Design of On-Premises Version of RAG with AI Agent for Framework Selection Together with Dify and DSL as Well as Ollama for LLM

Kohei Arai
Department of Science and Engineering, Saga University, Saga City, Japan

***Abstract*—Currently, most RAGs are cloud-based and include Bedrock. However, there is a trend to return from the cloud to on-premises due to security concerns. In addition, it is common for APIs to call Lambda or EC2 for data access, but it is not easy to select the optimal framework depending on the data attributes. For this reason, the author devised a system for selecting the optimal framework using an AI agent. Furthermore, the author decided to use Dify, which is based on a DSL, as the user interface for the on-premises version of RAG, and ollama as a large-scale language model that can be installed on-premises as well. The author also considered the specifications of the hardware required to build this RAG and confirmed the feasibility of implementation.**

Keywords*—*RAG (Retrieval-Augmented Generation); API (Application Programming Interface); Lambda; EC2 (Amazon Elastic Compute Cloud); AI agent; Dify; DSL (domain specific language); ollama; YAML (YAML Ain't Markup Language)

I. Introduction

The name RAG (Retrieval-Augmented Generation) and its specific methodology were proposed in 2020. In a paper published by researchers from Facebook AI Research (now Meta AI), University College London, and New York University, RAG was introduced as a "general-purpose fine-tuning recipe"[1].

RAG research ran on a cluster of NVIDIAS GPUs[2] and demonstrated how to make generative AI models more reliable. The research aimed to link generative AI services with external resources, especially those containing the latest technical details.

The concept of RAG has spread rapidly since its introduction and is now adopted in hundreds of papers and many commercial services. This technology has significantly improved the capabilities of large-scale language models (LLMs) and taken question answering systems to a new level.

The development of RAG is the culmination of many years of research in information retrieval and natural language processing and has become an important part of modern AI technology. It is expected that this technology will continue to evolve and contribute to improving the performance and reliability of AI systems.

Research issues related to RAG play an important role in the development and practical application of this technology. Some of the main research issues are listed below.

1) Chunking and embedding: The performance of RAG systems depends heavily on the chunking and embedding methods used. Optimal chunking methods: Research is needed to find effective ways to segment documents and extract relevant information appropriately [1]. Embedding multimedia content: Research is needed to find effective ways to embed non-text data (images, audio, video, etc.).

2) Comparison of RAG and fine-tuning: RAG and fine-tuning of LLMs (large-scale language models) take different approaches. Performance comparison: Research is needed to systematically compare the performance of both methods under various tasks and conditions. Cost-effectiveness: A comparative analysis of the costs of implementation and operation is also an important issue. Applicability: Research is needed to clarify in what situations and applications RAG and fine-tuning are suitable.

3) Testing and monitoring of RAG systems: Quality assurance and continuous improvement of RAG systems are important research topics. Establishment of performance evaluation indicators: It is necessary to develop indicators to properly evaluate the performance of RAG systems. Monitoring methods during operation: Research is required on methods to continuously monitor and improve system performance in actual usage environments.

4) Context optimization: Proper context management is essential for improving the performance of RAG systems. Optimization of chunk size: Research on the optimal chunk size is required, as larger contexts may produce better results. Balance with token restrictions: Optimization research is required that takes into account the trade-off with LLM token restrictions and latency.

5) Efficiency and cost reduction: Improvements in efficiency and cost are important for practical use of RAG systems. Semantic cashing: Research is being conducted into methods to reduce the number of LLM calls and reduce costs and latency by cashing frequent queries and their answers.

[1] https://blogs.nvidia.co.jp/2023/11/17/what-is-retrieval-augmented-generation/

[2] https://en.wikipedia.org/wiki/List_of_Nvidia_graphics_processing_units

Utilization of open-source models: For small texts, open-source sentence embedding models may perform as well as commercial models, and research in this area is progressing.

