Theoretical Research on RAG and Agent Systems

Assignment (Part 1)

submitted on August 29, 2025

Faculty of Economics and Health

Business informatics degree programme

Course WWI2022B

by

Maryna Vdovychenko

Table of Contents

1 RAG Systems

Retrieval-Augmented Generation (RAG) is a technique of retrieving relevant knowledge from an outside source and integration it into the generative model's input. It combines the strengths of large language models (LLMs) with external knowledge retrieval. This process enables the generative model to produce answers that are not only coherent and context-aware but also more reliable and factually grounded.[1]

A typical RAG system architecture consists of a knowledge source, an embedding model, a vector database and an LLM. These components interact with each other in four major steps in RAG pipeline. The process begins with indexing, where documents are divided into smaller chunks, transformed into embeddings and stored in a vector database for efficient similarity search. When a user submits a query, it is converted into an embedding with the same model used for indexing. In the retrieval step, this query embedding is compared with the stored vectors, ranked by similarity, and the most relevant text segments are returned. During generation, the selected top segments and the original query are added in a prompting template as context and the template is passed to LLM, which produces an answer grounded in retrieved information rather than only its pre-trained knowledge.[2]

Unlike purely generative models, which rely only on pre-trained knowledge, RAG systems combine retrieval and generation to dynamically access external sources and deliver more reliable responses.[3] Traditional QA systems depend on fixed knowledge bases, often limited to model training data such as BERT or GPT, and either extract answers directly from documents or generate them from stored knowledge. This restricts factual accuracy and contextual depth. In contrast, RAG systems overcome these limits by integrating a retriever with a generator, enabling access to current documents and producing more accurate and comprehensive answers. The trade-off is higher computational cost and latency, as retrieval introduces an additional processing step.[4]

The effectiveness of RAG system depends not only on its architecture but also on a set of components and design choices. These include how embeddings are created and stored, how context is constructed and integrated through prompting, and how models are selected with respect to performance and cost. Each of these factors directly impacts accuracy, efficiency, and scalability.

Embeddings play an important role in RAG as they capture semantic meaning and enable similarity search between user queries and documents in the database. This makes it possible to retrieve the most relevant content, which can then be used to generate more accurate answers.[5]

[1] Cf. Kimothi 2025, p. 10
[2] Cf. Kamath et al. 2024, pp. 276–277
[3] Cf. Shen et al. 2024, p. 1
[4] Cf. GeeksforGeeks 2025
[5] Cf. Bhavsar 2024

Embeddings are stored in vector databases that apply indexing methods such as Hierarchical Navigable Small World graphs (HNSW) or Facebook AI Similarity Search (FAISS) to ensure fast and precise similarity searches across millions or even billions of vectors by calculating distances between them.[6]

Context construction defines which retrieved segments are passed to the LLM and how they are ordered and formatted. An important factor is the volume parameter k, which controls how many documents are retrieved and directly influences the balance between completeness, noise, and efficiency. Too few documents may omit relevant content, while too many can add noise and slow down responses. The optimal value depends on the task as well as on the combination model and dataset.[7]

Another essential aspect is how the retrieved context is integrated. Prompting serves as the link between retrieval and generation by combining the user query and the retrieved text into a structured template (e.g., *"Using the following context, answer the question"*). This design determines how the model interprets both the query and the contextual evidence and influences the reliability of its response. While prompt changes often have little effect in QA tasks where retrieval quality dominates, they play a major role in code generation, where framing of retrieved information strongly influences output.[8]

Cost aspects in RAG depend, among other things, on the choice of embedding model and LLM. Advanced embedding models with high-dimensional vectors improve retrieval accuracy but raise computational costs for large document sets. Lightweight models mitigate this trade-off by balancing performance and efficiency, making them more suitable for real-world use.[9] In addition, querying costs are influenced by model size and latency, indexing costs by the encoder and reprocessing effort, and storage costs increase with embedding dimension, as higher-dimensional vectors require more memory and infrastructure.[10]

The choice of LLM influences RAG performance and costs. Large proprietary models (e.g., GPT-4, Claude Opus) deliver strong reasoning but are slower and more expensive, while smaller or open-source models (e.g., LLaMA 2, Mistral) are faster, cheaper, and locally deployable yet weaker on complex tasks. This trade-off makes model selection central to balancing accuracy, efficiency, and affordability.[11]

