

# KIMAs: A Configurable Knowledge Integrated Multi-Agent System

Zitao Li\*, Fei Wei\*, Yuexiang Xie\*, Dawei Gao, Weirui Kuang, Zhijian Ma, Bingchen Qian,  
Yaliang Li, Bolin Ding

Alibaba Group

## Abstract

Knowledge-intensive conversations supported by large language models (LLMs) have become one of the most popular and helpful applications that can assist people in different aspects. Many current knowledge-intensive applications are centered on retrieval-augmented generation (RAG) techniques. While many open-source RAG frameworks facilitate the development of RAG-based applications, they often fall short in handling practical scenarios complicated by heterogeneous data in topics and formats, conversational context management, and the requirement of low-latency response times. This technical report presents a configurable knowledge integrated multi-agent system, **KIMAs**, to address these challenges. **KIMAs** features a flexible and configurable system for integrating diverse knowledge sources with 1) context management and query rewrite mechanisms to improve retrieval accuracy and multi-turn conversational coherency, 2) efficient knowledge routing and retrieval, 3) simple but effective filter and reference generation mechanisms, and 4) optimized parallelizable multi-agent pipeline execution. Our work provides a scalable framework for advancing the deployment of LLMs in real-world settings. To show how **KIMAs** can help developers build knowledge-intensive applications with different scales and emphases, we demonstrate how we configure the system to three applications already running in practice with reliable performance.<sup>1</sup>

## 1 Introduction

Large language models (LLMs) have had a profound impact on various aspects of people’s lives, particularly as the foundational technology behind conversational applications such as chatbots. These models have become indispensable as virtual assistants, offering powerful capabilities for various tasks, including addressing common-sense queries, generating summaries for academic papers [16], and solving programming challenges and tasks [11]. Despite their impressive functionality, LLMs are of some limitations. Issues such as hallucinations and the inability to provide the most up-to-date information or private knowledge hinder their reliability in directly serving for knowledge-intensive applications. These shortcomings can be mitigated by integrating LLMs with external information in the input context [20, 28]. One notable approach is retrieval-augmented generation (RAG) techniques [1, 10], which enhances LLMs by equipping them with retrieval capabilities, allows LLMs to address questions that exceed the scope of their pre-trained internal knowledge. RAG has proven highly effective in improving performance on question-answering (QA) tasks emphasizing faithfulness to truths, showcasing its potential to bridge the gap between static pre-trained knowledge and dynamic, context-specific information.

While many real-world applications have adopted RAG techniques [13, 22], open-source frameworks have also emerged to facilitate the adaptation of RAG to a wide range of tasks [14, 18] for the public to hold RAG application services themselves with local data. While these open-source RAG frameworks provide convenient starting points for building RAG-based applications, there remain significant opportunities for improvement, especially in more practical and complicated scenarios, e.g., efficient multi-source knowledge retrieval, which provides primary motivations for this paper.

**Challenge 1.** While developing a basic chatbot using LLM APIs is relatively straightforward, the complexity increases significantly when the conversation requires intensive external knowledge. A user’s question may

\*Co-first authors.

<sup>1</sup>The source code is under review and will be available soon.

contain only partial information, while the rest is in the conversation history, such as containing pronouns referring to an object in the previous question. Using such a question with unclear pronouns as a query to retrieve knowledge will significantly limit knowledge retrieval accuracy, and answering such unclear questions may increase the risk of hallucination. Another common case is that when the application is built in a specific knowledge context (e.g., QA based on a GitHub repository), the user may tend to omit some meaningful keywords in their questions, without which the LLMs may misuse its internal knowledge to generate improper answers. Therefore, such applications require mechanisms to clarify and detail the user questions into queries that contain sufficient information for retrieval knowledge and final answer generation.

**Challenge 2.** Unlike commercial products [13, 22], building a complete powerful data collection and preprocessing pipeline may be too heavy for tasks relying on private knowledge. Therefore, a lightweight and configurable retrieval mechanism may be a more desirable solution for developers to utilize their local knowledge from different sources with heterogeneous topics and formats. Besides, when it comes to running time, accurately answering questions with multi-source knowledge also relies on correctly using a subset of knowledge. However, there is no existing solution for automatically and efficiently handling the knowledge selection while providing flexibility for taking human intervention.

**Challenge 3.** Moreover, as a QA application, the final answer generation is arguably the most critical step. Retrieved content from multiple knowledge sources may easily consume LLM’s effective context window. To maximize the answer performance, correctly identifying the useful retrieved content and organizing them for question answering are the keys. But it may be hard to tell how one retrieved knowledge piece from a source is more helpful than the other. Moreover, for the user experience, providing references is believed to be an effective operation to increase the trust level of the generated answer. However, it is unclear how to achieve it efficiently with minimum costs.

**Challenge 4.** Finally, in addition to correctly answering questions, users may expect the system’s response time (i.e., latency) to be as short as possible. Considering the system’s needs for necessary query processing, handling multi-source knowledge retrieval, and answer generation, optimization of execution efficiency requires careful design.

In this paper, we present the design and implementation of an open-source solution, **KIMAs**, based on AgentScope [5] framework. The solution is developed with the following key properties to address the challenges:

1. *Context management and query rewrite mechanisms to enhance retrieval accuracy and conversation coherency.* **KIMAs** provides a built-in *conversation context* query enrichment mechanism that can fill the semantic gaps in the user’s current question based on a long conversation history. A set of configurable *knowledge context* query rewrite mechanisms is provided for the developers to further adjust the query to fully unleash the power of the retrieval mechanisms and match the most desirable knowledge.
2. *Efficient routing and retrieval with heterogeneous knowledge sources.* **KIMAs** supports various knowledge sources with different data topics. **KIMAs** is also equipped with an efficient routing mechanism that can ensure when a query comes, only the most appropriate knowledge sources will be used to minimize the overall cost while ensuring the answer quality. The routing mechanism is also configurable in the sense that developers can provide extra manually written mix-in or scale the score to bias the selection according to their preference and actual use cases.
3. *Simple yet effective answer and reference generation.* We provide a filtering mechanism that is compatible with content from different knowledge sources so that the LLMs can effectively use their window size to generate the final answer. Recognizing that providing citations and references is critical for ensuring trust and transparency in knowledge-intensive applications, **KIMAs** also implements a straightforward yet reliable strategy for generating citations without requiring model training or introducing additional latency for the central answer generation.
4. *Configurable and low-latency multi-agent pipeline.* **KIMAs** employs a multi-agent pipeline architecture. While the pipeline is configurable with three different types of agents, we provide an optimized configuration with parallelization to maximize efficiency and resource utilization.