In this paper, open-source models utilizing RAG are investigated and designed to enhance cost-effectiveness. Currently, many RAG systems are cloud-based and use services such as Amazon Bedrock[3]. However, due to security concerns, there is a trend of moving from the cloud to on-premises environments. Traditionally, data access was typically achieved through API calls via AWS Lambda or EC2[4], but it was not easy to select the optimal framework according to the data attributes.

To address this challenge, the author devised a system that utilizes an AI agent to select the optimal framework. In addition, they adopted DSL[5] (Domain Specific Language)-based Dify[6] as the user interface for the on-premises version of RAG, and selected ollama[7] as a large-scale language model that can be deployed on-premises. In this paper, ollama llama3.2 is used.

The author also considered the specifications of the hardware required to build this RAG system and confirmed the feasibility of implementation. This made it possible to build an efficient and flexible RAG system while placing emphasis on security.

The following section described the related research works on RAG followed by proposed RAG system. Then, software and hardware requirements are described. After that, the conclusion is described with some remarks and discussions.

II. Related Research

There are the following RAG related papers, Retrieval-augmented generation for knowledge-intensive NLP (Natural Language Processing) tasks is proposed. Natural Language Processing (NLP) tasks include sentiment analysis, entity recognition, text summarization, machine translation, speech recognition, text classification, chatbot interaction, keyword extraction, question answering, part-of-speech tagging, topic modeling, predictive text, conference resolution, and spam detection; essentially, any activity where a computer analyzes and understands human language to perform a specific function. This paper is the first to propose the RAG model, an approach that improves performance on knowledge-based tasks by integrating a retrieval component into a generative model [2].

Improving zero-shot generalization in text classification using retrieval-augmented language models are proposed. In this study, the authors applied RAG to text classification tasks, aiming to improve zero-shot generalization performance. The authors showed that using a retrieval component improves classification accuracy for unseen classes [3].

RAG for knowledge-intensive NLP tasks is discussed. The paper provides a detailed description of RAG architecture and evaluates its performance on a variety of tasks, showing that it performs particularly well on knowledge-based tasks such as question answering and sentence generation [4].

Dense passage retrieval for open-domain question answering is proposed. In this study, a density-based passage retrieval method is proposed for open-domain question answering tasks, which is often used as the retrieval component of the RAG model to improve the accuracy of question answering [5].

Other than these, there are the following recent research works,

Self-RAG: Learning to Retrieve, Generate, and Critique through Self-Reflection is proposed which appears in the URL of https://arxiv.org/abs/2310.11511, [6].

Atlas: Few-shot Learning with Retrieval Augmented Language Models are proposed which appears in the URL of https://arxiv.org/abs/2208.03299, [7].

Internet-Augmented Dialogue Generation is also proposed which appears in the URL of https://arxiv.org/abs/2107.07566, [8].

REPLUG: Retrieval-Augmented Black-Box Language Models is proposed which appears in the URL of https://arxiv.org/abs/2301.12652, [9].

Dense Passage Retrieval for Open-Domain Question Answering is proposed which appears in the URL of https://arxiv.org/abs/2004.04906, [10].

Realm: Retrieval-Augmented Language Model Pre-Training is proposed which appears in the URL of https://arxiv.org/abs/2002.08909, [11].

Improving language models by retrieving from trillions of tokens are proposed which appears in the URL of https://arxiv.org/abs/2112.04426, [12].

Query2doc: Query Expansion with Large Language Models are also proposed which appears in the URL of https://arxiv.org/abs/2303.07678, [13].

Chain-of-Note: Enhancing Robustness in Retrieval-Augmented Language Models are proposed which appears in the URL of https://arxiv.org/abs/2311.09210, [14].

On the other hand, Dify related research works are as follows,

The literature on data integration in general, DSLs, and Dify is as follows: "A Survey of Data Integration Systems" This paper provides an overview of data integration systems and various approaches [15].

"Domain-Specific Languages: An Annotated Bibliography" This paper provides an overview of DSLs and examples of various amana DSLs [16].

