safety Institute devised an evaluation framework for large language models that offers a series of built-in components, including prompt engineering, tool usage, etc.
- *DeepEval*²⁶. DeepEval incorporates 14 evaluation metrics, and supports agent development frameworks such as LlamaIndex, Hugging Face, etc.

²⁵https://ukgovernmentbeis.github.io/inspect_ai/

²⁶<https://docs.confident-ai.com/>

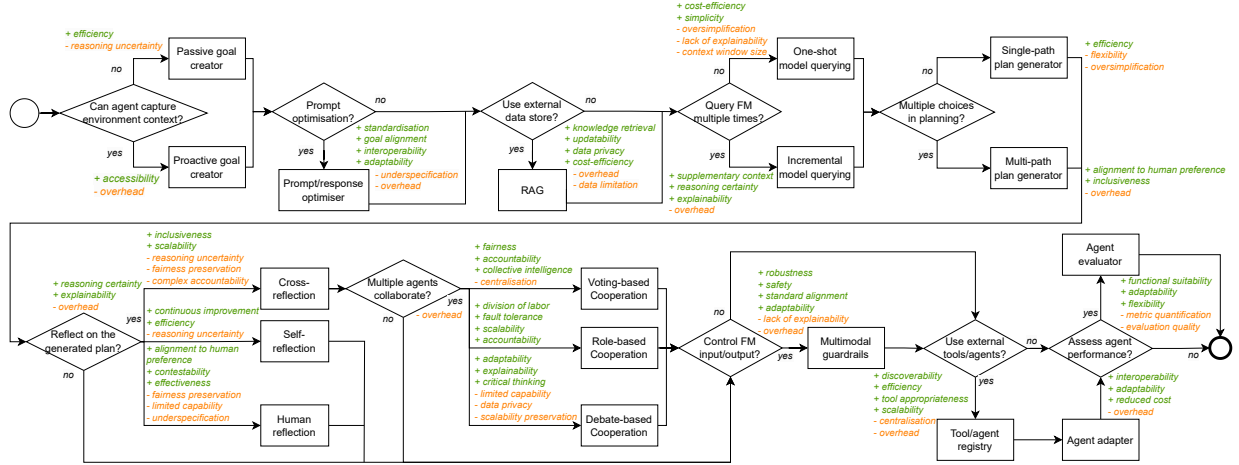


Figure 19: Decision model for agent design pattern catalogue.

- *Promptfoo*²⁷. Promptfoo can provide efficient evaluation services with caching, concurrency, and live reloading, and also enable automate scoring based on user-defined metrics.
- *Ragas*²⁸. Ragas facilitates evaluation on the RAG pipelines via test dataset generation and leveraging LLM-assisted evaluation metrics.

Related patterns: *Agent evaluator* can be configured and deployed to assess the performance of other pattern-oriented agent components during both design-time and runtime.

5 Lessons Learned From the Pattern Catalogue

In this section, we first propose a decision model for selecting the 18 identified patterns to enhance the development of foundation mode based agents. Afterwards, we would like to share our experiences on the application of certain patterns in research projects as case studies. Finally, we discuss several insights we learned during the pattern collection process and previous research works.

5.1 Decision Model

Fig. 19 illustrates a decision model which visualises the selection process of different patterns for practitioners. In particular, for a design problem, each decision can map to the corresponding solution space, which is regarded as the alternative pattern(s) for those in the opposite decision’s solution space. Further, within the same solution space, there can be multiple patterns serving as complements of each other. The strengths and trade-offs of each pattern are highlighted in green and orange colour respectively. Please note that the decision model omits the common attributes for alternative patterns as either decision will incorporate these attributes as consequences, and it centralises the shared attributes for complementary patterns for brevity. In addition, there is no specific selection or application sequence, considering the patterns aim to facilitate the design and development of different architectural components, which can be implemented via a decoupling strategy. A brief explanation is provided as follows.

