Towards the Next Generation of Agent Systems: From RAG to Agentic AI

Yingli Zhou
The Chinese University of Hong Kong, Shenzhen
Shenzhen, China
yinglizhou@link.cuhk.edu.cn

Shu Wang
The Chinese University of Hong Kong, Shenzhen
Shenzhen, China
shuwang3@link.cuhk.edu.cn

ABSTRACT

With the powerful emergence of AI technology, intelligent agent systems that are capable of precise task planning, efficient collaborative operation, and in-depth resolution of complex tasks are becoming the focal point of intense attention from both the industrial and academic communities. A typical agent system includes: (1) a planning module for task decomposition, agent role assignment, and collaboration structure generation; (2) an execution module for carrying out assigned tasks, managing progress, and coordinating intermediate outputs; (3) a knowledge module that guides these components via retrieval-augmented generation and memory; and (4) a tool module for selecting and invoking external tools to support execution. Despite recent advances, current systems remain limited in adaptivity, efficiency, accuracy, and robustness. This vision paper reviews and analyzes existing works across these four modules, identifies key limitations, and outlines future research directions toward the next generation of intelligent systems.

VLDB Workshop Reference Format:
Yingli Zhou and Shu Wang. Towards the Next Generation of Agent Systems: From RAG to Agentic AI. VLDB 2025 Workshop: LLM+Graph.

1 INTRODUCTION

The recent advancements in large language models (LLMs) such as GPT-4 [1], Gemini [34], and Qwen [41] have laid the foundation for building intelligent agents capable of interacting with users in natural language, solving diverse tasks, and integrating with external environments. As LLMs evolve from passive completion engines to active decision-makers, the research community has witnessed a surge of interest in *agentic systems*.

These emerging agentic systems can be naturally understood as a form of *multi-agent system* (MAS), where multiple specialized agents—or modular components—collaborate to solve complex tasks. Representative examples include systems such as Manus [23], AutoGen [39], CAMEL [19], and MetaGPT [14]. In such settings, LLMs often serve as the central reasoning or orchestration unit, while core functionalities: planning, execution, knowledge retrieval, and tool use. Applications span domains like software development [15], scientific discovery [22], autonomous reasoning [42], and workflow automation [40]. Despite these promising results, *building robust, reusable, and intelligent agent systems remains an open challenge.*

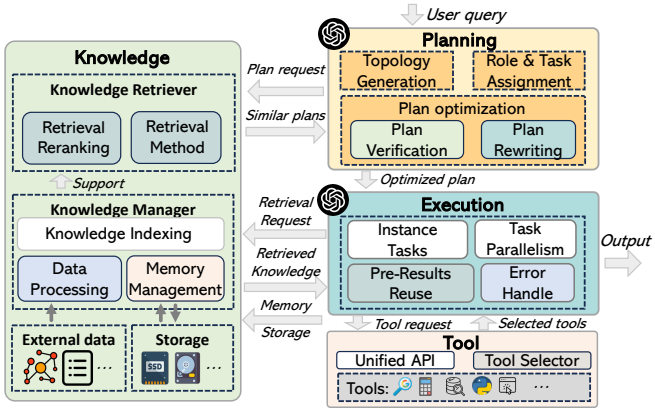

Figure 1: Overview of multi-agent system.

In this vision paper, we argue that realizing scalable and trustworthy MAS requires four core capabilities, as shown in Figure 1: (1) *Planning*, the process of decomposing goals, assigning subtasks, and organizing control flow; (2) *Execution*, the ability to instantiate, schedule, and monitor plans; (3) *Knowledge*, where retrieval and memory mechanisms provide persistent context and reasoning support; and (4) *Tool*, enabling seamless invocation and coordination of external APIs, models, or environments. These four modules interact through structured interfaces—such as plan requests, retrieval queries, and tool invocations—enabling agents to operate in a modular, interpretable, and verifiable manner. This architecture reflects the emerging design paradigm of modern MASs, where specialized components coordinate through well-defined protocols to achieve system-level intelligence.

In the remainder of this paper, we revisit recent agent architectures through this modular lens, identify representative patterns and technical bottlenecks, and outline a roadmap for building the next generation of intelligent agent systems. Just as modern commercial DBMSs have evolved into structured, verifiable, and highly optimized systems through decades of architectural refinement, we believe that modern MAS should also move toward this. Motivated by this parallel, the core insights from data management—such as declarative specification, cost-based optimization, and interoperable execution pipelines—are particularly well-suited to guide the design of next-generation MAS.

Proceedings of the VLDB Endowment. ISSN 2150-8097.

2 PRELIMINARIES

In this section, we review some key concepts of large language models (LLMs), as well as the foundational ideas behind agent and multi-agent system (MAS) architectures.