In summary, this paper introduces **KIMAs**, an open-source framework designed to address the challenges of integrating RAG techniques into real-world applications. By enabling versatile query enhancement mechanisms

with conversation and knowledge context, handling heterogeneous knowledge sources, and ensuring reliable citation generation, KIMAs seeks to improve the utility and trustworthiness of RAG-based systems. Through the practical use cases of KIMAs, we demonstrate its robustness and adaptability to building effective QA chatbots and other RAG-driven applications, advancing the state of the art in this rapidly evolving domain.

## 2 Preliminary

### 2.1 LLM-based Multi-Agent System

**Agent.** In this paper, “agent”, abbreviated from “LLM-based agent”, is usually characterized as a paradigm of LLM-centric applications that employ LLM to mimic human brain [31]. With the reasoning capability of LLMs [17, 19, 29, 35], an agent can decompose a complex task into subtasks that can be solved more reliably; meanwhile, LLM can make decisions so that an agent can dynamically decide when and how to use tools (i.e., APIs of different external functionalities, such as query portal of current weather of a city) given a task [21, 24, 36]; finally, with a memory for maintaining context and related knowledge [8, 26], agents can generate appropriate answers in long conversations or utilize external knowledge. In this paper, we relax the definition of an agent so that it may contain only a subset of these three elements.

**Multi-agent and pipeline.** At the current stage, an agent’s performance is not satisfying given complicated tasks. For example, end-to-end code generation [9] or have a large number of tools for selection. Multi-agent systems represent an emerging paradigm, where multiple agents, each specialized in some specific tasks, collaborate to solve complex tasks. Some of the multi-agent frameworks are conversational [15, 33], where multiple agents collaborate and communicate with each other by generating messages in natural language generated by LLM. Some other multi-agent frameworks [9] emphasize the diverse capabilities of LLMs to break down tasks into modular components, allowing individual agents to specialize in specific roles such as information retrieval, reasoning, or decision-making. By communicating and exchanging intermediate outputs, agents collectively achieve goals that exceed the capabilities of a single LLM operating in isolation. There are also many multi-agent systems that are designed for specific tasks, including medical [27], coding [34], and evaluation [2]. Although the above work may have implementation differences, the main idea is still to decompose complex tasks into simple subtasks and let each agent work as an information processing node in a *pipeline*.

### 2.2 Knowledge-intensive QA

RAG is one of the most popular techniques coupled with LLMs, designed to address knowledge-intensive tasks, particularly those requiring high-confidence answers or involving private knowledge unavailable during the model’s training phase. In the early stages, when language models lacked strong in-context learning capabilities, RAG techniques primarily relied on training or fine-tuning models to integrate retrieved knowledge [1, 10]. However, with the rapid advancement of language models and their demonstrated ability to perform in-context learning, a more efficient and cost-effective approach has emerged. This strategy involves appending retrieved text chunks as additional prompts to the LLM input, avoiding the need for task-specific fine-tuning [23]. In parallel, significant research has been dedicated to improving retrieval mechanisms to ensure higher-quality inputs, such as dense retrieval methods that enhance the relevance of retrieved text [12]. These developments highlight the ongoing evolution of RAG techniques and their critical role in extending the capabilities of LLMs for real-world applications.

## 3 Knowledge-oriented QA System Designs in KIMAs

KIMAs focuses on the scenario where application users expect to obtain accurate answers based on knowledge from multiple homogeneous or heterogeneous sources. We use the following hypothetical use case to demonstrate the challenges of building knowledge-oriented QA applications in practice. The real use cases are presented in the following Section 4.

*Hypothetical use case.* If a developer of a GitHub repository wants to build an LLM-based QA plugin based on his repository, he may need to consider the following potential questions:

1. Technical questions that LLMs can answer if enough locally available knowledge is provided, such as the code snippet, API documents or tutorials available in this GitHub repository;
2. Questions related to this GitHub but requiring knowledge beyond the locally-host one, such as whether there are any existing applications built based on this GitHub repository but implemented by a third party, whether there is any third-party solution for some issue when using this repository, or whether there are any third-party comments about this repository.

Inspired by this hypothetical use case, we identify the following problems in building a knowledge-intensive QA system for practical use cases. Our system design focuses mainly on solving these problems.

**(1) Technical questions may not always be formulated clearly.** There exists a gap between many research-oriented Retrieval-Augmented Generation (RAG) benchmarks and practical scenarios. When application users encounter technical issues, they often struggle to articulate their questions accurately, particularly in terms of using the correct terminology, for example, the meaning of an input parameter of a function in the context of the specific GitHub repository in the hypothetical use case above. Application users may require additional "warm-up" conversations to clarify and refine their inquiries in these cases. The missed information is expected to be filled in by understanding the conversation and knowledge contexts for better retrieval and QA performance.

**(2) Different questions may prefer different knowledge sources.** Consider the first type of question in the hypothetical use case above. A pipeline built with processed GitHub repository data stored in a local vector database can be an economical and flexible solution that can solve most of the questions. For the second type of question, collecting information through some online search APIs can provide LLMs with more comprehensive information to generate reliable and up-to-date answers. In addition, different knowledge sources may also require different query preprocessing techniques and information post-processing operations. For example, when using online search engines, it may be more important to propose a set of correct keywords based on the complete sentences in natural language, but complete but more organized sentences may be preferred when using embedding similarity-based search in vector databases.

**(3) Irrelevant information need to be filtered and adopted information need to be cited explicitly.** The retrieval mechanism cannot always guarantee that all the retrieved content is helpful for answering the question. The useless information need to be filtered out before being fed into the LLMs as the LLMs have limited effective context windows to process information. On the other hand, existing LLMs can still generate hallucinated messages or cannot generate perfect answers for some complicated questions in conversations (e.g., code generation with very specific requirements). As many commercial closed-source solutions (e.g., Perplexity[22]), providing information sources (e.g., links to GitHub files or websites) has been proven as a common way to accommodate such limitations, as the application users can further refer to those provided links for verification or proceeding their difficult task with ground-truth knowledge support.