If the agent is expected to capture users’ environmental information as supplementary context, *proactive goal creator* can be applied to analyse users’ goals based on the data captured by a series of sensors. In this case, the agent can also serve users with specified disabilities. Whilst, *passive goal creator* can provide a simple and efficient dialogue interface to interact with users. *Prompt/response optimiser* can enhance the goal alignment by refining users’ instructions into standardised prompts. Meanwhile, the agent can retrieve more information from external knowledge base via *retrieval augmented generation* whenever a component requires additional context. Besides, a component can query the incorporated foundation model for a single purpose multiple times (*incremental model querying*) based on either user-specified requirements or system configuration, to provide supplementary context and hence improving model reasoning certainty, or just query once (*one-shot model querying*) due to limited budget for model calling.

²⁷<https://github.com/promptfoo/promptfoo>

²⁸<https://github.com/explodinggradients/ragas>

For a particular goal, agents can create a linear plan via *single-path plan generator* for efficiency, or a complex plan in which each step has different options via *multi-path plan generator* to ensure inclusiveness and alignment to human preferences. A generated plan can be assessed via multiple solutions (i.e. *self-reflection*, *cross-reflection* and *human reflection*) for ensuring plan correctness and feasibility, and improving agent reasoning certainty and explainability, while each solution has respective strengths and trade-offs. Further, multiple agents can be employed for reflection, and they can interact with each other in terms of *voting-based*, *role-based* and *debate-based* cooperation schemes.

When the underlying foundation model is queried, malicious inputs can affect the reasoning process, and the model may continuously learn and generate skewed outputs. *Multimodal guardrails* can provide a layer between the foundation model and other components by inspecting the model inputs and outputs through both rule-based and AI-based examination. In case that external tools or agents are leveraged for certain tasks, a *tool/agent registry* can enhance discoverability, and ensure the efficiency and appropriateness of tools. *Agent adapter* guarantees the interoperability and adaptability of an agent to call external services whilst reducing the cost for manual development and maintenance. Finally, *agent evaluator* can be utilised to assess the agent’s functional suitability before release.

5.2 Pattern Application

In this section, we would like to discuss and share our experiences of applying the identified patterns in research projects. Researchers joined a project in which we are designing and developing an agent platform for creating and operating bespoke foundation model based agents with continuous learning capability. Specifically, the platform is divided into multiple products and components where different patterns can be applied.

Universal Task Assistant²⁹, developed by Xie et al., can help users discover and learn the usage of different mobile apps. This product applied *proactive goal creator* to capture the mobile UI screen and detect the included elements, then it can perform actions to complete the user’s tasks. Cheng et al. proposed an AI chain integrated development environment, Prompt Sapper [72], for practitioners to properly and seamlessly develop their own FM-based AI chain services. In particular, the Prompt Sapper co-pilot implemented *passive goal creator* for eliciting users’ requirements, *prompt/response optimiser* for refining users’ task descriptions and generating AI chain skeleton, and *tool/agent registry* for managing all available artifacts. Shamsujjoha et al. [73] further explored *multimodal guardrails* by devising a taxonomy of guardrails, which provides comparative analysis of diverse design options in terms of the actions, targets, scopes, rules, autonomy, modalities, and underlying techniques when incorporating guardrails in foundation model based systems. Whilst, Xia et al. [74] demonstrated an AI system evaluation framework, categorising the evaluation of foundation model based applications into system-level and component-level, each granularity has respective testing methods and benchmarking. The framework can offer guidance for the application of *agent evaluator*.

5.3 Discussion

Integration with extant patterns. Integrating different patterns can help structure and develop comprehensive and trustworthy agents. In particular, the proposed pattern catalogue can be applied together with responsible AI patterns [75] to ensure the agents behave in a responsible manner. For example, *bill of materials registry* can record the procurement of components such as guardrails, prompt/response optimiser, etc., offering a complete supply chain of foundation model-based agent development. Whilst, *black box* can be implemented to collect the inputs and outputs of foundation model-based agents at runtime. If any abnormal behaviour is detected or an audit is needed, the recorded information can provide evidence for the accountability process. Further, the collaboration patterns among agents can refer to existing social learning evolution models [76]. For instance, selecting suitable topologies and implementing effective control mechanisms can enhance the design of multi-agent workflows. Agents might also benefit from imitative learning in *debate-based cooperation*, enabling dynamic adaptation and knowledge transfer. In addition, a set of voting mechanisms can be leveraged via blockchain smart contracts [77] for *voting-based cooperation*, to ensure transparent and secure interactions between agents.