[3] https://docs.aws.amazon.com/ja_jp/bedrock/latest/userguide/what-is-bedrock.html

[4] https://www.serverless.direct/post/aws-lambda-vs-ec2-which-one-to-choose-for-your-app

[5] https://en.wikipedia.org/wiki/Domain-specific_language

[6] https://docs.dify.ai/ja-jp/guides/application-orchestrate/creating-an-application

[7] https://ollama.com/library/llama3.2

"Interactive Data Integration with User-Centric Approaches" This paper describes user-centric data integration approaches [17].

"VLDB Conference Proceedings" The latest research results on data integration are presented at the VLDB conference [18].

"SIGMOD Conference Proceedings" Research results on data integration and DSLs are also presented at the SIGMOD conference [19].

"Technical Report: Data Integration Using DSLs" Some research institutes have published technical reports on data integration using DSLs [20].

"Data Integration: A Theoretical Perspective" This book provides a detailed explanation of the theoretical background of data integration [21].

"Domain-Specific Languages in Action" This book gives practical examples of the application of various DSLs [22].

GitHub Repository: Some open-source projects publish code for data integration tools using DSLs [23].

III. Proposed RAG System

A. System Configuration

The on-premises version of RAG is intended to be created. The author devised a system that utilizes an AI agent to select the optimal framework. In addition, they adopted DSL-based Dify as the user interface for the on-premises version of RAG, and selected ollama as a large-scale language model that can be deployed on-premises.

Fig. 1 shows the block diagram of the proposed RAG.

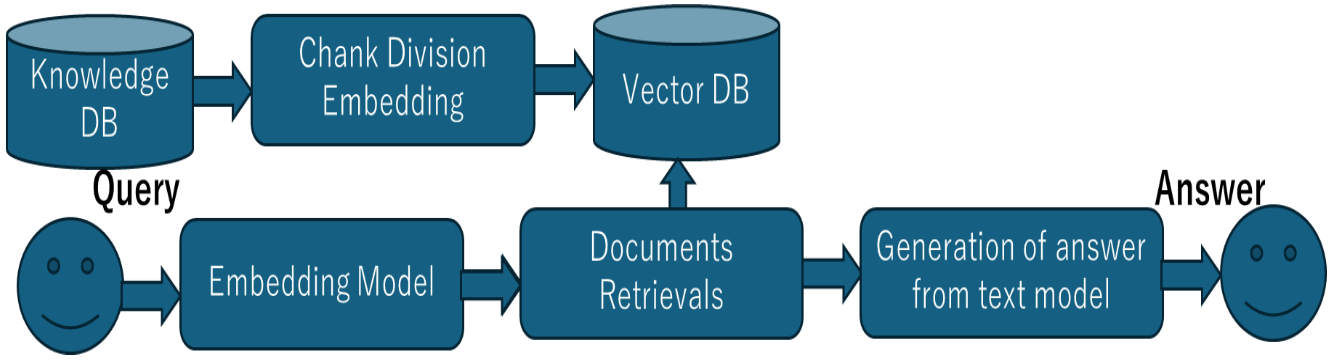

Fig. 1. Block diagram of the proposed RAG system.

The process from query to answer generation in the RAG system goes through the following steps:

1) Query processing: The system receives a question or task (query) from the user and converts it into a format that the system can easily understand. The system uses natural language processing technology to analyze the meaning of the query. Converts it into a format suitable for search.

2) Information retrieval: Based on the converted query, the system searches the knowledge base for relevant information. Query vectorization: Converts the question into a numerical expression (vector). Similarity calculation: Calculates the similarity between the query vector and the vector of information in the knowledge base. Ranking: Ranks the most relevant information based on the similarity.

3) Context generation: Based on the information obtained from the search results, the system generates a context related to the question.

4) Answer generation: A generative model (usually a large-scale language model) takes the query and context as input and generates an appropriate answer. Integrates the given information and then creates an answer in natural language.

5) Post-processing: The generated answer is further processed to be formatted into the final output format. Formatting the answer and filtering inappropriate content as well as adding additional information where necessary (e.g. citing sources). After that, adjust the length and complexity of the answer.

