Prompt Start: A Framework Optimizing Prompt Initialization for Developers

LEANDRO DUARTE KUSUMASTUTI CAHYANINGRUM
leandrod | kcah @kth.se

December 29, 2024

Abstract

Large Language Models (LLMs) are increasingly integral to modern software applications, enhancing both user interactions and internal processes. Despite their potential, developers often face the "Blank Page Problem," which arises when they first encounter the task of selecting the optimal prompt design and crafting the initial prompt text. Our research introduces a framework designed to assist developers in identifying the best prompt design techniques for their specific use cases in LLMs-based applications. By leveraging a Retrieval-Augmented Generation (RAG) system, our framework offers a recommended prompt design approach, an initial prompt proposal, and detailed explanations supported by the latest State Of The Art (SOTA) papers, GitHub repositories, and expert blogs. This approach aims to streamline the prompt creation process, reducing the time developers spend on research and decision-making. We anticipate that developer feedback, application improvements, and performance comparisons will demonstrate the framework's effectiveness. All code, demonstrations, and datasets will be made available at https://github.com/Leandr0Duar7e/II2202_PromptStart_Kotaemon.

Contents

1 Introduction **3**

2 Theoretical Framework **4**
2.1 Retrieval-Augmented Generation (RAG) 5
2.2 Related Work 5

3 Research Question and Hypothesis **6**

4 Research Methodology **6**
4.1 Framework Architecture 7
4.2 Designing Process 8

5 Results and Analysis **9**

6 Discussion **10**

7 Conclusion and Future Work **11**

List of Acronyms and Abbreviations

CoT Chain-of-Thought

GenAI Generative Artificial Intelligence

LLM Large Language Model

LM Language Model

NLP Natural Language Processing

RAG Retrieval-Augmented Generation

SOTA State Of The Art

1 Introduction

Generative Artificial Intelligence (GenAI) has been evolving rapidly. Since the seminal work ”Attention is All You Need” [1], which revolutionized Natural Language Processing (NLP), to the widespread adoption of ChatGPT in 2022, driven by advancements in LLMs, the field has seen remarkable progress. The OpenAI GPT series exemplifies this evolution, currently achieving human-level capabilities in numerous tasks. As LLMs scale in terms of training data and parameters, and computational resources grow, the models exhibit emergent abilities [2], making them powerful tools for developers across various applications, regardless of the provider company.

The rise of LLMs has spurred interest in prompt design to optimize their outputs. Techniques like Few-shot prompting and Chain-of-Thought (CoT) prompting [3] have become standard practices. Building on these, newer methods such as ReAct [4], Reflexion [5], Self-Refine [6], and Tree-of-Thoughts [7] have further enhanced task-solving capabilities by incorporating feedback loops and external tools. These innovations have culminated in the creation of Agentic Workflows, which integrate multiple LLM calls, prompts, and iterations before providing the final output, tackling complex tasks and achieving near-human performance. Notable examples include MetaGPT [8], ChatDev [9], and AutoGen [10].

In implementing these methods, developers follow a *prompt engineering process*, iteratively refining prompts to optimize outputs. Significant work, such as DSPy [11], has been done to automate the Prompt Engineering task. While it remains complex and challenging to implement, DSPy and similar projects are continuously evolving, supported by a wealth of research and papers.

However, the problem we address in this research is distinct and precedes the prompt engineering process. It focuses on the *initial prompt design process*. Beyond the prompt design techniques already mentioned, numerous others provide equally high-quality outputs. A large research team, including members from OpenAI, Microsoft, Stanford, and other institutions [12], analyzed 1565 papers and identified 58 different text-based prompt design techniques. This abundance of techniques and resources presents a significant challenge: each time there is a need to start a prompt engineering process, developers are confronted with overwhelming options. Then, they must choose between investing substantial time in research to find optimal methods or relying on intuition, potentially missing more effective approaches that could better serve their applications.

The motivation for our work is clear: addressing this bottleneck at the beginning of the design process is critical. By improving the initial prompt design phase, we can enable developers to make informed, high-quality decisions more efficiently, leading to more robust and effective applications downstream.

```
response = llm.ChatCompletion.create(
    model="llm",
    messages=[
        {"role": "system", "content": " "}
    ] )
```

Listing 1: Example code illustrating the Blank Page Problem

Listing 1 epitomizes the ”Blank Page Problem,” where developers grapple with the initial creation of a prompt before any iterative refinement.

Our research introduces a framework to streamline this process, providing developers with a reliable method to select the most suitable prompt design for their use case. The core of our framework is an Agentic RAG system [13], which combines retrieval-based methods with LLMs and agentic workflows to deliver optimal solutions. This system leverages a knowledge base comprising influential papers, reputable GitHub repositories, and model-specific documentation, guiding developers in choosing the best approach for their tasks.