Compliance with regulations and standards. Preserving the alignment of agents with both international and domestic regulations and standards should be noted as a fundamental factor for developers to provide agent services in different countries and regions. European Parliament has approved the Artificial Intelligence Act³⁰ which focuses on four risk categories of AI applications. NIST released a draft publication for managing the risks of generative AI³¹, and ISO published an international standard for implementing and maintaining AI management systems [78]. Future work can extract and analyse the requirements within each regulation and standard, and map the proposed pattern catalogue to the requirements. In particular, we are adopting the concept of DevOps into FM-based agents, where both agent

²⁹<https://apputa.online/>

³⁰<https://artificialintelligenceact.eu/>

³¹<https://www.nist.gov/itl/ai-risk-management-framework>

creation and operation information will be recorded for further auditing. Meanwhile, the agent workflow generation also requires inspection that whether it adheres to domain specifications.

Evaluation of foundation model-based agents. Evaluations on agents and the underlying foundation models are significant to ensure they behave as intended. The majority of pattern benefits and drawbacks are software quality attributes, which still require quantification for fine-grained metrics and rubrics. For instance, accountability can be further divided into three criteria of responsibility, auditability, and redressability, and each criterion has its own process, resource, and product metrics [79]. Proper quantification can promote the evaluation of agents and validate the effectiveness of applied patterns. Further, authors are also exploring real-time exception identification and handling in agent workflow execution, which includes rationalising task dependencies, external tool API version matching, etc.

6 Conclusion

Foundation model based agents are gaining increasing attention in various domains to intellectualise and automate the business process. However, practitioners are troubled by architectural challenges to design agents. Our previous work demonstrates a reference architecture to present an overview of agent design [5], while in this study, we scrutinise the forces, solutions, and trade-offs of 18 patterns. The pattern catalogue is provided as a holistic guidance for architects to better design and develop foundation model-based agents. In our future work, we will study how to apply the pattern catalogue with existing patterns to preserve the trustworthiness of agents, and further explore the architecture decisions that are related to foundation model-based agents.