6) Output: The final answer is presented to the user. Answer in text form and possibly include relevant images, links, and/or additional references.

7) Feedback and learning: Many RAG systems have mechanisms for collecting user feedback and continually improving the system's performance. Recording user responses (e.g. clicking the "helpful" button) and evaluating the accuracy and relevance of answers as well as tweaking search algorithms and generative models based on feedback.

As for the user interface, Dify is featured as follows:

Dify has all the features to develop AI apps. It also supports hundreds of AI language models, allowing creating custom chatbots with intuitive operations. It also has a high-performance and flexible RAG engine, allowing building AI agents to use a variety of tools. It can also be freely designing workflows.

Dify Studio[8] offers three ways to create an application: It can be used on an application template. It can start with a blank application. You can create one (locally or online) by importing a DSL file. To get a quick overview of the types of applications

[8] https://note.com/jolly_dahlia842/n/n41bf7cf085fd

you can create with Dify, select "Studio" from the navigation menu, then "Create from Template" from the application list.

When creating the first application with Dify, it is important to understand the basic concepts behind the four different types of applications: chat voice, text generators, agents, and workflows. Also, when creating an application, give it a name, choose an appropriate icon, and briefly describe the purpose of this application to make it easier to use within your team.

Dify DSL is a standard file format (YML)[9] for AI application development defined by Dify.AI. This standard includes basic information about the application, model parameters, orchestration settings, etc. Firstly, import local DSL files. Then if a DSL file (template) exists, provided by the community or others, select "Import DSL file" from Studio. After importing, the original application settings will be loaded directly. After that, import a DSL file via URL. It can import a DSL file via URL using the following format: https://example.com/your_dsl.yml

DSL is a specialized language designed to efficiently build and configure AI applications on the Dify platform. The features of Dify DSL are: YAML format[10]: Dify DSL is written in YAML format, Workflow definition: It can concisely define the workflow and settings of your application and sharing and reuse: It can easily share and reuse the workflow you created by exporting and importing it as a DSL file.

To use DSL, it can import it from the Dify dashboard by selecting "Create an app" and then "Import DSL file". Then select the YAML file and click the "Create" button. Customize it: After importing, it can adjust the workflow and settings as needed. Finally, export it: it can export the workflow it created as a DSL file and share it with other users.

It can easily create chatbots specialized for specific industries or purposes using knowledge bases. This application can be developed without programming expertise by combining Dify's no-code interface with the flexibility of DSL. In addition, it can export and import the workflow it created as a DSL file, making it easy to share and reuse within a team or community.

B. *Ollama of LLM*

Next, the author will explain ollama, a small-scale LLM that can run on-premises on a local machine. It has the following features:

Local execution: With ollama, you can run LLM on your own computer without an Internet connection.

Multiplatform support: Currently it supports Mac and Linux and will support Windows in the future.

Various models: It supports large-scale language models such as Llama. Users can select and use various models.

Easy environment construction: It is relatively easy to install and configure, and you can easily obtain and run models from the command line.

Python integration: ollama can be used from Python environments using the ollama-python library. This allows you to leverage local LLMs through APIs and integrate them into RAGs and agents.

Customizability: Since it is open source, users can customize it to suit their needs.

Security: Since it runs locally, it is suitable for projects that value data privacy and security.

Ollama is a powerful tool for individuals and businesses to effectively leverage AI technology, allowing them to run advanced natural language processing tasks such as text generation, question answering, and text summarization in a local environment.

To install ollama, follow the steps below:

Download the installer from the ollama official website.

Open the downloaded file and follow the instructions to install.

Once the installation is complete, a llama icon will appear in the taskbar on Windows.

Download and run the model

Open Command Prompt (Windows) or Terminal (Mac/Linux).

Run the following command to download and run the model:

```
bash

ollama run gemma2:2b
```

This command downloads and runs the Gemma 2b model. If the model is not already available locally, it will be downloaded automatically.