2.1 Large Language Models (LLMs)

We introduce some fundamental concepts of LLMs, including LLM prompting and retrieval-augmented generation (RAG).

LLM Prompting. After instruction tuning on large corpora of human interaction scenarios, LLMs are capable of following natural language instructions to perform a wide range of tasks [5, 27]. Specifically, given a task input, a prompt is constructed to encode the desired behavior. The LLM then processes this prompt and generates a corresponding output. Due to pre-training on trillions of tokens, LLMs demonstrate strong generalization abilities, often solving unseen tasks by simply modifying the prompt [27].

Retrieval-Augmented Generation (RAG). To improve factual grounding and context awareness, RAG enhances LLMs with access to external corpora or knowledge bases. Instead of relying solely on parametric memory, the model retrieves relevant documents or information in response to a query, and incorporates them into the prompt for answer generation. RAG is widely used in open-domain QA, tool-augmented reasoning, and knowledge-intensive tasks.

Agent. An *agent* refers to an autonomous computational entity that can perceive its environment, make decisions, and execute actions toward achieving specific goals. In the context of LLMs, an agent typically includes not only the model itself, but also the surrounding planning, execution, memory, and tool interfaces that enable task-oriented behavior. LLM-based agents operate by interpreting user instructions, formulating action plans, invoking tools or APIs, and adapting through feedback.

2.2 Multi-Agent System

A *multi-agent system (MAS)* consists of a collection of interacting agents, each responsible for different sub-goals, roles, or capabilities. MAS architectures are characterized by distributed problem-solving, role specialization, and inter-agent communication. Agents within a MAS may cooperate, coordinate, or compete to achieve system-level objectives, often using negotiation, messaging, or shared memory.

In the LLM era, MAS principles are increasingly adopted in agentic systems such as AutoGen [42], CAMEL [19], and MetaGPT [14], where different LLM agents (e.g., planner, executor, critic) work together through structured prompting protocols. This modular, role-based decomposition enables better scalability, interpretability, and division of labor, especially for complex, long-horizon tasks.

3 PLANNING

In this section, we introduce recent breakthroughs in planning strategies for modern MAS, discuss their current limitations, and highlight promising directions for future research.

Overview of planning. As shown in Figure 2, given a high-level task $\mathcal{T}$, the MAS begins by constructing the *topological structure* of the plan, which defines the decomposition of subtasks and their dependency relationships. This is followed by the *role and task assignment* phase, where specific agents are allocated to corresponding subtasks based on their capabilities and the task structure. The initial plan is then subject to *plan optimization*, including plan rewriting [38] and verification to ensure logical consistency and feasibility before execution.

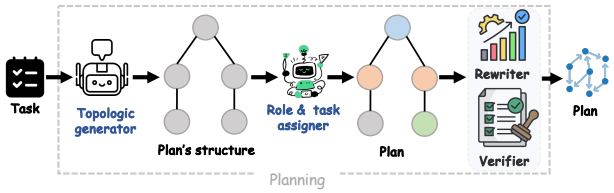

Figure 2: Overview of the planning in MAS.

3.1 Manually planning in MAS

Traditional planning in MAS depends heavily on expert-designed rules, finite state machines, or hierarchical task networks. While interpretable and precise, such methods demand significant domain expertise and manual effort, limiting their scalability and adaptability. As this manual dependency becomes a bottleneck in complex, dynamic environments, the community has increasingly turned to automated, data-driven planning approaches to enhance flexibility and reduce human intervention.

3.2 Automatically planning in MAS

Recent advances have enabled several categories of automated planning in MAS:

- **LLM-direct planning.** LLMs are increasingly employed to decompose goals and generate task plans in natural language. Specifically, systems like `HuggingGPT` [32] leverage GPT-4 to generate a task graph—typically modeled as a directed acyclic graph (DAG)—where each node represents a specific task and each directed edge encodes a dependency between tasks. This structured decomposition enables effective task coordination and has demonstrated superior performance in complex tasks. While powerful, LLM-based planning still faces challenges such as hallucinations, inconsistencies, and difficulties in verifiability.
- **Search-based planning.** Search-based strategies aim to generate high-quality plans by systematically exploring the space of task sequences or workflows. A notable approach is `AFlow` [47], which frames agent planning as workflow code optimization. Instead of searching in action space, `AFlow` applies MCTS over structured code representations, iteratively refining plans using execution feedback and learned priors. Evaluated on six benchmarks, it achieves a 5.7% average improvement over strong baselines.
- **Learning-based planning.** Learning-based planning approaches enable agents to acquire effective policies through interaction with their environment, facilitating the emergence of complex behaviors and coordinated strategies. For example, the `G-Designer` [46] leverages graph neural networks (GNNs) to dynamically construct adaptive communication topologies among agents, significantly enhancing efficiency and robustness in multi-agent collaboration. On the other hand, such approaches underscore the potential

of integrating learning-based methods with structural optimization, paving the way for more scalable and intelligent planning in complex environments.