References

- [1] R. Bommasani, D. A. Hudson, E. Adeli, R. Altman, S. Arora, S. von Arx, M. S. Bernstein, J. Bohg, A. Bosselut, E. Brunskill *et al.*, “On the opportunities and risks of foundation models,” *arXiv preprint arXiv:2108.07258*, 2021.
- [2] U. Anwar, A. Saparov, J. Rando, D. Paleka, M. Turpin, P. Hase, E. S. Lubana, E. Jenner, S. Casper, O. Sourbut *et al.*, “Foundational challenges in assuring alignment and safety of large language models,” *arXiv preprint arXiv:2404.09932*, 2024.
- [3] A. Chan, R. Salganik, A. Markelius, C. Pang, N. Rajkumar, D. Krasheninnikov, L. Langosco, Z. He, Y. Duan, M. Carroll, M. Lin, A. Mayhew, K. Collins, M. Molamohammadi, J. Burden, W. Zhao, S. Rismani, K. Voudouris, U. Bhatt, A. Weller, D. Krueger, and T. Maharaj, “Harms from increasingly agentic algorithmic systems,” in *Proceedings of the 2023 ACM Conference on Fairness, Accountability, and Transparency*, ser. FAccT ’23. New York, NY, USA: Association for Computing Machinery, 2023, p. 651–666. [Online]. Available: <https://doi.org/10.1145/3593013.3594033>
- [4] A. D’Amour, K. Heller, D. Moldovan, B. Adlam, B. Alipanahi, A. Beutel, C. Chen, J. Deaton, J. Eisenstein, M. D. Hoffman, F. Hormozdiari, N. Houlsby, S. Hou, G. Jerfel, A. Karthikesalingam, M. Lucic, Y. Ma, C. McLean, D. Mincu, A. Mitani, A. Montanari, Z. Nado, V. Natarajan, C. Nielson, T. F. Osborne, R. Raman, K. Ramasamy, R. Sayres, J. Schrouff, M. Seneviratne, S. Sequeira, H. Suresh, V. Veitch, M. Vladymyrov, X. Wang, K. Webster, S. Yadlowsky, T. Yun, X. Zhai, and D. Sculley, “Underspecification presents challenges for credibility in modern machine learning,” *Journal of Machine Learning Research*, vol. 23, no. 226, pp. 1–61, 2022. [Online]. Available: <http://jmlr.org/papers/v23/20-1335.html>
- [5] Q. Lu, L. Zhu, X. Xu, Z. Xing, S. Harrer, and J. Whittle, “Towards responsible generative ai: A reference architecture for designing foundation model based agents,” *ICSA’23*, 2023.
- [6] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altschmidt, S. Altman, S. Anadkat *et al.*, “Gpt-4 technical report,” *arXiv preprint arXiv:2303.08774*, 2024.
- [7] K. Hu, “Chatgpt sets record for fastest-growing user base - analyst note,” <https://www.reuters.com/technology/chatgpt-sets-record-fastest-growing-user-base-analyst-note-2023-02-01/>, 2023, accessed 3-September-2024.
- [8] S. Schulhoff, M. Ilie, N. Balepur, K. Kahadze, A. Liu, C. Si, Y. Li, A. Gupta, H. Han, S. Schulhoff *et al.*, “The prompt report: A systematic survey of prompting techniques,” *arXiv preprint arXiv:2406.06608*, 2024.
- [9] A. Ng, <https://www.deeplearning.ai/the-batch/issue-242/>, 2024, accessed 3-September-2024.
- [10] C. Packer, V. Fang, S. G. Patil, K. Lin, S. Wooders, and J. E. Gonzalez, “Memgpt: Towards llms as operating systems,” *arXiv preprint arXiv:2310.08560*, 2024.
- [11] S. Colabianchi, A. Tedeschi, and F. Costantino, “Human-technology integration with industrial conversational agents: A conceptual architecture and a taxonomy for manufacturing,” *Journal of Industrial Information*