To interact with the AI, once the model is running, a prompt will appear. Enter users' message and press Enter, and the AI will respond.

Users can customize and manage your models using the following commands:

ollama create: Create a custom model

ollama show: Show model information

ollama list: Show a list of installed models

ollama rm: Delete a model

C. *Python Code for Knowledgebase Creation*

In this example we'll create a simple knowledge base with a dictionary structure and show how to perform some basic operations. Python code for creation of Knowledgebase System is as follows,

```
python

class KnowledgeBase:
```

[9] https://docs.dify.ai/guides/application-orchestrate/creating-an-application

[10] https://spacelift.io/blog/yaml

```
def __init__(self):
self.knowledge = {}
def add_fact(self, key, value):
""Add a new fact to the knowledge base"""
self.knowledge[key] = value
def get_fact(self, key):
""Get a fact from the knowledge base"""
return self.knowledge.get(key, "No information found")
def update_fact(self, key, value):
""Update an existing fact"""
if key in self.knowledge:
self.knowledge[key] = value
return True
return False
def remove_fact(self, key):
""Remove a fact from the knowledge base"""
if key in self.knowledge:
del self.knowledge[key]
return True
return False
def list_all_facts(self):
""List all facts in the knowledge base"""
return self.knowledge
# Knowledge Base Usage Example
if __name__ == "__main__":
kb = KnowledgeBase()
# Add fact
kb.add_fact("Python", "High-level programming language")
kb.add_fact("AI", "Artificial intelligence")
kb.add_fact("ML", "Machine learning")
# Get fact
print(kb.get_fact("Python")) # Output: High-level programming language
print(kb.get_fact("Database")) # Output: No information found
# Update fact
kb.update_fact("AI", "Artificial intelligence technology")
print(kb.get_fact("AI")) # Output: Artificial intelligence technology
# Remove fact
kb.remove_fact("ML")
# List all facts
print(kb.list_all_facts())
```

This simple implementation can be extended as follows:

1) *Persistence:* Use a database (e.g. SQLite) to store the knowledge base and maintain the information even after the program ends.

2) *Complex data structures:* Store objects or structured data instead of simple strings.

3) *Search capabilities:* Implement advanced search capabilities using keyword searches or regular expressions.

4) *Version control:* Add the ability to track changes to each fact.

5) *Relationship expression:* Be able to express relationships between facts (e.g. a graph database-like approach).

6) *Inference engine:* Implement a simple inference function to derive new facts from existing facts.

D. Required Hardware Specifications

The hardware specifications for building a RAG using ollama are as follows.

1) *CPU*

a) *Best choice:* Intel CPU of 11th generation or later that supports the AVX512 instruction set, or AMD CPU based on Zen4. An AMD CPU based on the "Zen 4" architecture is the first AMD processor to support the AVX-512 instruction set, meaning if you're looking for an AMD CPU with AVX-512 capability, you should choose one based on the Zen 4 microarchitecture.

b) *Reason:* To speed up the matrix calculations required for AI models

c) *Minimum requirement:* Any CPU that supports the AVX instruction set will work

2) *RAM*

a) *Recommendation:* 16GB or more

b) *Reason:* To comfortably run models with 7B parameters

c) *Minimum requirement:* May work with around 8GB

3) *Storage*

a) *Recommendation:* 50GB of freer space

b) *Breakdown:* Docker container (2GB+), model file, vector store, etc.

4) *GPU*

a) *Recommended:* Equipped with NVIDIA GPU (e.g. GTX 1080 Ti or higher)

b) *Reason:* Can significantly speed up model inference

c) *Not required:* Can be run with CPU only, but processing speed will be reduced.

In this connection, GIGABYTE AORUS GeForce GTX 1080 Ti 11GB Video Card - GV-N108TAORUS-11GD[11] is one of the candidates.