**(4) Application users desire similar response latency as simple RAG solutions.** Another critical metric that influences user experience is response latency. Application users generally expect LLM applications to deliver responses with consistent speed, regardless of the complexity of the backend information processing. However, a significant portion of RAG application developers use only the available LLM APIs and cannot control how model inference optimization works. This expectation introduces a significant challenge in scenarios where multi-stage information processing is required. Despite the inherent complexity of such systems, maintaining efficiency remains a crucial metric that must be considered.

### 3.1 Modules and System Structure Overview

**Agentive Modularization.** In our design, three modules serve as cores in KIMAs: conversation context management, information retrieval, and final answer generation. As in Figure 1, we design a specialized agent class for each module: context manager, retrieval agent, and summarizer.

- *Context manager:* In everyday communication, conversations often rely heavily on contextual information. The same is true in knowledge-intensive conversational QA tasks. In general, the context manager is designed to enrich the query by extracting missing information from the conversation context, ensuring its completeness. For instance, when KIMAs is used to provide QA services for the AgentScope [5] GitHub repository, a user might query whether there is any multi-agent games application in the

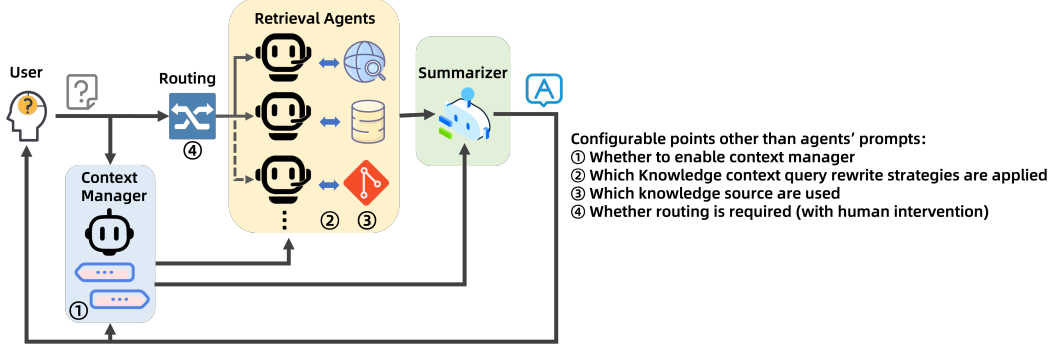


Figure 1: KIMAs system with agentic modularization and configurable pipeline.

repository. An initial response can be “Yes. For example, there is a simulation for the game Werewolf that...” After that, the user may ask a follow-up question, “Where can I find the code for it?” In this case, the context manager must rely on context to determine that “it” refers to the Werewolf game in the AgentScope repository and enrich the user question for the following operations.

- *Retrieval agents*: Each retrieval agent has its own information source(s) (e.g., similarity search with a local vector database or used by an online search engine). One of the primary tasks of these agents is to further revise the query based on the agent’s specific knowledge context and retrieve relative information. For example, when the retrieval agent has access to an online search engine, it rewrites the query into a set of keywords to align the search mechanism. In contrast, when a retrieval agent relies primarily on a vector database with dense and sparse retrieval, a sentence enriched with conversation context and task background can improve the matching process during retrieval.
- *Summarizer*: The summarizer is designed to finalize the answer to a query by integrating information from the context manager and retrieval agents. The goal of the summarizer is to generate content that is both faithful to the information provided by the retrieval agents (i.e., faithful) and appropriate within the context of the conversation (i.e., helpful).

**Configurable end-to-end pipeline.** While three types of built-in agents are provided and their system prompt can be adjusted according to the application, there is still large room for configuration for different kinds of knowledge-intensive QA applications. The following points are exposed for configuration by developers.

- *Whether to enable context manager.* While the context manager can enrich the user query and provide necessary context information for the final answer generation, analyzing and processing the context will take extra time for LLM. In some cases, where extremely low latency is required, the context manager can be disabled to further reduce latency. Instead, one can directly provide the entire conversation history for the summary and ask it to generate the final answer directly. Of course, if the context manager is disabled, the quality of the final answer may be hard to control if the conversation is complicated or the LLM is weaker.
- *Which knowledge context query rewrite strategies are applied.* As discussed above, query rewrite can be a critical point for a knowledge-intensive application, helping to retrieve more accurate content from knowledge sources. KIMAs provides interfaces in the configuration with multiple build-in rewrite strategies (and rewrite prompts of some strategies) for the developers. More details will be discussed in Section 3.2.
- *Which knowledge sources are used.* We provide built-in vector database and online search engine APIs as the two built-in knowledge sources for the application. For locally hosted knowledge, KIMAs inherits LlamaIndex [18] functions and is integrated into our system so that developers can switch different functions by just changing their knowledge configuration files. Details are discussed in Section 3.3.

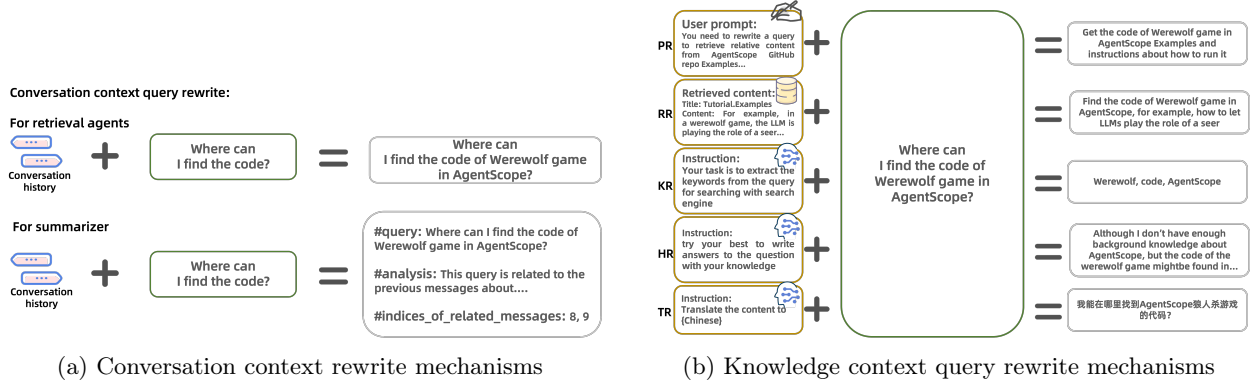


Figure 2: Query rewrite mechanisms in KIMAs.