- Integration*, vol. 35, p. 100510, 2023. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S2452414X23000833>
- [12] “Openai’s bet on a cognitive architecture,” <https://blog.langchain.dev/openais-bet-on-a-cognitive-architecture/>, 2023, accessed 3-September-2024.
 - [13] R. Jain, “Design Patterns for Compound AI Systems (Conversational AI, CoPilots & RAG),” <https://medium.com/@raunak-jain/design-patterns-for-compound-ai-systems-copilot-rag-fa911c7a62e0>, 2024, accessed 3-September-2024.
 - [14] D. Zhou, X. Xue, X. Lu, Y. Guo, P. Ji, H. Lv, W. He, Y. Xu, Q. Li, and L. Cui, “A hierarchical model for complex adaptive system: From adaptive agent to ai society,” *ACM Trans. Auton. Adapt. Syst.*, Aug. 2024, just Accepted. [Online]. Available: <https://doi.org/10.1145/3686802>
 - [15] E. Yan, B. Bischof, C. Frye, H. Husain, J. Liu, and S. Shankar, “What We Learned from a Year of Building with LLMs (Part I),” <https://www.oreilly.com/radar/what-we-learned-from-a-year-of-building-with-llms-part-i/>, 2024, accessed 3-September-2024.
 - [16] A. E. Hassan, D. Lin, G. K. Rajbahadur, K. Gallaba, F. R. Cogo, B. Chen, H. Zhang, K. Thangarajah, G. A. Oliva, J. Lin *et al.*, “Rethinking software engineering in the era of foundation models: A curated catalogue of challenges in the development of trustworthy finware,” *arXiv preprint arXiv:2402.15943*, 2024.
 - [17] S. Gao, A. Fang, Y. Huang, V. Giunchiglia, A. Noori, J. R. Schwarz, Y. Ektefaie, J. Kondic, and M. Zitnik, “Empowering biomedical discovery with ai agents,” *arXiv preprint arXiv:2404.02831*, 2024.
 - [18] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, 3rd ed. Addison-Wesley Professional, 2012.
 - [19] M. Nafreen, S. Bhattacharya, and L. Fiondella, “Architecture-based software reliability incorporating fault tolerant machine learning,” in *2020 Annual Reliability and Maintainability Symposium (RAMS)*, 2020, pp. 1–6.
 - [20] G. Meszaros and J. Doble, *Pattern languages of program design 3*. Addison-Wesley Longman Publishing Co., Inc., 1997, p. 529–574.
 - [21] Y. Liu, S. Chen, H. Chen, M. Yu, X. Ran, A. Mo, Y. Tang, and Y. Huang, “How ai processing delays foster creativity: Exploring research question co-creation with an llm-based agent,” *arXiv preprint arXiv:2310.06155*, 2023.
 - [22] S. S. Kannan, V. L. Venkatesh, and B.-C. Min, “Smart-llm: Smart multi-agent robot task planning using large language models,” *arXiv preprint arXiv:2309.10062*, 2023.
 - [23] D. Ha and J. Schmidhuber, “World models,” *arXiv preprint arXiv:1803.10122*, 2018.
 - [24] Y. LeCun, “A path towards autonomous machine intelligence version 0.9. 2, 2022-06-27,” *Open Review*, vol. 62, no. 1, 2022.
 - [25] X. Zeng, X. Wang, T. Zhang, C. Yu, S. Zhao, and Y. Chen, “Gesturegpt: Zero-shot interactive gesture understanding and grounding with large language model agents,” *arXiv preprint arXiv:2310.12821*, 2023.
 - [26] D. Zhao, Z. Xing, X. Xia, D. Ye, X. Xu, and L. Zhu, “Seehow: Workflow extraction from programming screencasts through action-aware video analytics,” *arXiv preprint arXiv:2304.14042*, 2023.
 - [27] C. Zhang, K. Yang, S. Hu, Z. Wang, G. Li, Y. Sun, C. Zhang, Z. Zhang, A. Liu, S.-C. Zhu *et al.*, “Proagent: Building proactive cooperative ai with large language models,” *arXiv preprint arXiv:2308.11339*, 2023.
 - [28] A. Zhao, D. Huang, Q. Xu, M. Lin, Y.-J. Liu, and G. Huang, “Expel: Llm agents are experiential learners,” *arXiv preprint arXiv:2308.10144*, 2023.
 - [29] R. Schumann, W. Zhu, W. Feng, T.-J. Fu, S. Riezler, and W. Y. Wang, “Velma: Verbalization embodiment of llm agents for vision and language navigation in street view,” *arXiv preprint arXiv:2307.06082*, 2023.
 - [30] Y. Hu and Y. Lu, “Rag and rau: A survey on retrieval-augmented language model in natural language processing,” *arXiv preprint arXiv:2404.19543*, 2024.
 - [31] S. Wang, E. Khramtsova, S. Zhuang, and G. Zuccon, “Feb4rag: Evaluating federated search in the context of retrieval augmented generation,” *arXiv preprint arXiv:2402.11891*, 2024.
 - [32] J. Larson and S. Truitt, “GraphRAG: Unlocking LLM discovery on narrative private data,” <https://www.microsoft.com/en-us/research/blog/graphrag-unlocking-llm-discovery-on-narrative-private-data/>, 2024, accessed 3-September-2024.
 - [33] S.-Q. Yan, J.-C. Gu, Y. Zhu, and Z.-H. Ling, “Corrective retrieval augmented generation,” *arXiv preprint arXiv:2401.15884*, 2024.