RAG systems face several challenges in practical deployment. A first limitation is the restricted context window of LLMs, which only allows a fixed number of tokens. Too much input can truncate relevant content, while too little reduces answer quality.[12] Another issue is latency, since retrieval adds processing steps before generation. Studies show that retrieval can account

[6] Cf. Shamim, Isachenko 2025, p. 41
[7] Cf. Zhao et al. 2025, pp. 2–3
[8] Cf. Zhao et al. 2025, pp. 2–3
[9] Cf. Yu et al. 2024, p. 1
[10] Cf. Bhavsar 2024
[11] Cf. Cooper 2024
[12] Cf. Juvekar, Purwar 2024, pp. 1, 6

for 40–45% of the overall delay and in some cases nearly double the time to first token, making it a major performance bottleneck.[13] Finally, retrieval quality is critical. Single-step similarity search may miss relevant passages or include noise, resulting in incomplete or misleading answers. Agentic RAG addresses this by reformulating queries, validating results, and applying multi-step retrieval to improve reliability.[14]

[13] Cf. Shen et al. 2024, p. 1
[14] Cf. Hugging Face 2025

2 Agent Systems

2.1 Foundations of Agent Systems

An agent is an AI program that operates autonomously and dynamically determines how to act instead of following a fixed sequence. The core of an agent consists of a model, tools, and instructions. The model decides whether to act, respond or reason. Tools allow the agent to interact with external systems, while instructions define its behavior. In addition, agents have memory, which lets them store and reuse information from previous interactions. In this way, they can remember user preferences and provide more personalized outputs.[15]

In contrast to this adaptive design, traditional approaches such as prompt chains rely on a rigid, predefined sequence of steps. Here, each input leads deterministically to the next stage, similar to a state machine where user interactions and decisions are hard coded in advance. While prompt chains are effective for predictable and repetitive workflows, they lack the flexibility of agents, which can leverage decision logic, tools, and memory to adjust dynamically to different contexts. This makes agents more capable of handling complex and evolving tasks.[16]

Multi-agent system is composed of multiple specialized agents that work together on complex problems. By dividing tasks across agents with different strengths, such systems become more modular, scalable, and resilient than those relying on a single agent.[17] The fundamental capabilities of such systems include tool usage, orchestration of tasks, routing work to specialized agents, memory, planning, and adapting to the user's context.

Tool usage allows agents to interact with external systems such as APIs, databases, or services. Since these systems often expect structured inputs rather than plain language, tools act as interfaces that translate between the agent's natural language instructions and the required schema. By calling tools, agents can access real-time data, perform computations, or trigger external workflows, extending their abilities beyond text generation.[18] Typical examples include code interpreters for analysis, databases or knowledge bases for structured retrieval, APIs for services like search or weather, and web browsers for extracting online content.[19]

Orchestration refers to the coordination of multiple agents that work together on a shared task. Each agent may take on a specific role, such as planning, data retrieval, or tool usage, and orchestration ensures that these activities are aligned. This design improves scalability because new agents can be added without major changes to the system. It also supports maintainability,

[15] Cf. Agno 2025c
[16] Cf. Greyling 2024
[17] Cf. Hugging Face 2025
[18] Cf. LangChain 2025a
[19] Cf. Microsoft 2025

since individual agents can be developed and tested independently. A common approach is sequential orchestration, where agents pass outputs step by step to build a complete solution.[20]

Routing involves categorizing an input and sending it to the most suitable task or agent in a multi-agent workflow. It enables clear task division so that agents can focus on roles like planning, retrieval, or tool execution, without becoming overloaded. Common methods include rule-based matching, machine learning classifiers, and LLM-based routers that apply prompts and dynamic logic. Routing patterns vary from choosing a single agent to parallel execution or hierarchical decision-making. Good routing maintains context, reduces hallucinations, improves efficiency, and increases transparency by making errors traceable to either the router or the selected agent.[21]