- *Whether routing is required (with human intervention).* We also prove some flexibility for the routing mechanism. Besides the built-in mechanism routing, we allow developers to cast their human preferences for the routing by providing additional manually written mix-in text or scaling the importance of some weights. More details are deferred to Section 3.3.

With these key agents and configurable pipeline design working together, it becomes KIMAs.

### 3.2 Context-based Query Enhancing

Knowledge-intensive QA applications usually have two kinds of “contexts” as essential factors to answer a question: the context of conversation and the context of knowledge sources. Correspondingly, two types of agents in our design can rewrite the user query to enrich its semantic meaning with different information sources: context manager and retrieval agent. In general, the context manager, designed to help the retrieval agent and summarize digesting conversation in advance, ensures that the query contains necessary information from the conversation context; on the other hand, a retrieval agent is supposed to rewrite the query better to fit the retrieval mechanism in its knowledge source context. While the rewrite mechanisms of context manager are fixed, the ones for retrieval agents are designed to be more flexible. The following are details about context-based query enhancement.

**Context manager: conversation-contextual query rewrite for retrieval agents.** As discussed in Section 3.1, understanding the user’s real intention behind a query heavily depends on the context of the conversation. Without some keywords in the conversation context, knowledge retrieval can be pointless. Thus, the goal of the query rewrite mechanism at this stage is to enrich the query with precise conversation context information. 1) The first point to consider is that some key information is lost in the query but can be obtained from the conversation history (e.g., the example mentioned in Section 3.1). The context manager checks whether the query itself is ambiguous, including containing pronouns referring to terms appearing in the previous conversation. 2) The second consideration is that some users may rephrase their previous questions when they find that the answer provided by KIMAs is not satisfactory. Therefore, the context manager also needs to revise the query with reflection. 3) Besides, this step is expected not to consume too much time as one of the intermediate steps in the pipeline. Therefore, these two goals are integrated into the LLM prompt together with the conversation history and the LLM is expected to properly rewrite the question with necessary details filled and emphasized. Considering the time constraint, we also recommend using a lightweight LLM to handle this task.

**Context manager: conversation-contextual query rewrite for summarizer.** The final step in generating the answer is another step that is heavily based on the context information. In this step, the quality of the generated answer depends directly on the degree to which the LLM understands the relevant context. Our observations indicate that while many LLMs perform well for requests with simple requirements,

but they may struggle to handle more complex requests involving multiple sub-tasks (e.g., understanding the conversation and answering the current question with given context in one generation). Besides, when the context is long (for example, a long answer is provided to the previous query), including the entire conversation history in the final step can unnecessarily consume the LLM’s effective context length, reducing efficiency and potentially affecting output quality. Nevertheless, the final answer generation requires more detailed information to generate the final answer, which may not always aligned with the rewrite goal for retrieval agents. To address these challenges, a second function of the context manager is to perform a detailed and reliable analysis of the conversation context, and only the digested conversation context will be passed to the summarizer for final answer generation. This ensures that only the most relevant information is retained, improving the overall quality and efficiency of the final response generation.

More precisely, we prompt the context manager to generate answers with two fields: **analysis** and **indices\_of\_related\_messages**. Generating **analysis** for the relation between the history and the current query can be considered with a similar effect to the Chain of Thoughts [30], which enables LLMs to generate answers more rationally. After the **analysis**, LLMs should have higher confidence about which previous historical conversation piece is indeed related to the current one. To provide more precise information for summarization, we also prompt the LLM to generate **indices\_of\_related\_messages**, with which we can directly extract the related messages by the indices.

**Retrieval agent: knowledge-contextual query rewrite.** In KIMAs, different retrieval agents have different knowledge sources with potentially heterogeneous data types. In order to maximize the accuracy of information retrieval, different query rewrite prompts or even query rewrite algorithms can be employed at the agent level.

The built-in rewriting strategies are listed as follows and illustrated in Figure 2b.

- **Prompt rewrite (PR):** Rewrite the query following the instructions of the prompt. This mechanism allows developers to customize the prompt to LLM inject necessary information to the query, typically prior knowledge or understanding of the knowledge.
- **Retrieval rewrite (RR):** Rewrite the query with some knowledge content retrieved with the original query. An intuition of this method is that LLMs can help revise more accurate queries when providing some knowledge context.
- **Keyword rewrite (KR):** Extract only keywords from the original query for search engines. When using online search engine APIs, keywords are usually the best way to perform a search, as the redundant information in the original query may diverge the search to unrelated content.
- **HyDE rewrite [6] (HR):** Generate a paragraph with LLM’s internal knowledge to answer the query, then take the paragraph as the new query. It is shown that in some cases, the embeddings of the LLM generated response without the interference of external knowledge can be more similar than the one of the raw query to the desired embeddings of the desired knowledge, so that it can improve the retrieval performance.
- **Translation rewrite (TR):** Rewrite the query in the same language as the one specified in the configuration of the knowledge source. For the cases where the language in the knowledge source is different from the one of the query, it is believed that maybe mapping it first to the targeted language can provide a more accurate retrieval result.

All the above rewrite strategies are available and only require changing configuration files. However, developers can provide their own rewrite strategies and easily integrate them into their applications.

### 3.3 Efficient Multi-source Information Retrieval

#### 3.3.1 Knowledge retrieval mechanisms

The retrieval agents play key roles in KIMAs because they have access to knowledge sources. Their responsibility is to retrieve and provide relative knowledge to user queries. The following built-in support for the different knowledge sources can be easily assigned to retrieval agents via configuration.