- [34] Z. Levonian, C. Li, W. Zhu, A. Gade, O. Henkel, M.-E. Postle, and W. Xing, “Retrieval-augmented generation to improve math question-answering: Trade-offs between groundedness and human preference,” *arXiv preprint arXiv:2310.03184*, 2023.
- [35] L. Wang, C. Ma, X. Feng, Z. Zhang, H. Yang, J. Zhang, Z. Chen, J. Tang, X. Chen, Y. Lin *et al.*, “A survey on large language model based autonomous agents,” *Frontiers of Computer Science*, vol. 18, no. 6, pp. 1–26, 2024.
- [36] L. Wang, W. Xu, Y. Lan, Z. Hu, Y. Lan, R. K.-W. Lee, and E.-P. Lim, “Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models,” in *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, A. Rogers, J. Boyd-Graber, and N. Okazaki, Eds. Toronto, Canada: Association for Computational Linguistics, Jul. 2023, pp. 2609–2634. [Online]. Available: <https://aclanthology.org/2023.acl-long.147>
- [37] Y. Shen, K. Song, X. Tan, D. Li, W. Lu, and Y. Zhuang, “Hugginggpt: Solving ai tasks with chatgpt and its friends in hugging face,” in *Advances in Neural Information Processing Systems*, A. Oh, T. Neumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, Eds., vol. 36. Curran Associates, Inc., 2023, pp. 38 154–38 180. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2023/file/77c33e6a367922d003ff102ffb92b658-Paper-Conference.pdf
- [38] J. Zhang, R. Krishna, A. H. Awadallah, and C. Wang, “Ecoassistant: Using llm assistant more affordably and accurately,” *arXiv preprint arXiv:2310.03046*, 2023.
- [39] B. Xu, Z. Peng, B. Lei, S. Mukherjee, Y. Liu, and D. Xu, “Rewoo: Decoupling reasoning from observations for efficient augmented language models,” *arXiv preprint arXiv:2305.18323*, 2023.
- [40] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, D. Zhou *et al.*, “Chain-of-thought prompting elicits reasoning in large language models,” *Advances in Neural Information Processing Systems*, vol. 35, pp. 24 824–24 837, 2022.
- [41] Z. Wang, Z. Liu, Y. Zhang, A. Zhong, L. Fan, L. Wu, and Q. Wen, “Rcagent: Cloud root cause analysis by autonomous agents with tool-augmented large language models,” *arXiv preprint arXiv:2310.16340*, 2023.
- [42] Z. Zhang, Y. Yao, A. Zhang, X. Tang, X. Ma, Z. He, Y. Wang, M. Gerstein, R. Wang, G. Liu *et al.*, “Igniting language intelligence: The hitchhiker’s guide from chain-of-thought reasoning to language agents,” *arXiv preprint arXiv:2311.11797*, 2023.
- [43] S. Yao, D. Yu, J. Zhao, I. Shafran, T. Griffiths, Y. Cao, and K. Narasimhan, “Tree of thoughts: Deliberate problem solving with large language models,” *Advances in Neural Information Processing Systems*, vol. 36, 2024.
- [44] J. Huang, S. S. Gu, L. Hou, Y. Wu, X. Wang, H. Yu, and J. Han, “Large language models can self-improve,” *arXiv preprint arXiv:2210.11610*, 2022.
- [45] N. Shinn, F. Cassano, A. Gopinath, K. Narasimhan, and S. Yao, “Reflexion: language agents with verbal reinforcement learning,” in *Advances in Neural Information Processing Systems*, A. Oh, T. Neumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, Eds., vol. 36. Curran Associates, Inc., 2023, pp. 8634–8652. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2023/file/1b44b878bb782e6954cd888628510e90-Paper-Conference.pdf
- [46] J. Chen, S. Yuan, R. Ye, B. P. Majumder, and K. Richardson, “Put your money where your mouth is: Evaluating strategic planning and execution of llm agents in an auction arena,” *arXiv preprint arXiv:2310.05746*, 2023.
- [47] J. S. Park, J. O’Brien, C. J. Cai, M. R. Morris, P. Liang, and M. S. Bernstein, “Generative agents: Interactive simulacra of human behavior,” in *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, ser. UIST ’23. New York, NY, USA: Association for Computing Machinery, 2023. [Online]. Available: <https://doi.org/10.1145/3586183.3606763>
- [48] W. Yao, S. Heinecke, J. C. Niebles, Z. Liu, Y. Feng, L. Xue, R. Murthy, Z. Chen, J. Zhang, D. Arpit *et al.*, “Retro-former: Retrospective large language agents with policy gradient optimization,” *arXiv preprint arXiv:2308.02151*, 2023.
- [49] C. Qian, X. Cong, C. Yang, W. Chen, Y. Su, J. Xu, Z. Liu, and M. Sun, “Communicative agents for software development,” *arXiv preprint arXiv:2307.07924*, 2023.
- [50] Y. Talebirad and A. Nadiri, “Multi-agent collaboration: Harnessing the power of intelligent llm agents,” *arXiv preprint arXiv:2306.03314*, 2023.
- [51] W. Huang, F. Xia, T. Xiao, H. Chan, J. Liang, P. Florence, A. Zeng, J. Tompson, I. Mordatch, Y. Chebotar *et al.*, “Inner monologue: Embodied reasoning through planning with language models,” *arXiv preprint arXiv:2207.05608*, 2022.