Other points to consider are that if you are using larger models (13B or more), a high-performance GPU and 32GB or more of RAM are recommended. Users also need to consider the amount of VRAM (depending on the model size and quantization level). ollama can run on relatively lightweight systems, but to get comfortable user experience, it is recommended that you use the above recommended specifications as a guide. Especially in the RAG system, it is desirable to have a generous specification because building and searching the vector store also requires resources.

E. AI Agent

Although the definition of an AI agent may vary slightly depending on the technical field, it generally refers to software that interacts with its environment, collects data, and autonomously executes tasks based on that data to achieve a specific goal. In particular, our focus here is on AI agents based on LLMs. Using an AI agent as the user interface of the RAG system is a good approach to achieve a more flexible and advanced conversational interface. Below are some recommended implementation ideas.

1) Multi-agent system: This method uses a combination of multiple specialized agents.

a) Triage agent: Analyzes the user's question and assigns it to the appropriate specialized agent.

b) Search agent: Responsible for the search function of the RAG system to retrieve related information.

c) Answer generation agent: Generates an appropriate answer based on the search results.

d) Dialogue management agent: Manages the flow of dialogue with the user and asks additional questions as necessary.

In this method, each agent specializes in a specific role, allowing complex tasks to be handled efficiently.1.

2) Plan-and-Execute agent: This method is performed by a single advanced agent that plans and executes the plan.

Analyzes the user's question and plans the steps required to answer it.

Based on the plan, it sequentially searches the RAG system, integrates information, generates answers, etc.

The plan is revised as necessary to generate the final answer.

This method is particularly effective when complex questions or multi-step processing are required.1.

3) Conversational RAG Agent: An agent that collects information through dialogue with the user and gradually refines its answer.

It first provides a concise answer to the user's initial question.

It then asks the user if additional information or clarification is needed.

Based on the user's response, it re-uses the RAG system to complement the information and expand the answer.

This method allows for flexible information provision tailored to the user's needs.

4) Self-improving RAG Agent: An agent system that incorporates a feedback loop.

After answering the user's question, it asks for feedback on the quality and appropriateness of the answer.

Based on the feedback, it automatically adjusts how it generates search queries and constructs answers.

Continuous learning improves performance over time.

This method is particularly effective in long-term use, allowing the system to continually improve its accuracy and usefulness.

Implementation Considerations:

1) Model selection: Using high-performance models such as GPT-4 allows for more sophisticated dialogue and accurate information processing.

2) Context management: It is important to properly manage long-term dialogue history and maintain consistent dialogue.

3) Error handling: Users need to implement a way to handle cases when the agent cannot respond appropriately and provide appropriate feedback to the user.

4) Security and privacy: Be careful with user data, filtering and anonymizing information where necessary.

By using these recommendations as a starting point and customizing them for specific use cases and requirements, it can implement an AI agent as an effective RAG user interface.

F. Swarm AI Agent

Swarm is a framework for multi-agent orchestration released by OpenAI on October 12, 2024[12]. This framework uses Python and is designed to enable AI agents to work together and autonomously complete complex tasks. By using Swarm, it becomes easy to build multi-agent systems.

Another library for building AI agents is LangGraph[13]. LangGraph is feature-rich and highly flexible but tends to be complicated to implement. In contrast, Swarm has fewer features but is very easy to implement.

Table I shows the classes for creating AI agents. Users can set the agent's name, behavior, model to be used, etc. client.run() is a function to execute the created agent. The arguments of this function are shown in Table II.

It processes messages for the agent and advances conversation. Furthermore, run_demo_loop() is a function to

[11] https://www.gigabyte.com/jp/Graphics-Card/GV-N108TAORUS-11GD#kf

[12] https://github.com/openai/swarm

[13] https://langchain-ai.github.io/langgraph/

repeatedly run the created agent on the console as shown in Table III. It uses client.run() internally to run the agent.