- *Local knowledge stored in vector databases (VDB).* Using local vector databases is the most popular and efficient method to construct local knowledge bases, especially after LLMs demonstrate their in-context learning capability. The embeddings of the knowledge in VDB only need to be computed once and used to construct the indexing. At inference time, it only requires the embedding model to encode a query into a vector for a similarity match. The storage of local knowledge and retrieval from local VDB has a few advantages. The first advantage is that it can utilize local knowledge, which is available locally, with little data privacy or data sovereignty concerns. A second advantage is that VDBs usually support a mixture of dense (embedding similarity) and sparse (BM25) retrieval, which can provide superior performance when semantic similarity is the only metric for retrieval. A third advantage is that even if the embedding relies on API, generating embedding for a query is usually much more affordable (even if it can be done locally) than calling the API of search engines. In KIMAs, users can choose to use LlamaIndex [18] built-in in-memory VDB or Elasticsearch [4].
- *Online search engines.* There are many advantages of using online search engines as an information source. One is that it can guarantee the information is up-to-date. A second advantage is that the search engine algorithms rank the returned results, which consider many different factors (e.g., timeliness, popularity and authority with Internet-scale information). Therefore, even if it is more expensive than the local VDB method, online search engines are still one of the popular knowledge sources in many applications. The built-in support in KIMAs for search engine is built on Bing search [25], but developers can easily switch to Google search and others.
- *Domain specific HTTP API.* Some websites provide in-site search APIs, which can return some domain-specific knowledge presented on the website but are not open to web crawlers. To leverage such APIs, we provide some flexible request-response parsing functions that facilitate the calling step and response parsing steps.

**Discussion: raw text information or LLM digested information?** Determining whether the retrieval agent should process the retrieved information before feeding it into the summarizer is tricky. The answer should be given case by case, depending on many different factors. For example: 1) Can the LLM used by the summarizer support processing long context information and still provide reasonably good reasoning capability? 2) Do we expect that the final answer of KIMAs can be generated efficiently (e.g., seeing the first token within ten seconds)? If the answers to both questions are “yes”, then letting the retrieval agents return the raw text and letting the summarizer process the raw information directly may be more appropriate because it can avoid additional latency because of LLM processing. Besides, such a more straightforward approach can reduce the chance of introducing LLM hallucinations. However, suppose only context-length-limited LLMs are available, or the available LLMs cannot provide acceptable performance in long-context reasoning; it may be a better solution to let each retrieval agent process the raw retrieved information, extract the key information first, and only pass the valuable information to the final answer generation step.

### 3.3.2 Embedding clustering routing

We expect each retrieval agent to take charge of one or a few knowledge sources. The key guidance is that one can group similar knowledge sources to the same retrieval agent so that the knowledge context query rewrite can benefit the retrieval of all similar knowledge sources. On the other hand, knowledge sources with very different topics or contents are supposed to be assigned to different retrieval agents.

However, a challenge is correctly selecting the best-fit retrieval agent(s) with the most relevant knowledge source. Most existing multi-agent routing mechanisms rely on 1) manually created descriptions for the (functionalities of) agents and 2) using LLMs as decision-makers to decide which agents should be activated to provide knowledge. However, such a combination is not suitable for routing between knowledge retrieval agents. As a knowledge source may contain a large volume of knowledge, it can easily exceed the context length of any LLMs; nevertheless, it is challenging for human beings to summarize all the knowledge from a source comprehensively. Meanwhile, using LLMs to select the knowledge source may not be a good idea in practice because 1) its output can be non-negligible randomness, which can make it hard to ensure consistency; 2) LLMs inference can be slow and restricted by the context length (i.e., not suitable for high-efficiency applications or context with long conversation).



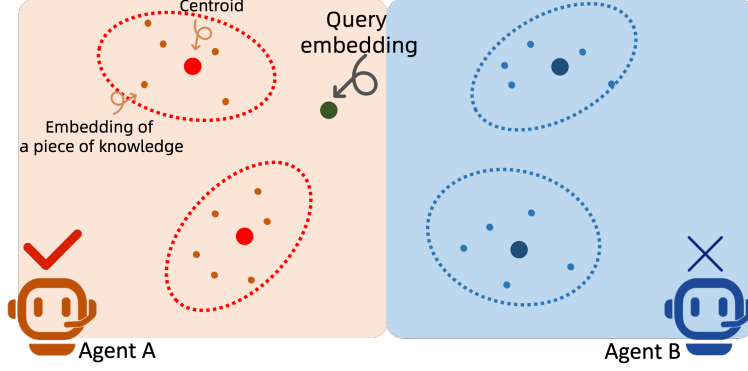


Figure 3: A simple visualization of the routing mechanism. Because the query embedding is closer to centroids in Agent A’s knowledge domain, Agent A is roused to conduct knowledge retrieval.

**Backbone routing mechanism.** We adopt an algorithm similar to [32]. Figure 3 gives a simple visualization of the core idea. The key idea is to utilize the embeddings of the knowledge in each agent. In addition to being used in retrieval, the embeddings of the knowledge (e.g., text chunks) are also perfect representations of that knowledge. Therefore, the embeddings of chunks of knowledge are first clustered in each agent, and the centroids of the cluster are considered to be *synopses* of the knowledge. When routing, an embedding of the query needs to be used for similarity search with the centroids of all retrieval agents. Only the agents with top- $K$  similar centroids to the query embedding are activated for exact knowledge retrieval.

**Manual mix-in.** Different from [32], we also need to consider retrieval agents using online search APIs as the knowledge source, which has limited or even no local knowledge to guide the selection of knowledge. In addition, the method of in [32] is completely data-driven. While complete data-driven is a desired feature, it also means difficult to impose human preferences when it comes to real practice.

To resolve such inconveniences, we allow developers to either 1) provide the initial description or 2) provide a set of typical knowledge chunks (e.g., QA pairs or chunks of plain text) as mix-in to enrich the routing. Such manual descriptions are also encoded into embeddings. When routing, the similarity considered becomes a weighted average of the similarity score of the local knowledge pieces to the query and one of the manual mix-ins to the query. When developers have higher confidence in their manual description and want it to dominate the process of the selection, a higher weight can be assigned to the similarity score of the manual description; if the developer wants to save effort and have plenty of coherent knowledge in each agent’s knowledge base, a higher weight can be assigned to the similarity score of the local knowledge pieces. If the knowledge source is not local, the manual description can serve as the only centroid of the knowledge.

**Score scaling.** In practice, the similarity scores when comparing the same query with different types of documents can vary in various ranges. For example, when matching a natural language query with Python code knowledge, it usually has lower scores than matching with natural language documents, even if the key relative information related to the query is indeed in the Python code. To handle such cases, we employ a simple but effective strategy that allows users to scale some similarity score related to specific types of knowledge, either up or down.