To build this framework, we utilize three open-source projects: PaperQA [14], RagBuilder [15], and Kotaemon [16]. These tools are instrumental in searching research papers, configuring RAG systems, and developing agentic RAG systems, respectively.

The subsequent sections of this report will delve into the theoretical background, methodologies, and expected outcomes of our work.

2 Theoretical Framework

This research is grounded in the capabilities of LLMs and aims to optimize their use through innovative prompting and retrieval techniques. At the core of this study is the concept of prompting, which involves crafting specific input texts to guide LLMs in generating desired outputs. Effective prompting is crucial as it directly influences the performance and applicability of LLMs across various tasks.

Schulhoff et al. [12] provide a comprehensive taxonomy of prompting techniques, offering a structured understanding essential for developers and researchers navigating the diverse landscape of LLM applications. This taxonomy identifies six key types of prompting techniques: (1) zero-shot prompting, which relies solely on task instructions without examples; (2) few-shot prompting, which includes example input-output pairs; (3) thought generation techniques like CoT and Tree-of-Thought, which break down complex problems into sequential steps; (4) decomposition prompting, which divides tasks into manageable sub-tasks; (5) self-criticism prompting, which enables iterative refinement; and (6) ensembling prompting, which combines multiple outputs for improved accuracy. The paper [12] serves as an excellent starting point for anyone entering the field, providing insights into recent concepts and ide

Prompt design techniques are methodologies employed to elicit specific behaviors and outputs from LLMs. Early techniques like Few-Shot Learning involve providing input-output examples within prompts to guide the model's responses, while CoT prompting encourages LLMs to reason through problems step by step, enhancing their problem-solving capabilities [3]. Recent advancements have introduced more sophisticated techniques that build upon these foundational methods. For instance, Self-Refine [6] allows LLMs to iteratively improve their outputs by generating and incorporating their own feedback. Reflexion [5] enables models to maintain reflective memory buffers, fostering better decision-making through verbal reinforcement learning. Additionally, ReAct [4] integrates reasoning traces with task-specific actions, allowing LLMs to interact with external tools and refine their outputs dynamically. More complex approaches like Tree of Thoughts [7] introduce frameworks for deliberate problem-solving by exploring multiple reasoning paths, thereby achieving higher accuracy in complex tasks.

Even seemingly simple techniques, like prompting the model to reread the prompt [17], can significantly improve reasoning by firing bidirectional attention mechanisms.

Recent developments from OpenAI with their o1 model demonstrate strong performance even with zero-shot prompting, suggesting the existence of advanced internal prompt design techniques that remain undisclosed. These techniques were likely used to train the model through reinforcement learning, enhancing its inherent reasoning capabilities. The fact that their methods remain closed source is a significant limitation for the broader community, motivating this research to develop open-source, high-quality prompting methods that can empower developers to achieve better outputs.

Prompt Engineering represents the iterative process of refining prompts to enhance LLM performance. This systematic approach involves testing and modifying prompts to achieve more accurate and contextually appropriate outputs. Significant progress has been made in automating this process, as demonstrated by DSPy [11], a framework that optimizes prompts based on input-output examples tailored to specific tasks. Such automation reduces the manual effort required in traditional prompt engineering while maintaining or improving output quality.

Agentic workflows, commonly referred to as agents, represent an evolution of prompt design techniques by incorporating external tools and collaborative strategies among multiple LLM-based entities. These systems leverage advanced prompting techniques to enable agents to communicate, collaborate, and utilize external resources to solve complex tasks efficiently. Notable frameworks in this domain include MetaGPT [8], which utilizes meta-programming to coordinate multiple agents, each with specialized roles, enhancing collaborative problem-solving capabilities. ChatDev [9] employs communicative agents that interact through unified language-based communication, streamlining phases like design, coding, and testing in software development. AutoGen [10] provides a framework for multi-agent conversations, allowing agents to converse and collaborate to accomplish tasks across diverse domains. GraphReader [18] enhances long-context abilities by structuring inputs into graphs, enabling agents to autonomously explore and process extensive information efficiently.

These agentic systems rely on advanced prompt design techniques to enable seamless coordination among agents, optimize decision-making, and facilitate interactions with external tools. Prompt design serves as the foundation for defining agent behaviors, task-specific instructions, and inter-agent communication. By carefully crafting these prompts, developers ensure that agents can dynamically collaborate, adapt to changing contexts, and address complex tasks effectively. This integration of prompt design with external tools highlights the continued importance of high-quality, well-structured prompts in achieving reliable and superior LLM outputs.