TABLE I. THE CLASSES FOR CREATING AI AGENTS

Field_Name	Type	Default	Description
name	str	"Agent"	Name_of_the_agent
model	str	"gpt-4o"	AI_model_to_use
instruction	str	You_are_a_helpful_agent.	Instructions_to_the_agent
functions	List	[]	List_of_functions_available_to_the_agent
tool_choice	str	None	Specific_tool_to_be_used_by_the_agent

TABLE II. THE ARGUMENT FOR CLIENT.RUN ()

Argument_Name	Type	Initial_Value	Description
agent	Agent	Required	Initial_agent_to_be_called
messages	List	Required	List_of_message_objects
context_variables	dict	{}	Context_with_additional_information
max_turns	int	float("inf")	Maximum_number_of_turns_in_conversation
model_override	str	None	Option_to_change_model
stream	bool	False	Whether_to_show_streaming_responses
debug	bool	False	Debug_mode

TABLE III. THE ARGUMENT FOR RUN_DEMO_LOOP ()

Argument_Name	Type	Initial_Value	Description
starting_agent	Agent	Required	Initial_agent_to_be_called
context_variables	dict	{}	Context_containing_additional_information
stream	bool	False	Whether_to_display_the_response_in_streaming_mode
debug	bool	False	Debug_mode

IV. CONCLUSION

On-premises version of RAG is proposed for secure reasons. The proposed RAG utilizes Swarm-based AI agent which allows easy to select the optimal framework depending on the data attributes. Furthermore, the Dify which is based on a DSL is used as the user interface for the on-premises version of RAG. Also, ollama is used as a large-scale language model that can be installed on-premises as well.

Other than that, the specifications of the hardware required to build this RAG and confirmed the feasibility of implementation. It is confirmed that the proposed RAG system can be created with just one PC with a GPU card.

V. FUTURE RESEARCH WORKS

Although it is confirmed that the proposed RAG system can be feasible, knowledgebase system is not being developed. There are so many applications of the proposed RAG system. Therefore, one of the business use cases will be attempted in the near future.

ACKNOWLEDGMENT

The author would like to thank Prof. Dr. Hiroshi Okumura and Prof. Dr. Osamu Fukuda of Saga University for their valuable comments and suggestions.

REFERENCES

[1] Scott Barnett, Stefanus Kurniawan, Srikanth Thudumu, Zach Brannelly, Mohamed Abdelrazek, Seven Failure Points When Engineering a Retrieval Augmented Generation System, URL : https://arxiv.org/abs/2401.05856, Applied Artificial Intelligence Institute, Geelong, Australia, 2024.

[2] Lewis, P., Perez, E., Pott, C., & Riedel, S., "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks" by Patrick Lewis et al., Retrieval-augmented generation for knowledge-intensive NLP tasks. In Proceedings of the 34th International Conference on Neural Information Processing Systems (NIPS 2020) (pp. 1–12), 2020.

[3] Sap, M., Lourie, N., & Riedel, S., "Improving Zero-Shot Generalization in Text Classification using Retrieval-Augmented Language Models" by Maarten Sap et al., Improving zero-shot generalization in text classification using retrieval-augmented language models. In Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing (EMNLP 2021) (pp. 1–12), 2021.

[4] Riedel, S., Lewis, P., & Perez, E. (2020). RAG: Retrieval-augmented generator for knowledge-intensive nlp tasks. arXiv preprint arXiv:2005.11401., "RAG: Retrieval-Augmented Generator for Knowledge-Intensive NLP Tasks" by Sebastian Riedel et al., 2005.

[5] Karpukhin, V., Oğuz, B., Min, S., Wu, L., Edunov, S., Chen, D., & Yih, W. T. (2020). Dense passage retrieval for open-domain question answering. In Proceedings of the 34th International Conference on Neural Information Processing Systems (NIPS 2020) (pp. 1–12)., "Dense Passage Retrieval for Open-Domain Question Answering" by Vladimir Karpukhin et al.,

[6] Akari Asai, et al., "Self-RAG: Learning to Retrieve, Generate, and Critique through Self-Reflection", https://arxiv.org/abs/2310.11511, 2023.

[7] Gautier Izacard, et al., "Atlas: Few-shot Learning with Retrieval Augmented Language Models", https://arxiv.org/abs/2208.03299, 2023.