### 3.4 Summarization

**Reranking for filtering.** A common strategy to avoid the false-negative cases (i.e., high-relevance knowledge is not retrieved) is to retrieve slightly overwhelming information that is larger than the context length of the LLMs. In KIMAs, since multiple retrieval agents can provide different knowledge sources, the summarizer can receive overwhelming information that cannot fit into the effective context window of LLMs. However, although embedding similarity matching mechanisms can efficiently retrieve a lot of relevant information, they are not metrics to rank or filter information due to the following important issues. 1) Some

retrieval agents may even never compute the similarity score, such as those using the online search engine as the knowledge source. 2) High similarity scores can contain false positive signals as they are computed in compressed vector space. 3) The similarity scores from different resources may not be comparable because the retrieval agents may rewrite the query for some purpose, so the similarity scores used in retrieval are actually based on different queries. 4) The nature of knowledge can affect the outcome, such as the data format, chunk size, etc.

Reciprocal rank fusion (RRF) [3] can be a model-free statistic that may help in reranking and filtering. However, it is unlikely to have duplicated knowledge pieces retrieved from different knowledge sources. Therefore, RRF may not be that useful for reranking in multi-source knowledge retrieval. Instead, we use the reranking model to sort the raw retrieved fragments. Although a reranking model can introduce additional computation cost, it is a more reliable and general method for our tasks.

**Citation generation.** Citations and references can provide additional confidence for the generated content as users can verify the answer by looking at the references provided. However, generating citations is a challenging task for LLMs [7]. In KIMAs, our goal is to generate citations efficiently without training a specific model or introducing a significant regression in latency or reasoning performance for the final answer.

We tested several approaches. Our initial is *one-step*, with which the LLMs are prompted to finish the following tasks in a single answer: 1) analyze which chunks can help answer the query question; 2) generate the answer to the query question; 3) select the indices of the related chunks. Then, the final answer is to assemble the generated answer and extract the reference with the indices. However, such an approach has several limitations. First, the LLM must have strong reasoning and context-processing ability. Second, the LLM must generate structure output (e.g. in JSON format) from which different information can be extracted, which means that it is impossible to use the stream mode of the LLMs to provide a low-latency experience for users and introduce additional task failure risk as LLMs cannot always correctly format its output. Third, as the retrieved knowledge is usually in chunks, they can be from different or the same information source (e.g., the same paper). Therefore, it requires additional effort to handle or distinguish duplications.

*Solution: A look-back approach.* To bypass the above problems, we design a look-back strategy for citation generation. The entire generation utilizes the stream mode provided by many LLM APIs or local inference frameworks and consists of two stages. The first stage is to prompt the LLM to generate an answer to the query question based on the retrieved knowledge. The answer is presented directly to the users. The second stage is to provide the generated answer together with the retrieved knowledge to the LLM and let it output the reference of the knowledge used to generate answers. Such a strategy is robust with weaker requirements on the reasoning capability of LLMs. Besides, as the content is generated in stream mode, the user’s experience can be significantly improved as the first token they observed at the same time it is generated, and only a small pause will be observed after the answer is generated and waiting for the citation generation.

### 3.5 Optimizing Pipeline for Efficient Execution

Many LLM-based applications seek low response latency, which poses a challenge for multi-stage applications like KIMAs. More specifically, since multiple sub-tasks exist, LLM will inevitably be used to generate multiple times and introduce some intermediate result waiting time. Therefore, efficiently arranging the execution requires parallelizing as much LLM usage as possible. In order to optimize efficiency, the execution of KIMAs can be executed in parallel with requests for asynchronous model APIs as follows.

- **Parallelization 1: Query ingest.** At this stage, two functions are executed in parallel. One is the *context manager conversation context query rewrite for retrieval agents*, which fills in missing key information for the current query question based on the whole conversation history so that the following retrieval stage can use more accurate information. The other is *query routing*, deciding which retrieval agents are the correct ones to execute.
- **Parallelization 2: Knowledge retrieval and context analysis.** At this stage, each retrieval agent obtains the query enriched with the necessary context information of the conversation. At this stage, each agent can rewrite a knowledge-context query to match the knowledge context and retrieve knowledge. Meanwhile, the context manager analyzes the context of the previous conversation

Use case	With context manager	Routing / Preference adjust	Offline knowledge source(s)	Online knowledge source(s)
AgentScope QA	Yes	ON / No	AgentScope tutorial, code, examples, API docs and FQA set	-
ModelScope QA	Yes	ON / Yes	ModelScope tutorials, and 5 GitHub repos	ModelScope official article, models, datasets,
Olympic Bot	No	OFF/-	-	Olympic events

Table 1: Use Case Configurations

and generates distilled information to ensure the final answer fits the conversation. All of the above agent operations are performed in parallel. At the end of this stage, the retrieval information and the conversation context analysis results are shared with the summarizer for the final response generation.

## 4 Use Cases

**Sytem implementation.** KIMAs is implemented based on a multi-agent framework, AgentScope [5]. At the agent level, the agents inherits from the built-in agents in AgentScope but with extra features specialized for KIMAs, such as the methods supporting the efficient routing mechanism and some external knowledge management and retrieval functions. At the pipeline level, the agents receive input and pass process information via AgentScope’s message objects; some of the parallel execution stages in the pipeline are tailored specifically in KIMAs for easy management and code management.

In the following, we demonstrate how we configure KIMAs to build different QA applications for three different tasks.

### 4.1 Small scale application: AgentScope QA

As a proof-of-concept example, we first present an example with offline knowledge sources with different data formats and topics as a starting point to demonstrate how KIMAs works.

**Goals.** In this use case, we adapt KIMAs to help answer questions about AgentScope’s GitHub repository. The expectation for this application is to serve as a chatbot in a Q&A group for developers building their multi-agent application with AgentScope framework, providing an accurate response in time to the raised questions. It is observed that the most common questions fall in the following categories:

- *Preliminary project questions.* As the Q&A group is open for everyone, some potential users of AgentScope are also presented in the group. Their questions are usually about the feature of AgentScope, its advantages compared with other similar open-source frameworks, and the feasibility of using AgentScope for their task. Once some of them decide to initiate their project with AgentScope, they may be looking for whether there are already demonstration examples for tasks similar to theirs provided in the GitHub repository for reference.
- *Coding or debugging related questions.* Another majority of questions in the Q&A group appear to be developers using AgentScope but encounter issues in their development process. For example, they may seek clarifications of two different agent classes or help to solve the execution errors in their AgentScope-based applications.