Memory equips agents with the ability to retain and reuse information across interactions. Short-term memory provides access to information from earlier steps within a task, while long-term memory enables recalling data from previous sessions, such as earlier user inputs or conversation history. By maintaining context over time, memory helps agents learn from past experiences, personalize responses, and make more informed decisions. Effective memory management therefore improves coherence, adaptability, and overall performance in multi-step or ongoing tasks.[22]

Planning enables an agent to structure actions across multiple steps to solve a task. Instead of answering immediately, the agent decides which tools to use, in what order, and with what inputs. Results of these tool calls are then integrated back into the reasoning process, allowing the agent to adjust its next steps. This iterative cycle continues until enough information is gathered to provide a solution. This process improves efficiency and supports complex, multi-step tasks.[23]

Context awareness is the ability of agent systems to adapt their behavior to situational factors. Context may include people, location, objects, events, or other environmental data that define an agent's situation. Context-aware systems usually acquire, abstract, and then use this information, allowing agents to understand their environment and adjust actions. In practice, context can involve task objectives, organizational roles, or timing. By using such information, agents retrieve more relevant data and make decisions better aligned with the situation.[24]

Typical frameworks for developing agent systems include LangChain, AutoGen, CrewAI, and Agno. LangChain provides modular components and integrations for connecting LLMs with external data sources and supports easy model interoperability.[25] AutoGen is an open-source framework for creating flexible multi-agent systems, recently redesigned to improve scalability and robustness.[26] CrewAI emphasizes collaboration by organizing agents into teams with de-

[20] Cf. Kittel, Siemens 2025
[21] Cf. Patronus AI 2025
[22] Cf. LangChain 2025a
[23] Cf. LangChain 2025a
[24] Cf. Du et al. 2025, p. 3
[25] Cf. LangChain 2025d
[26] Cf. Microsoft Research 2025

fined roles and goals, combined with event-driven orchestration.[27] Agno is a lightweight, model-agnostic framework focused on simplicity, speed, and support for multimodal agents.[28]

2.2 Comparison of LangChain and Agno

Architecture

At its core, LangChain defines minimal, abstract interfaces for essential components, such as chat models, vector stores, and tools, without imposing third-party dependencies. Building on this, the main LangChain package implements generic chains, agents, and retrieval strategies. In this way, LangChain provides a flexible, model-independent structure that supports reasoning and decision-making workflows for LLMs. Integrations such as LangChain-OpenAI or LangChain-Anthropic are separate packages, keeping them lightweight and versioned independently. Its ecosystem also includes LangGraph, for building stateful multi-actor applications through graph-based workflows, and LangSmith, a platform for debugging, testing, and monitoring LLM applications.[29]

Agno's architecture centers on lightweight, modular agents that combine reasoning, memory, and multi-agent orchestration. Unlike LangChain's more fine-grained abstraction layers, Agno offers a unified, model-agnostic interface to multiple LLM providers, emphasizing simplicity. It natively supports multimodal inputs and outputs (text, image, audio, video) and integrates agentic RAG for runtime retrieval from vector databases. Persistent memory and session storage are built in, as are deployment features such as structured outputs, FastAPI routes, and monitoring. These choices make Agno performance-oriented, developer-friendly, and production-ready.[30]

Modularity

LangChain distinguishes itself through a modular architecture that allows flexible composition of workflows. Developers configure models, tools, memory, and agent types as separate components and then combine them into a working system. This design enables complex applications, such as chatbots that retrieve data, process it with an LLM, and store interactions in memory. Its modularity makes components easy to swap or extend, offering high flexibility, though at the cost of added complexity and boilerplate.[31]

In contrast, Agno takes a declarative approach, defining the model, tools, and instructions directly within the agent for a unified and more readable configuration.[32] This reduces abstraction layers and simplifies development, though it allows less fine-grained control than LangChain.