3.3 Future direction

In this subsection, we outline three promising opportunities for the future of planning strategies for the MAS.

▶ **Lessons from historical plans.** By analyzing the execution of past plans—both successful and failed—it is possible to identify recurring patterns that characterize effective or problematic strategies [43, 50]. These patterns serve as reusable knowledge, enabling better planning and adaptation when similar queries arise in the future. The key challenges lie in: (1) effectively abstracting and structuring the learned lessons to support efficient storage and retrieval, and (2) dynamically adapting those lessons to fit the requirements of new, context-specific plans.

▶ **Rule-based Plan Rewriting and Verification.** Existing agent planning approaches heavily rely on LLMs and the reflection paradigm, where plans are iteratively revised by prompting the LLM to self-correct [31]. While flexible, this process often lacks structure, interpretability, and consistency, and may yield brittle or suboptimal plans. Inspired by query optimization in databases, we explore an alternative: rule-based plan rewriting, which improves efficiency and correctness through structured transformations. For example, techniques like predicate pushdown—which applies filters early in execution—can be mirrored by restructuring plans to front-load constraints or eliminate redundant steps. In addition to rewriting, plan verification plays a critical role in ensuring that the transformed plan still satisfies the original intent. Such techniques as constraint checking, symbolic execution, or goal-condition validation can be applied to verify the semantic equivalence and safety of rewritten plans before execution.

▶ **Declarative plan specification.** Similar to query languages like SQL, declarative plan representations allow users to specify what the desired outcome is, rather than how to achieve it [20]. This abstraction enables the system to autonomously explore and optimize execution strategies, making planning more flexible and adaptable. In agent systems, declarative specifications help reduce ambiguity, avoid unreliable or ad hoc plan generation, and enforce a unified, structured representation of intent. To achieve this, this approach also requires system-level support to interpret, compile, and execute such high-level specifications effectively.

4 EXECUTION

In this section, we introduce the key components of the plan execution in modern MAS, discuss their current limitations, and highlight promising directions for future research.

4.1 From Logical plan to physical plan

Given the optimized agentic plan, the execution module is responsible for instantiating and dispatching specific tasks to the appropriate tools or environments. This process resembles query execution in database systems: the agentic plan corresponds to a *logical plan*, while the execution process maps to a *physical plan* that determines how and when each action is carried out.

The system must ensure that the high-level intent of the plan is faithfully and efficiently realized through concrete execution strategies. This involves task instantiation, scheduling, error handling, and potential reuse of intermediate results to avoid redundant computations.

4.2 Execution components

As shown in Figure 1, the execution module consists of four key components:

- **Instance Tasks**: This component instantiates the abstract plan into concrete API/tool calls or function executions, filling in parameters and inputs as needed.
- **Task Parallelism**: When possible, independent tasks are executed in parallel to improve overall efficiency and reduce latency [16].
- **Pre-Results Reuse**: To avoid redundant computation, the system caches and reuses results from previously executed tasks when encountering similar queries or subtasks.
- **Error Handle**: Robust execution requires the ability to detect failures, handle tool unavailability, and trigger fallback or repair strategies to maintain reliability.

Together, these components form a flexible and resilient execution layer capable of translating abstract plans into real-world actions while optimizing for performance and robustness.

4.3 Future direction

▶ **Learning-based Execution Optimization.** Inspired by query optimization in databases, future execution modules can learn to select optimal strategies—such as parallelism levels or fallback ordering—based on historical execution traces and contextual features. The key tasks are: (1) collecting and generalizing execution feedback across heterogeneous tasks, and (2) designing effective learning-based models.

▶ **Adaptive recovery and self-healing.** To ensure robust agent behavior in dynamic or failure-prone environments, execution systems must support adaptive recovery strategies. These include rollback, dynamic plan revision, and tool substitution based on failure signals.

▶ **Cross-session result reuse.** Beyond single-query caching, execution modules can leverage persistent result reuse across users and sessions, enabling long-term memory and improved efficiency. For example, systems can proactively index and manage cached results across diverse plan contexts, while also ensuring semantic validity and version compatibility when reusing prior outputs.

5 KNOWLEDGE

In this section, we introduce the role of knowledge in MAS, focusing on how agents store, retrieve, and update information from both external sources and internal memory. Knowledge modules enable agents to reason over past experiences and external data, supporting informed and context-aware decision-making.