- [52] S. Ma, Q. Chen, X. Wang, C. Zheng, Z. Peng, M. Yin, and X. Ma, “Towards human-ai deliberation: Design and evaluation of llm-empowered deliberative ai for ai-assisted decision-making,” *arXiv preprint arXiv:2403.16812*, 2024.
- [53] Y. Wang, Z. Liu, J. Zhang, W. Yao, S. Heinecke, and P. S. Yu, “Drdt: Dynamic reflection with divergent thinking for llm-based sequential recommendation,” *arXiv preprint arXiv:2312.11336*, 2023.
- [54] S. Hamilton, “Blind judgement: Agent-based supreme court modelling with GPT,” in *The AAAI-23 Workshop on Creative AI Across Modalities*, 2023. [Online]. Available: <https://openreview.net/forum?id=Nx9ajnqG9Rw>
- [55] C.-M. Chan, W. Chen, Y. Su, J. Yu, W. Xue, S. Zhang, J. Fu, and Z. Liu, “Chateval: Towards better LLM-based evaluators through multi-agent debate,” in *The Twelfth International Conference on Learning Representations*, 2024. [Online]. Available: <https://openreview.net/forum?id=FQepisCUWu>
- [56] J. C. Yang, M. Korecki, D. Dailisan, C. I. Hausladen, and D. Helbing, “Llm voting: Human choices and ai collective decision making,” *arXiv preprint arXiv:2402.01766*, 2024.
- [57] J. Li, Q. Zhang, Y. Yu, Q. Fu, and D. Ye, “More agents is all you need,” *arXiv preprint arXiv:2402.05120*, 2024.
- [58] S. Hong, X. Zheng, J. Chen, Y. Cheng, J. Wang, C. Zhang, Z. Wang, S. K. S. Yau, Z. Lin, L. Zhou *et al.*, “Metagpt: Meta programming for multi-agent collaborative framework,” *arXiv preprint arXiv:2308.00352*, 2023.
- [59] X. Tang, A. Zou, Z. Zhang, Y. Zhao, X. Zhang, A. Cohan, and M. Gerstein, “Medagents: Large language models as collaborators for zero-shot medical reasoning,” *arXiv preprint arXiv:2311.10537*, 2023.
- [60] J. Wang, J. Wang, B. Athiwaratkun, C. Zhang, and J. Zou, “Mixture-of-agents enhances large language model capabilities,” *arXiv preprint arXiv:2406.04692*, 2024.
- [61] T. Liang, Z. He, W. Jiao, X. Wang, Y. Wang, R. Wang, Y. Yang, Z. Tu, and S. Shi, “Encouraging divergent thinking in large language models through multi-agent debate,” *arXiv preprint arXiv:2305.19118*, 2023.
- [62] Y. Du, S. Li, A. Torralba, J. B. Tenenbaum, and I. Mordatch, “Improving factuality and reasoning in language models through multiagent debate,” *arXiv preprint arXiv:2305.14325*, 2023.
- [63] H. Chen, W. Ji, L. Xu, and S. Zhao, “Multi-agent consensus seeking via large language models,” *arXiv preprint arXiv:2310.20151*, 2023.
- [64] R. Li, T. Patel, and X. Du, “Prd: Peer rank and discussion improve large language model based evaluations,” *arXiv preprint arXiv:2307.02762*, 2023.
- [65] T. Rebedea, R. Dinu, M. Sreedhar, C. Parisien, and J. Cohen, “Nemo guardrails: A toolkit for controllable and safe llm applications with programmable rails,” *arXiv preprint arXiv:2310.10501*, 2023.
- [66] H. Inan, K. Upasani, J. Chi, R. Rungta, K. Iyer, Y. Mao, M. Tontchev, Q. Hu, B. Fuller, D. Testuggine *et al.*, “Llama guard: Llm-based input-output safeguard for human-ai conversations,” *arXiv preprint arXiv:2312.06674*, 2023.
- [67] J. Ruan, Y. Chen, B. Zhang, Z. Xu, T. Bao, du qing, shi shiwei, H. Mao, X. Zeng, and R. Zhao, “TPTU: Task planning and tool usage of large language model-based AI agents,” in *NeurIPS 2023 Foundation Models for Decision Making Workshop*, 2023. [Online]. Available: <https://openreview.net/forum?id=GrkgKtOjaH>
- [68] G. Wang, Y. Xie, Y. Jiang, A. Mandlikar, C. Xiao, Y. Zhu, L. Fan, and A. Anandkumar, “Voyager: An open-ended embodied agent with large language models,” in *Intrinsically-Motivated and Open-Ended Learning Workshop @NeurIPS2023*, 2023. [Online]. Available: <https://openreview.net/forum?id=nfx5IutEed>
- [69] T. Xie, F. Zhou, Z. Cheng, P. Shi, L. Weng, Y. Liu, T. J. Hua, J. Zhao, Q. Liu, C. Liu *et al.*, “Openagents: An open platform for language agents in the wild,” *arXiv preprint arXiv:2310.10634*, 2023.
- [70] C. Qu, S. Dai, X. Wei, H. Cai, S. Wang, D. Yin, J. Xu, and J.-R. Wen, “Tool learning with large language models: A survey,” *arXiv preprint arXiv:2405.17935*, 2024.
- [71] J. Yang, C. E. Jimenez, A. Wettig, K. Lieret, S. Yao, K. Narasimhan, and O. Press, “Swe-agent: Agent-computer interfaces enable automated software engineering,” *arXiv preprint arXiv:2405.15793*, 2024.
- [72] Y. Cheng, J. Chen, Q. Huang, Z. Xing, X. Xu, and Q. Lu, “Prompt sapper: A llm-empowered production tool for building ai chains,” *ACM Trans. Softw. Eng. Methodol.*, vol. 33, no. 5, jun 2024. [Online]. Available: <https://doi.org/10.1145/3638247>
- [73] M. Shamsujjoha, Q. Lu, D. Zhao, and L. Zhu, “Towards ai-safety-by-design: A taxonomy of runtime guardrails in foundation model based systems,” *arXiv preprint arXiv:2408.02205*, 2024.
- [74] B. Xia, Q. Lu, L. Zhu, and Z. Xing, “Towards ai safety: A taxonomy for ai system evaluation,” *arXiv preprint arXiv:2404.05388*, 2024.