[27] Cf. CrewAI 2025
[28] Cf. Bedi 2025
[29] Cf. LangChain 2025b
[30] Cf. Agno 2025a
[31] Cf. LangChain 2025e; Milvus 2025
[32] Cf. Agno 2025d

Use Cases

LangChain covers a wide range of applications, including chatbots, document question answering, knowledge retrieval, and data extraction. It is particularly effective when LLMs must connect to external data sources, APIs, or tools, offering high flexibility for integration.[33]

Agno also supports diverse use cases but structures them by complexity. Simple agents can perform tasks such as financial analysis, data processing, or web scraping. More advanced workflows include automating multi-step processes like generating reports or blog content. At the highest level, Agno enables full-stack solutions that integrate agents with interfaces, databases, and backend services for production-ready applications.[34]

Technical Entry Barriers

One technical hurdle in LangChain is its modular installation. While the base package is simple to install, most integrations, such as OpenAI or Anthropic tools, require separate packages. This design provides flexibility but also adds complexity, as projects often pull in many dependencies and demand careful version management. The situation is further complicated by frequent breaking changes and unstable APIs, which reduce confidence in long-term compatibility. Developers also note that the documentation is often outdated or inconsistent, increasing the learning curve and making it harder to use the abstractions correctly. Together, these issues create significant entry barriers for new users.[35]

In contrast to LangChain, Agno offers a much simpler setup, with all core functionality, including memory, tools, reasoning, and orchestration, available in a unified package.[36] This minimal dependency footprint reduces setup friction and accelerates onboarding. Agno's documentation receives mixed feedback. Some developers find it clear and accessible, while others note gaps or insufficient detail. Once users become familiar with the framework, however, the underlying code is considered clean and relatively easy to follow, which helps reduce complexity in practical use.[37]

[33] Cf. LangChain 2025d
[34] Cf. Agno 2025b
[35] Cf. LangChain 2025c; Guda 2025
[36] Cf. Bedi 2025
[37] Cf. Maheshwari 2025

References

Agno (2025a): agno-agi/agno. original-date: 2022-05-04T15:23:02Z. URL: https://github.com/agno-agi/agno (retrieval: 08/26/2025).

Agno (2025b): Examples Gallery. URL: https://docs.agno.com/examples/introduction (retrieval: 08/26/2025).

Agno (2025c): What are Agents? URL: https://docs.agno.com/agents/introduction (retrieval: 08/26/2025).

Agno (2025d): What is Agno? URL: https://docs.agno.com/introduction (retrieval: 08/26/2025).

Bedi, Ashpreet (2025): Agno: a lightweight library for building Multi-Agent Systems. URL: https://pypi.org/project/agno/ (retrieval: 08/26/2025).

Bhavsar, Pratik (2024): Mastering RAG: How to Select an Embedding Model. URL: https://galileo.ai/blog/mastering-rag-how-to-select-an-embedding-model (retrieval: 08/27/2025).

Cooper, Aidan (2024): How to Beat Proprietary LLMs With Smaller Open Source Models. URL: https://www.aidancooper.co.uk/how-to-beat-proprietary-llms/ (retrieval: 08/27/2025).

CrewAI (2025): CrewAI Documentation – Introduction. URL: https://docs.crewai.com/en/introduction (retrieval: 08/26/2025).

Du, Hung; Thudumu, Srikanth; Vasa, Rajesh; Mouzakis, Kon (2025): A Survey on Context-Aware Multi-Agent Systems: Techniques, Challenges and Future Directions. DOI: 10.48550/arXiv.2402.01968. URL: http://arxiv.org/abs/2402.01968 (retrieval: 08/26/2025).

GeeksforGeeks (2025): RAG vs Traditional QA. URL: https://www.geeksforgeeks.org/nlp/rag-vs-traditional-qa/ (retrieval: 08/27/2025).

Greyling, Cobus (2024): Comparing LLM Agents to Chains: Differences, Advantages & Disadvantages. URL: https://cobusgreyling.medium.com/comparing-llm-agents-to-chains-differences-advantages-disadvantages-a86029a1445f (retrieval: 08/26/2025).

Guda, Shashank (2025): Challenges & Criticisms of LangChain. URL: https://shashankguda.medium.com/challenges-criticisms-of-langchain-b26afcef94e7 (retrieval: 08/26/2025).

Hugging Face (2025): Multi-Agent Systems - Hugging Face Agents Course. URL: https://huggingface.co/learn/agents-course/unit2/smolagents/multi_agent_systems (retrieval: 08/26/2025).