Overview of knowledge. As shown in Figure 3, the knowledge module first processes raw context and memory to build a structured *knowledge index* (e.g., vector database, knowledge graph, and graph database), which is continuously updated with new memory

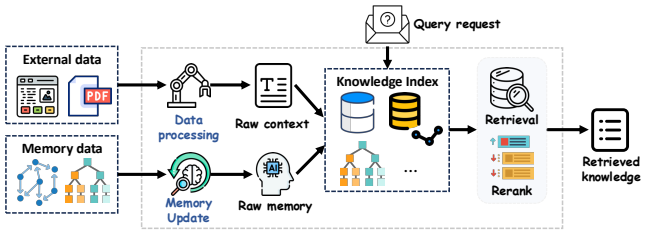

Figure 3: Overview of the knowledge in MAS.

and external data. Given a query, relevant information is retrieved and then *reranked* to select the most appropriate knowledge, which is subsequently used to support downstream planning and execution.

5.1 Knowledge manager

The Knowledge Manager orchestrates how agents store and update information from external sources and past experiences. It handles knowledge indexing and data processing as well as memory management. Typically, external knowledge encompasses a variety of formats, including unstructured documents such as web pages and PDF files, structured data from knowledge graphs and databases. Internal memory captures the agent's own experiences, such as successful planning examples, execution traces, dialogues, and inferred facts accumulated over time.

To effectively utilize this diverse information, the Knowledge Manager employs data processing techniques that transform heterogeneous knowledge sources into structured representations suitable for retrieval and reasoning. For example, it may involve: parsing and interpreting various document formats and semantic chunking to divide content into meaningful units;

Complementing data processing, memory management dynamically handles the agent's internal memory to ensure relevance and accuracy. This includes: 1) Inserting new experiences and knowledge into memory. 2) Retrieving similar past experiences to inform current decision-making. 3) Updating or removing outdated or incorrect information to maintain memory integrity.

Knowledge indexing involves constructing and maintaining efficient data structures that facilitate rapid and contextually relevant information retrieval. In practice, a hybrid storage model is common: an agent may store data in both a graph database and a vector database. This allows semantic embeddings to be indexed in a vector store for similarity search, while structured relationships are captured in a knowledge graph. Currently, knowledge index can be broadly categorized into the following categories:

- **Graph-based indexing**: LLMs can construct knowledge graphs from corpora (entities, relations, communities) to index information. For example, GraphRAG [7] uses an LLM to extract entities and relations from text, building a knowledge graph and community hierarchy for indexing.
- **Graph databases**: Knowledge graphs may be stored in systems like Neo4j [25] or NetworkX [13], enabling complex relational queries and incremental updates.
- **Vector stores**: Embedding-based indexes (Faiss [6], Milvus [36], etc.) hold dense vectors of text chunks. These databases support fast k-NN search on semantic embeddings.

5.2 Knowledge retriever

The Knowledge Retriever module fetches relevant information to support agent reasoning. In practice, systems often use a two-stage pipeline: a fast first-stage retriever produces candidates, and a second-stage reranker refines them. Recent research has introduced advanced graph-augmented retrieval techniques alongside reranking to boost accuracy and context-awareness, which can be broadly categorized into two main approaches:

- **Classic and dense retrieval**: Classic and dense retrieval: Keyword-based methods like BM25 remain strong baselines for first-stage retrieval. Dense retrievers (e.g., DPR or embedding lookup) index vectors for semantic search [3]. These can be deployed in vector databases (Faiss [6] / Milvus [36]) that are optimized for large-scale similarity search.
- **Graph-based retrieval**: Graph-based RAG systems leverage graph structure during retrieval [45, 49]. For example, HippoRAG [12] borrows from hippocampal memory to integrate new experiences via knowledge graphs and PageRank. LightRAG [10] combines graph structures with vector indexes in a dual-level retrieval process. ArchRAG [37] constructs an attributed, community-based hierarchical graph index: it augments the query with detected communities and employs LLM-based clustering to guide retrieval.

Following the initial retrieval stage, the Knowledge Retriever module employs reranking techniques to refine the list of candidate documents, enhancing the relevance and contextual appropriateness of the information presented to the agent. Reranking methods can be broadly categorized into two primary approaches: neural reranking and fine-tuned LLMs. The former ones typically use models such as BERT [4] and T5 [30] to jointly encode query-document pairs, allowing for deep interaction between the query and each candidate document. For instance, frameworks like HLATR [48] have been developed to integrate features from both retrieval and reranking stages, resulting in efficient and effective multi-stage retrieval systems. On the other hand, reranking with fine-tuned LLMs has gained attention for its ability to leverage the extensive pretraining of these models to capture complex language patterns and contextual information. Models such as RankLLaMA [21], which is fine-tuned from the LLaMA architecture, have demonstrated strong performance in reranking applications.

The integration of these reranking strategies within the Knowledge Retriever module enhances the agent's ability to access and utilize pertinent information, thereby improving reasoning capabilities and overall system performance.