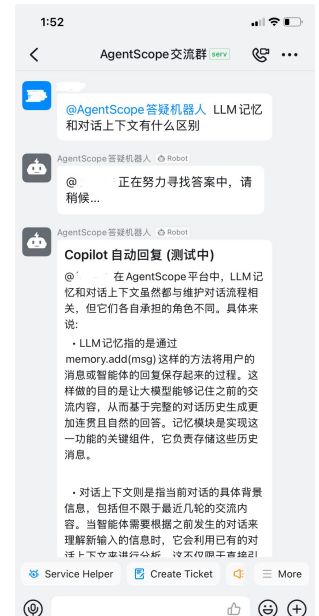


Figure 4: Use case in Q&A group of AgentScope

**Application Configuration.** To satisfy the above goals and provide reliable answers to the questions in Q&A group, we configure our KIMAs in terms of knowledge and execution as the following.

- *Knowledge sources configuration.* The knowledge bases equipped to the agents in this use case span the content related to the core functions of library: the tutorial (Markdown files), code (mainly Python code), API documents (processed into text files). The examples in the AgentScope library, which serve as good demonstration examples for beginners, are also included as knowledge of how to use the AgentScope framework with code and description in Markdown format. Besides, some of the frequently asked questions beyond the scope of the repository (e.g., comparison with other frameworks) are gathered manually, and are summarized and filled with appropriate answers to form a FQA set as an additional knowledge source to the repository. All of this knowledge is processed (i.e., chunking and generating embeddings) and stored in a vector database for query by a knowledge configuration file. Each type of knowledge is assigned to a retrieval agent.

With the routing mechanism, the preliminary project questions are usually answered with the knowledge shared by the agents charging tutorial, examples, and FQA set, while answering the coding or debugging questions will depend on information from tutorial, code, and examples.

- *Pipeline.* This application is configured with all three types of agents activated: context manager, retrieval agent, and summarizer. The context manager is used to help understand the real intention of the user in a conversation context. The retrieval agents are configured with the **Prompt rewrite** module with prompts design for each agent according to their equipped knowledge.

Figure 4 is a screenshot of a real QA with KIMAs AgentScope QA application in the DingTalk Q&A group.

## 4.2 Larger scale: ModelScope QA

**Goal.** While serving as a Q&A chatbot similar to the AgentScope use case, the spectrum of questions to be handled is significantly larger than the ones in Agentscope. Modelscope community is a platform of open-sourced machine learning models, datasets, training/finetuning libraries and applications built with LLMs and other models. It is expected a chatbot to provide an accurate initial answers potentially based on all these kinds of the knowledges from different sources.

**Application Configuration.** To helpfully serve the users in the community, we highlight the specialties in configure as the following.

- *Knowledge sources configuration.* The knowledge sources used in these applications can be categorized into two types, online and offline.

The online knowledge sources include model and dataset information, official articles about the latest open-source community technical progress. These knowledge sources are achieved by Bing search API with different constraints, i.e., restricted to domain of available models/datasets/articles on the website. It is configured to use a commercial search engine instead of in-site search because the results of Bing can also be used to provide related knowledge that ranked by more sophisticated mechanisms beyond text similarity, such as recommending the most popular Text-to-Image models.

The offline knowledge sources include the tutorial documents and eight different GitHub repositories affiliated with ModelScope. The tutorial covers knowledge about how to use models, datasets and computation resources available on modelscope.cn, and the repositories contain code files and repository-level instructions. Compared with the online knowledge sources, these knowledge sources can be hosted and retrieved locally because the retrieval standard is more

- *Pipeline configuration.* Generally speaking, the pipeline configuration is similar to the AgentScope use case, involving all three kinds of agents. But key difference is in the routine mechanism configuration. We provide some manual description mix-in for the routing mechanism to bias some selections manually. For example, both the tutorial and model knowledge source contains the information of some models (or models with similar names); however, the information from model knowledge source is more up-to-date than the one from the tutorial. Therefore, we add some manual mix-in to bias the "model



Figure 5: Three demonstration QA pairs using different knowledge resources in ModelScope QA.

recommendation" type question to use model knowledge source rather than tutorial. Because some knowledge sources are considered more reliable and official (i.e., official articles and tutorial), we set a scaling factor greater than 1 to make the information pieces from those sources have a higher chance of being selected.

Figure 5 shows three QA pairs from the ModelScope QA using different knowledge sources.

### 4.3 Turbo Scale: Olympic Bot on Weibo

**Goal.** During the Paris 2024 Olympic Games, KIMAs serves as the core of the back-end algorithm to generate auto-comments for the posts related to the Olympics<sup>2</sup>. The posts and comments of this Weibo bot are supposed to focus only on Olympic Games, including the news and historic Olympic events. However, there are many other bots performing on Weibo, but there is a restriction that no more than two bots can reply to the same Weibo post. Therefore, there is a race condition and the response generation efficiency becomes a key point in this scenario. To encounter such challenge, our configuration of KIMAs needs to make the following changes.

**Application Configuration.** In order to fulfill the requirements, KIMAs is configured as the following.

- *Knowledge sources configuration.* We configure a retrieval agent to use the specialized [search API](#) for Olympic related events. The APIs perform searches similar to public search engines using keywords to match related information.
- *Pipeline configuration.* To extremely reduce response latency, the pipeline configuration becomes simple. The context manager is deactivated, and the only retrieval agent will perform a keyword query rewrite only. The full conversation history and the retrieved knowledge will be directly fed to the summarizer to generate the final answer.

With such knowledge and pipeline configuration, the end-to-end latency is reduced to less than 10 seconds per post or comment, while the responses are still very informative and popular.

## 5 Conclusion

In this technical report, we introduce KIMAs, our configurable knowledge-integrated multi-agent system for developers to build their knowledge-intensive QA system. We present three use cases built on our system with different emphases to demonstrate that KIMAs can be configured to various applications. While the current version of KIMAs focuses on knowledge-intensive QA tasks, future development can expand its capabilities to address a broader range of challenges, including code generation based on some specific local code base and interactive recommendation system for e-business.