Juvekar, Kush; Purwar, Anupam (2024): Introducing a new hyper-parameter for RAG: Context Window Utilization. arXiv:2407.19794. DOI: 10.48550/arXiv.2407.19794. URL: http://arxiv.org/abs/2407.19794 (retrieval: 08/27/2025).

Kamath, Uday; Keenan, Kevin; Somers, Garrett; Sorenson, Sarah (2024): Large language models: a deep dive: bridging theory and practice. Cham: Springer. ISBN: 978-3-031-65647-7.

Kimothi, Abhinav (2025): A Simple Guide to Retrieval Augmented Generation. 1st ed. New York: Manning Publications Co. LLC. ISBN: 978-1-63343-585-8 978-1-63835-758-2.

Kittel, Chad; Siemens, Clayton (2025): AI Agent Orchestration Patterns. URL: https://learn.microsoft.com/en-us/azure/architecture/ai-ml/guide/ai-agent-design-patterns (retrieval: 08/26/2025).

LangChain (2025a): Agent architectures. URL: https://langchain-ai.github.io/langgraph/concepts/agentic_concepts/ (retrieval: 08/26/2025).

LangChain (2025b): Architecture | LangChain. URL: https://python.langchain.com/docs/concepts/architecture/ (retrieval: 08/26/2025).

LangChain (2025c): How to install LangChain packages. URL: https://python.langchain.com/docs/how_to/installation/ (retrieval: 08/26/2025).

LangChain (2025d): LangChain. URL: https://www.langchain.com/langchain (retrieval: 08/26/2025).

LangChain (2025e): Workflows & agents. URL: https://langchain-ai.github.io/langgraph/tutorials/workflows/ (retrieval: 08/26/2025).

Maheshwari, Harsh (2025): Agno vs. Pydantic AI: The Ultimate Showdown for Building AI Agents. URL: https://hrshdg8.medium.com/agno-vs-pydantic-ai-the-ultimate-showdown-for-building-ai-agents-79b2c975cbec (retrieval: 08/26/2025).

Microsoft (2025): Tool Use Design Pattern – AI Agents for Beginners. URL: https://microsoft.github.io/ai-agents-for-beginners/04-tool-use/ (retrieval: 08/26/2025).

Microsoft Research (2025): AutoGen: Open-Source Framework for Agentic AI. URL: https://www.microsoft.com/en-us/research/project/autogen/ (retrieval: 08/26/2025).

Milvus (2025): What is the difference between LangChain and other LLM frameworks? URL: https://milvus.io/ai-quick-reference/what-is-the-difference-between-langchain-and-other-llm-frameworks (retrieval: 08/26/2025).

Patronus AI (2025): AI Agent Routing: Tutorial & Best Practices. URL: https://www.patronus.ai/ai-agent-development/ai-agent-routing (retrieval: 08/26/2025).

Shamim, Bhuiyan; Isachenko, Timur (2025): Generative AI with local LLM. Leanpub. URL: https://leanpub.com/quickstartwithai (retrieval: 04/18/2025).

Shen, Michael; Umar, Muhammad; Maeng, Kiwan; Suh, G. Edward; Gupta, Udit (2024): Towards Understanding Systems Trade-offs in Retrieval-Augmented Generation Model Inference. arXiv:2412.11854. DOI: 10.48550/arXiv.2412.11854. URL: http://arxiv.org/abs/2412.11854 (retrieval: 08/27/2025).

Yu, Puxuan; Merrick, Luke; Nuti, Gaurav; Campos, Daniel (2024): Arctic-Embed 2.0: Multilingual Retrieval Without Compromise. arXiv:2412.04506 [cs]. DOI: 10.48550/arXiv.2412.04506. URL: http://arxiv.org/abs/2412.04506 (retrieval: 04/18/2025).

Zhao, Shengming; Shao, Yuchen; Huang, Yuheng; Song, Jiayang; Wang, Zhijie; Wan, Chengcheng; Ma, Lei (2025): Understanding the Design Decisions of Retrieval-Augmented Generation Systems. arXiv:2411.19463. DOI: 10.48550/arXiv.2411.19463. URL: http://arxiv.org/abs/2411.19463 (retrieval: 08/27/2025).