5.3 Future direction

We identify several promising directions to evolve knowledge systems for MAS:

▶ **Dynamic, incremental knowledge updates.** Agentic systems demand a memory architecture capable of evolving alongside newly acquired data. Static indices are insufficient in dynamic environments where information changes rapidly; instead, agents require

a continuously updated memory layer that integrates ongoing interactions and incoming documents in real time. Key challenges lie in updating graph and vector indices incrementally without resorting to full recomputation, ensuring consistency across successive updates, and efficiently revising community-level summaries or embeddings.

▶ **Lighter, faster, adaptive retrieval.** Current deep retrieval can be computationally heavy. Future systems should leverage model compression, quantized embeddings, and smarter indexing to reduce latency. Research challenges include balancing model size and retrieval quality, developing on-the-fly adaptation (e.g., warming caches with expected queries), and creating evaluation benchmarks for latency in agentic settings.

▶ **Integration of other components.** To enhance agent robustness and adaptability, the knowledge module should be tightly integrated with planning and execution components. When plans fail verification or task execution encounters errors, these signals can be fed back to trigger targeted re-retrieval, content validation, or memory updates. Such a feedback loop enables context-aware refinement of knowledge usage, improving system reliability over time. Future systems will benefit from unified memory interfaces and cross-module coordination protocols to support continual correction and adaptation.

6 TOOLS

In this section, we delve into the critical role of tools within agentic AI systems, focusing on their integration, selection, and future directions. Tools empower agents to interact with external environments, execute complex tasks, and enhance their reasoning capabilities.

6.1 Tools library

Early AI agents typically relied on a fixed, hand-crafted set of utilities or APIs, each manually integrated into the system. Such static toolsets were often rigid: for example, an agent might include a hard-coded function for database lookup or a fixed knowledge base. These legacy designs meant that adding new capabilities required bespoke coding, and tools could not be discovered or composed dynamically at run time. This limitation is well illustrated by tool-augmented models like Toolformer – despite using external APIs, Toolformer is "constrained by a fixed set of available tools" and cannot chain or extend tools flexibly.

In contrast, modern agentic AI envisions dynamic, extensible tool libraries whereby agents can select, invoke, and even discover new tools as needed. Common categories of tools in LLM-based agent systems include:

- **Search tools**: Searching allows the agent to query external information sources (e.g., the web search API, such as Google/Bing) to retrieve the real-time relevant data, particularly for knowledge-intensive tasks. For example, Qin et al. [33] describe using a search engine tool to extend an LLM's information beyond its training data.
- **Computational tools**: Computational tools encompass any functions that perform reliable mathematical or logical computation. Examples include using a Python REPL, scientific computing libraries, or services like Wolfram Alpha.
- **Code execution tools**: Code execution provides general programming capabilities by running code in a live environment, used for tasks like data analysis, simulations, or even invoking other software components. A common use case is *LangChain's PythonREPL tools*, where the agent can generate code snippets, execute them, and retrieve outputs such as numerical results or visualizations.
- **API invocation tools**: API tools serve as connectors that enable agents to interact with external web services or plugins. These tools allow agents to access a wide range of HTTP-based endpoints, including RESTful, SOAP, or GraphQL interfaces, ranging from weather and stock data providers to translation or text-to-speech engines. An illustrative example is *WebAgent* [11], an LLM-driven agent designed for real-world web automation tasks.

Several contemporary frameworks provide scalable tool libraries for agent development. For example, LangChain [35] treats tools as first-class objects, allowing LLMs to autonomously decide when to invoke them. The Model Context Protocol (MCP) [2], introduced by Anthropic, standardizes tools as "plug-and-play" services, enabling AI assistants to connect directly to various data sources without custom code. AutoGen [39], an open-source multi-agent framework by Microsoft, facilitates collaboration among multiple LLM agents, each potentially invoking tools in natural dialogue.

Unified API. To streamline tool integration and enhance agent interoperability, the concept of a Unified API has emerged as a pivotal solution. This approach standardizes the interface for tool invocation, allowing agents to seamlessly interact with both predefined and user-defined tools through a consistent schema. Frameworks like ModelScope-Agent [18] exemplify this by enabling open-source LLMs to connect with over 1,000 AI models and services via a unified interface, facilitating efficient tool registration and usage. Additionally, ToolFactory [26] automates the generation of AI-compatible tools from unstructured REST API documentation, addressing challenges of inconsistent schemas and incomplete information. By adopting a Unified API framework, agents can dynamically select and invoke appropriate tools, thereby enhancing their adaptability and effectiveness in complex, real-world tasks.

6.2 Tools selection

Once a task is planned or decomposed, an agent must choose which tool(s) to invoke. In practice, this often involves two stages: first, narrowing the candidate tools by matching task content to tool descriptions, and then selecting from among the shortlist [29]. A common strategy is to retrieve the top-k tools whose descriptions are most relevant to the task or subtask. Some studies [9, 17, 28] train a SentenceBert model as the tool retriever, enabling the high-efficiency retrieval of relevant tools.