<sup>2</sup><https://weibo.com/u/7929611818>

## References

- [1] Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George Bm Van Den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, et al. Improving language models by retrieving from trillions of tokens. In *International conference on machine learning*, pages 2206–2240. PMLR, 2022.
- [2] Chi-Min Chan, Weize Chen, Yusheng Su, Jianxuan Yu, Wei Xue, Shanghang Zhang, Jie Fu, and Zhiyuan Liu. Chateval: Towards better llm-based evaluators through multi-agent debate. *arXiv preprint arXiv:2308.07201*, 2023.
- [3] Gordon V Cormack, Charles LA Clarke, and Stefan Buettcher. Reciprocal rank fusion outperforms condorcet and individual rank learning methods. In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, pages 758–759, 2009.
- [4] BV Elasticsearch. Elasticsearch. *software*, version, 6(1), 2018.
- [5] Dawei Gao, Zitao Li, Xuchen Pan, Weirui Kuang, Zhijian Ma, Bingchen Qian, Fei Wei, Wenhao Zhang, Yuexiang Xie, Daoyuan Chen, et al. Agentscope: A flexible yet robust multi-agent platform. *arXiv preprint arXiv:2402.14034*, 2024.
- [6] Luyu Gao, Xueguang Ma, Jimmy Lin, and Jamie Callan. Precise zero-shot dense retrieval without relevance labels. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1762–1777, 2023.
- [7] Tianyu Gao, Howard Yen, Jiatong Yu, and Danqi Chen. Enabling large language models to generate text with citations. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 6465–6488, 2023.
- [8] Kostas Hatalis, Despina Christou, Joshua Myers, Steven Jones, Keith Lambert, Adam Amos-Binks, Zohreh Dannenhauer, and Dustin Dannenhauer. Memory matters: The need to improve long-term memory in llm-agents. In *Proceedings of the AAAI Symposium Series*, volume 2, pages 277–280, 2023.
- [9] Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiwu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, et al. Metagpt: Meta programming for a multi-agent collaborative framework. In *The Twelfth International Conference on Learning Representations*.
- [10] Gautier Izacard and Edouard Grave. Leveraging passage retrieval with generative models for open domain question answering. *arXiv preprint arXiv:2007.01282*, 2020.
- [11] Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. Swe-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations*.
- [12] Vladimir Karpukhin, Barlas Oğuz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. Dense passage retrieval for open-domain question answering. *arXiv preprint arXiv:2004.04906*, 2020.
- [13] Kimi.ai. Kimi.ai. <https://www.perplexity.ai/>, 2023. Accessed: 2025-01-09.
- [14] LangChain. Langchain. <https://www.langchain.com/>, 2023. Accessed: 2025-01-09.
- [15] Guohao Li, Hasan Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. Camel: Communicative agents for "mind" exploration of large language model society. *Advances in Neural Information Processing Systems*, 36:51991–52008, 2023.
- [16] Guanyu Lin, Tao Feng, Pengrui Han, Ge Liu, and Jiaxuan You. Arxiv copilot: A self-evolving and efficient LLM system for personalized academic assistance. In Delia Irazu Hernandez Farias, Tom Hope, and Manling Li, editors, *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 122–130, Miami, Florida, USA, November 2024. Association for Computational Linguistics.

- [17] Bo Liu, Yuqian Jiang, Xiaohan Zhang, Qiang Liu, Shiqi Zhang, Joydeep Biswas, and Peter Stone. Llm+ p: Empowering large language models with optimal planning proficiency. *arXiv preprint arXiv:2304.11477*, 2023.
- [18] LlamaIndex. LlamaIndex: Build ai knowledge assistants over your enterprise data. <https://www.llamaindex.ai/>, 2023. Accessed: 2025-01-09.
- [19] Pan Lu, Baolin Peng, Hao Cheng, Michel Galley, Kai-Wei Chang, Ying Nian Wu, Song-Chun Zhu, and Jianfeng Gao. Chameleon: Plug-and-play compositional reasoning with large language models. *Advances in Neural Information Processing Systems*, 36, 2024.
- [20] Sewon Min, Xinxi Lyu, Ari Holtzman, Mikel Artetxe, Mike Lewis, Hannaneh Hajishirzi, and Luke Zettlemoyer. Rethinking the role of demonstrations: What makes in-context learning work? In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 11048–11064, 2022.
- [21] Aaron Parisi, Yao Zhao, and Noah Fiedel. Talm: Tool augmented language models. *arXiv preprint arXiv:2205.12255*, 2022.
- [22] Perplexity. Perplexity. <https://www.perplexity.ai/>, 2023. Accessed: 2025-01-09.
- [23] Ori Ram, Yoav Levine, Itay Dalmedigos, Dor Muhlgay, Amnon Shashua, Kevin Leyton-Brown, and Yoav Shoham. In-context retrieval-augmented language models. *Transactions of the Association for Computational Linguistics*, 11:1316–1331, 2023.
- [24] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems*, 36:68539–68551, 2023.
- [25] Bing Search. Bing search. <https://www.bing.com/>. Accessed: 2025-01-09.
- [26] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36, 2024.
- [27] Xiangru Tang, Anni Zou, Zhuosheng Zhang, Ziming Li, Yilun Zhao, Xingyao Zhang, Arman Cohan, and Mark Gerstein. Medagents: Large language models as collaborators for zero-shot medical reasoning. *arXiv preprint arXiv:2311.10537*, 2023.
- [28] Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, et al. Emergent abilities of large language models. *arXiv preprint arXiv:2206.07682*, 2022.
- [29] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.
- [30] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.
- [31] Lilian Weng. LLM powered autonomous agents. <https://lilianweng.github.io/posts/2023-06-23-agent/>, 2023. Accessed: 2025-01-09.
- [32] Feijie Wu, Zitao Li, Fei Wei, Yaliang Li, Bolin Ding, and Jing Gao. Talk to right specialists: Routing and planning in multi-agent system for question answering, 2025.
- [33] Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Shaokun Zhang, Erkang Zhu, Beibin Li, Li Jiang, Xiaoyun Zhang, and Chi Wang. Autogen: Enabling next-gen llm applications via multi-agent conversation framework. *arXiv preprint arXiv:2308.08155*, 2023.

- [34] John Yang, Carlos E Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering. *arXiv preprint arXiv:2405.15793*, 2024.
- [35] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *Advances in Neural Information Processing Systems*, 36, 2024.
- [36] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.