[8] Mojtaba Komeili, et al., "Internet-Augmented Dialogue Generation", https://arxiv.org/abs/2107.07566, 2022.

[9] Weijia Shi, et al., "REPLUG: Retrieval-Augmented Black-Box Language Models", https://arxiv.org/abs/2301.12652, 2023.

[10] Vladimir Karpukhin, et al., "Dense Passage Retrieval for Open-Domain Question Answering", EMNLP 2020, https://arxiv.org/abs/2004.04906, 2020.

[11] Kelvin Guu, et al., "Realm: Retrieval-Augmented Language Model Pre-Training", ICML 2020, https://arxiv.org/abs/2002.08909, 2020.

[12] Sebastian Borgeaud, et al., "Improving language models by retrieving from trillions of tokens", ICML 2022, https://arxiv.org/abs/2112.04426, 2022.

[13] Zhuyun Dai, et al., "Query2doc: Query Expansion with Large Language Models", https://arxiv.org/abs/2303.07678, 2023.

[14] Wenhao Yu, et al., "Chain-of-Note: Enhancing Robustness in Retrieval-Augmented Language Models", https://arxiv.org/abs/2311.09210, 2023.

[15] Halevy, A. Y., et al. "Enterprise information integration: successes, challenges and controversies." Proceedings of the 2005 ACM SIGMOD international conference on Management of data. 2005.

[16] Mernik, M., Heering, J., & Sloane, A. M. "When and how to develop domain-specific languages." ACM Computing Surveys (CSUR) 37.4 (2005): 316-344, 2005.

[17] Sarma, A. D., et al. "Interactive data integration." Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data. 2010, 2010.

[18] International Conference on Very Large Data Bases (VLDB). VLDB Conference Website, https://vldb.org/2024/, accessed on 19 December 2024.

[19] ACM SIGMOD International Conference on Management of Data. SIGMOD Conference Website,

https://dl.acm.org/doi/proceedings/10.1145/3626246, accessed on 19 December 2024.

[20] Technical report series of research institutes (e.g. MIT CSAIL Technical Reports), https://libguides.mit.edu/c.php?g=176306&p=1159542, accessed on 19 December 2024.

[21] Lenzerini, M. "Data integration: a theoretical perspective." ACM SIGMOD Record 33.3 (2004): 66-73, 2004.

[22] Fowler, M. "Domain-specific languages." Addison-Wesley Professional, 2010.

[23] GitHub (e.g. Apache NiFi, Apache Beam). https://github.com/apache/beam, accessed on 19 December 2024.

Author's Profile

Kohei Arai, He received BS, MS and PhD degrees in 1972, 1974 and 1982, respectively. He was with The Institute for Industrial Science and Technology of the University of Tokyo from April 1974 to December 1978 also was with National Space Development Agency of Japan from January 1979 to March 1990. During from 1985 to 1987, he was with Canada Centre for Remote Sensing as a Post-Doctoral Fellow of National Science and Engineering Research Council of Canada. He moved to Saga University as a Professor in Department of Information Science in April 1990. He was a councilor for the Aeronautics and Space related to the Technology Committee of the Ministry of Science and Technology during from 1998 to 2000. He was a councilor of Saga University for 2002 and 2003. He also was an executive councilor for the Remote Sensing Society of Japan for 2003 to 2005. He is a Science Council of Japan Special Member since 2012. He is an Adjunct Professor at Brawijaya University. He also is an Award Committee member of ICSU/COSPAR. He also is an adjunct professor of Nishi-Kyushu University and Kurume Institute of Technology Applied AI Research Laboratory. He wrote 119 books and published 728 journal papers as well as 569 conference papers. He received 98 of awards including ICSU/COSPAR Vikram Sarabhai Medal in 2016, Science award of Ministry of Mister of Education of Japan in 2015 and so on. He is now Editor-in-Chief of IJACSA and IJISA. http://teagis.ip.is.saga-u.ac.jp/index.html