Beyond simple retrieval, an LLM-based selector can be employed by providing the model with the names, descriptions, and parameters of the candidate tools, prompting it to identify the most suitable choice. It can be further divided into two broad categories, depending on whether they require fine-tuning:

- **Tuning-Free Methods.** These approaches operate directly by providing a set of candidate tools and a task description to the LLM, prompting it to choose the most appropriate

tool using zero-shot or few-shot examples. Such methods are straightforward to deploy and easily adaptable to different tool libraries. For instance, ToolLLM [28] introduces tool-use abilities via instruction tuning. ReAct [44] is a framework that integrates reasoning with action, enabling LLMs to not only justify actions but also to refine their reasoning processes based on feedback from the environment.

- **Tuning-Based Methods.** The goal of tuning-based methods is to endow the model with stronger tool-selection abilities by learning from supervised or reinforcement learning signals. ToolVerifier [24] introduces a self-verification method that distinguishes between close candidates by self-asking Such methods generally offer higher accuracy and robustness but require labeled data and training costs.

Together, these strategies highlight the evolving sophistication of tool selection in agentic systems—from simple matching to context-aware reasoning and learned decision policies.

6.3 Future direction

In this subsection, we provide three opportunities for the future of the tool in MAS.

▶ **Learning tool selection policies.** Rather than relying on fixed heuristics or prompting, agents could learn to pick tools via feedback. Training a policy (e.g., via reinforcement learning) to choose tools could yield more robust and adaptive behavior, especially as new tools appear. For example, AGILE [8] demonstrates that fine-tuning an agent end-to-end with PPO improves its ability to invoke tools effectively in conversational QA.

▶ **Automatic tool discovery and composition.** Agents that can autonomously find or create new tools would greatly expand their capability. For example, composing chains of simple tools into higher-level macros is a promising avenue: an agent might automatically sequence or "chain" tools to perform complex tasks. Investigating how LLMs can discover, synthesize, or recombine tools on the fly is a key open challenge.

▶ **Standardization of tool schema.** Consistent API interfaces for tools would make it easier for agents to reason about and use them. Defining a common schema or ontology for tool inputs and outputs (for example, JSON-based function specifications) allows agents to automatically parse and generate calls without parsing. Going forward, developing standardized metadata (arguments, effects, data types) for tools could improve interoperability and reduce the burden of integrating new tools into agent systems.

7 CONCLUSION

In this vision paper, we examined intelligent multi-agent systems through four core modules: planning, execution, knowledge, and tools. By analyzing recent architectures, we identified key limitations in adaptability, robustness, and modularity. We further outlined future directions toward building agentic systems that are more structured, verifiable, and reusable. We hope this perspective inspires new research toward more capable and trustworthy AI agent systems.

REFERENCES

[1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774* (2023).

[2] Anthropic. 2024. Introducing the Model Context Protocol. https://www.anthropic.com/news/model-context-protocol Accessed: 2025-05-26.

[3] Riyaz Ahmad Bhat, Jaydeep Sen, Rudra Murthy, et al. 2025. UR2N: Unified Retriever and ReraNker. In *Proceedings of the 31st International Conference on Computational Linguistics: Industry Track*. 595–602.

[4] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*. 4171–4186.

[5] Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Jingyuan Ma, Rui Li, Heming Xia, Jingjing Xu, Zhiyong Wu, Tianyu Liu, et al. 2022. A survey on in-context learning. *arXiv preprint arXiv:2301.00234* (2022).

[6] Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. 2024. The faiss library. *arXiv preprint arXiv:2401.08281* (2024).

[7] Darren Edge, Ha Trinh, Newman Cheng, Joshua Bradley, Alex Chao, Apurva Mody, Steven Truitt, Dasha Metropolitansky, Robert Osazuwa Ness, and Jonathan Larson. 2024. From Local to Global: A Graph RAG Approach to Query-Focused Summarization. *arXiv preprint arXiv:2404.16130* (2024).

[8] Peiyuan Feng, Yichen He, Guanhua Huang, Yuan Lin, Hanchong Zhang, Yuchen Zhang, and Hang Li. 2024. AGILE: A Novel Reinforcement Learning Framework of LLM Agents. *arXiv preprint arXiv:2405.14751* (2024).

[9] Shen Gao, Zhengliang Shi, Minghang Zhu, Bowen Fang, Xin Xin, Pengjie Ren, Zhumin Chen, Jun Ma, and Zhaochun Ren. 2024. Confucius: Iterative tool learning from introspection feedback by easy-to-difficult curriculum. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 38. 18030–18038.