- [75] Q. Lu, L. Zhu, J. Whittle, and X. Xu, *Responsible AI: Best Practices for Creating Trustworthy AI Systems*, 1st ed. Addison-Wesley Professional, 2023.
- [76] D. Zhou, X. Xue, and Z. Zhou, “Sle2: The improved social learning evolution model of cloud manufacturing service ecosystem,” *IEEE Transactions on Industrial Informatics*, vol. 18, no. 12, pp. 9017–9026, 2022.
- [77] Y. Liu, Q. Lu, G. Yu, H.-Y. Paik, H. Perera, and L. Zhu, “A pattern language for blockchain governance,” in *Proceedings of the 27th European Conference on Pattern Languages of Programs*, ser. EuroPLop ’22. New York, NY, USA: Association for Computing Machinery, 2023. [Online]. Available: <https://doi.org/10.1145/3551902.3564802>
- [78] “Information Technology - Artificial Intelligence - Management System,” International Organization for Standardization, Standard ISO/IEC 42001:2023, 2023. [Online]. Available: <https://www.iso.org/standard/81230.html>
- [79] B. Xia, Q. Lu, L. Zhu, S. U. Lee, Y. Liu, and Z. Xing, “From principles to practice: An accountability metrics catalogue for managing ai risks,” *arXiv preprint arXiv:2311.13158*, 2023.