[10] Zirui Guo, Lianghao Xia, Yanhua Yu, Tu Ao, and Chao Huang. 2024. Lightrag: Simple and fast retrieval-augmented generation. (2024).

[11] Izzeddin Gur, Hiroki Furuta, Austin Huang, Mustafa Safdari, Yutaka Matsuo, Douglas Eck, and Aleksandra Faust. 2023. A real-world webagent with planning, long context understanding, and program synthesis. *arXiv preprint arXiv:2307.12856* (2023).

[12] Bernal Jiménez Gutiérrez, Yiheng Shu, Yu Gu, Michihiro Yasunaga, and Yu Su. 2024. Hipporag: Neurobiologically inspired long-term memory for large language models. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

[13] Aric Hagberg and Drew Conway. 2020. Networkx: Network analysis with python. *URL: https://networkx. github. io* (2020), 1–48.

[14] Sirui Hong, Xiawu Zheng, Jonathan Chen, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, et al. 2023. Metagpt: Meta programming for multi-agent collaborative framework. *arXiv preprint arXiv:2308.00352* 3, 4 (2023), 6.

[15] Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. 2023. Swe-bench: Can language models resolve real-world github issues? *arXiv preprint arXiv:2310.06770* (2023).

[16] Sehoon Kim, Suhong Moon, Ryan Tabrizi, Nicholas Lee, Michael W Mahoney, Kurt Keutzer, and Amir Gholami. 2024. An llm compiler for parallel function calling. In *Forty-first International Conference on Machine Learning*.

[17] Yilun Kong, Jingqing Ruan, Yihong Chen, Bin Zhang, Tianpeng Bao, Shiwei Shi, Guoqing Du, Xiaoru Hu, Hangyu Mao, Ziyue Li, et al. 2023. Tptu-v2: Boosting task planning and tool usage of large language model-based agents in real-world systems. *arXiv preprint arXiv:2311.11315* (2023).

[18] Chenliang Li, Hehong Chen, Ming Yan, Weizhou Shen, Haiyang Xu, Zhikai Wu, Zhicheng Zhang, Wenmeng Zhou, Yingda Chen, Chen Cheng, et al. 2023. Modelscope-agent: Building your customizable agent system with open-source large language models. *arXiv preprint arXiv:2309.00986* (2023).

[19] Guohao Li, Hasan Abed Al Kader Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. 2023. CAMEL: Communicative Agents for "Mind" Exploration of Large Language Model Society. In *Thirty-seventh Conference on Neural Information Processing Systems*.

[20] Chunwei Liu, Matthew Russo, Michael Cafarella, Lei Cao, Peter Baile Chen, Zui Chen, Michael Franklin, Tim Kraska, Samuel Madden, Rana Shahout, et al. 2025. Palimpzest: Optimizing ai-powered analytics with declarative query processing. In *Proceedings of the Conference on Innovative Database Research (CIDR)*.

[21] Xueguang Ma, Liang Wang, Nan Yang, Furu Wei, and Jimmy Lin. 2024. Fine-tuning llama for multi-stage text retrieval. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 2421–2425.

[22] Yubo Ma, Zhibin Gou, Junheng Hao, Ruochen Xu, Shuohang Wang, Liangming Pan, Yujiu Yang, Yixin Cao, Aixin Sun, Hany Awadalla, et al. 2024. Sciagent: Tool-augmented language models for scientific reasoning. *arXiv preprint arXiv:2402.11451* (2024).

[23] manus. 2025. manus. https://manus.im/.

[24] Dheeraj Mekala, Jason Weston, Jack Lanchantin, Roberta Raileanu, Maria Lomeli, Jingbo Shang, and Jane Dwivedi-Yu. 2024. Toolverifier: Generalization to new tools via self-verification. *arXiv preprint arXiv:2402.14158* (2024).

[25] Justin J Miller. 2013. Graph database applications and concepts with Neo4j. In *Proceedings of the southern association for information systems conference, Atlanta, GA, USA*, Vol. 2324. 141–147.

[26] Xinyi Ni, Qiuyang Wang, Yukun Zhang, and Pengyu Hong. 2025. ToolFactory: Automating Tool Generation by Leveraging LLM to Understand REST API Documentations. *arXiv preprint arXiv:2501.16945* (2025).

[27] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. 2022. Training language models to follow instructions with human feedback. *Advances in neural information processing systems* 35 (2022), 27730–27744.

[28] Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. 2023. Toolllm: Facilitating large language models to master 16000+ real-world apis. *arXiv preprint arXiv:2307.16789* (2023).

[29] Changle Qu, Sunhao Dai, Xiaochi Wei, Hengyi Cai, Shuaiqiang Wang, Dawei Yin, Jun Xu, and Ji-Rong Wen. 2025. Tool learning with large language models: A survey. *Frontiers of Computer Science* 19, 8 (2025), 198343.

[30] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research* 21, 140 (2020), 1–67.

[31] Matthew Renze and Erhan Guven. 2024. Self-reflection in llm agents: Effects on problem-solving performance. *arXiv preprint arXiv:2405.06682* (2024).

[32] Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. 2023. Hugginggpt: Solving ai tasks with chatgpt and its friends in hugging face. *Advances in Neural Information Processing Systems* 36 (2023), 38154–38180.

[33] Zhengliang Shi, Shen Gao, Xiuyi Chen, Yue Feng, Lingyong Yan, Haibo Shi, Dawei Yin, Pengjie Ren, Suzan Verberne, and Zhaochun Ren. 2024. Learning to use tools via cooperative and interactive agents. *arXiv preprint arXiv:2403.03031* (2024).

[34] Gemini Team, Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, Katie Millican, et al. 2023. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805* (2023).

[35] Oguzhan Topsakal and Tahir Cetin Akinci. 2023. Creating large language model applications utilizing langchain: A primer on developing llm apps fast. In *International Conference on Applied Engineering and Natural Sciences*, Vol. 1. 1050–1056.

[36] Jianguo Wang, Xiaomeng Yi, Rentong Guo, Hai Jin, Peng Xu, Shengjun Li, Xiangyu Wang, Xiangzhou Guo, Chengming Li, Xiaohai Xu, et al. 2021. Milvus: A purpose-built vector data management system. In *Proceedings of the 2021 International Conference on Management of Data*. 2614–2627.

[37] Shu Wang, Yixiang Fang, Yingli Zhou, Xilin Liu, and Yuchi Ma. 2025. ArchRAG: Attributed Community-based Hierarchical Retrieval-Augmented Generation. arXiv:2502.09891 [cs.IR] https://arxiv.org/abs/2502.09891

[38] Yifan Wang, Haodi Ma, and Daisy Zhe Wang. 2024. No more optimization rules: LLM-enabled policy-based multi-modal query optimizer. *arXiv preprint arXiv:2403.13597* (2024).

[39] Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, et al. 2023. Autogen: Enabling next-gen llm applications via multi-agent conversation. *arXiv preprint arXiv:2308.08155* (2023).

[40] Tianbao Xie, Fan Zhou, Zhoujun Cheng, Peng Shi, Luoxuan Weng, Yitao Liu, Toh Jing Hua, Junning Zhao, Qian Liu, Che Liu, et al. 2023. Openagents: An open platform for language agents in the wild. *arXiv preprint arXiv:2310.10634* (2023).

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

[42] Hui Yang, Sifu Yue, and Yunzhong He. 2023. Auto-gpt for online decision making: Benchmarks and additional opinions. *arXiv preprint arXiv:2306.02224* (2023).

[43] Ling Yang, Zhaochen Yu, Tianjun Zhang, Shiyi Cao, Minkai Xu, Wentao Zhang, Joseph E Gonzalez, and Bin Cui. 2024. Buffer of thoughts: Thought-augmented reasoning with large language models. *Advances in Neural Information Processing Systems* 37 (2024), 113519–113544.

[44] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*.

[45] Fangyuan Zhang, Zhengjun Huang, Yingli Zhou, Qintian Guo, Zhixun Li, Wensheng Luo, Di Jiang, Yixiang Fang, and Xiaofang Zhou. 2025. EraRAG: Efficient and Incremental Retrieval Augmented Generation for Growing Corpora. *arXiv preprint arXiv:2506.20963* (2025).

[46] Guibin Zhang, Yanwei Yue, Xiangguo Sun, Guancheng Wan, Miao Yu, Junfeng Fang, Kun Wang, Tianlong Chen, and Dawei Cheng. 2024. G-designer: Architecting multi-agent communication topologies via graph neural networks. *arXiv preprint arXiv:2410.11782* (2024).

[47] Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xionghui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, et al. 2024. Aflow: Automating agentic workflow generation. *arXiv preprint arXiv:2410.10762* (2024).

[48] Yanzhao Zhang, Dingkun Long, Guangwei Xu, and Pengjun Xie. 2022. HLATR: enhance multi-stage text retrieval with hybrid list aware transformer reranking. *arXiv preprint arXiv:2205.10569* (2022).

[49] Yingli Zhou, Yaodong Su, Youran Sun, Shu Wang, Taotao Wang, Runyuan He, Yongwei Zhang, et al. 2025. In-depth Analysis of Graph-based RAG in a Unified Framework. *arXiv preprint arXiv:2503.04338* (2025).

[50] Nan Zhuang, Boyu Cao, Yi Yang, Jing Xu, Mingda Xu, Yuxiao Wang, and Qi Liu. 2025. LLM Agents Can Be Choice-Supportive Biased Evaluators: An Empirical Study. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 39. 26436–26444.