In summary, the integration of effective prompt design, iterative prompt engineering, and collaborative agentic workflows is fundamental to optimizing LLMs. This theoretical framework lays the groundwork for developing our project, Prompt Start, which leverages RAG methods to enhance initial prompt definition, empowering developers with open-source, high-quality prompting solutions.

2.1 Retrieval-Augmented Generation (RAG)

RAG is a hybrid framework that combines the generative capabilities of LLMs with information retrieval systems to enhance response quality, factual accuracy, and context-awareness. Unlike standalone LLMs, which rely solely on pre-trained knowledge, RAG incorporates a retrieval mechanism to fetch relevant, up-to-date information from external knowledge bases. This integration ensures that responses are grounded in factual, current data, effectively reducing hallucination issues where LLMs generate plausible yet incorrect information [19].

The RAG framework operates in two stages: retrieval and generation. In the retrieval stage, relevant documents or data are fetched using mechanisms such as dense passage retrieval (DPR) [20]. In the generation stage, this retrieved content is provided as additional context, enabling the LLM to produce responses that are both contextually relevant and factually accurate.

A recent study by Borgeaud et al. [21] explores retrieval-augmented methods that enhance domain adaptation and task-specific customization, which are particularly relevant for integrating RAG into prompt engineering frameworks. Similarly, Yun et al. [22] provide insights into optimizing retrieval pipelines, contributing to more efficient LLM interactions. These advancements underscore the importance of RAG in bridging the gap between static language models and the dynamic information needs of real-world applications.

By incorporating real-time or domain-specific information, RAG reduces hallucination issues often observed in LLMs and enhances their applicability in specialized contexts. In the domain of prompt optimization for software engineers, RAG provides access to the latest research papers, blog posts, and community-driven resources, enabling developers to explore state-of-the-art prompting techniques. It facilitates the selection of curated prompt design options while offering explanations tailored to specific use cases. Furthermore, RAG has the potential to enhance decision-making by grounding recommendations in factual, up-to-date references, which may address limitations associated with relying solely on the static pre-training of LLMs.

2.2 Related Work

In developing our framework, we draw upon several pioneering projects. One such project is PaperQA, spearheaded by Skarlinski et al. [14]. This project stands out for its innovative approach to enhancing the factual accuracy of Language Models (LMs). PaperQA has demonstrated to outperform human experts in tasks such as information retrieval, summarization, and contradiction detection, in scientific knowledge.

Another integral component of our framework is RagBuilder [15], a toolkit for optimizing RAG configurations. RagBuilder employs Bayesian optimization to efficiently identify optimal combinations of RAG parameters, including chunking strategies, chunk sizes, embedding models, and retriever types.

Kotaemon [16] serves as the foundational platform for our project, offering a hybrid RAG pipeline with sophisticated retrieval and reasoning capabilities. Its architecture combines full-text and vector search with re-ranking mechanisms to ensure optimal retrieval quality. We'll also leverage its modern web interface.

3 Research Question and Hypothesis

With this research, we propose Prompt Start, a framework designed to leverage recent advancements in GenAI to provide developers with an efficient, fast, and high-quality method for defining initial prompts tailored to their specific use cases in each prompt engineering process.

The central research question guiding this work is: *What framework can support developers in selecting appropriate prompt design techniques for initial prompts in LLM-based applications?*

Our approach will utilize SOTA components, including embedding models to transform data into vectors, vector databases for data storage, retrieval methods to search and retrieve relevant information, ranking algorithms to prioritize documents based on their relevance, language models to construct outputs, and agentic workflows to optimize the final outcome. By integrating these innovations, we aim to provide the best possible solution to the ”Blank Page Problem.”

Our key contributions include:

- Utilizing papers, projects, and blogs to identify the most suitable prompt design technique for any use case
- Developing Prompt Start, a framework for initial prompt design definition
- Publishing an open-source repository with our work
- Evaluating the framework's impact on applications and the accuracy of the chosen techniques

To achieve these goals, we develop code that integrates three open-source projects [14, 15, 16] into our framework. We gathered qualitative feedback from industry developers and conduct quantitative evaluations of the correctness of chosen prompt designs and performance improvements in specific use cases. These evaluations measure the success of our framework through both qualitative and quantitative metrics.

4 Research Methodology

Our research methodology is centered on the development and integration of three key open-source projects: PaperQA, RagBuilder, and Kotaemon. PaperQA is utilized to focus on retrieving information from scientific papers and writings, ensuring that our framework has access to reliable and factual data. RagBuilder provides the initial configurations for our RAG system, allowing us to optimize retrieval processes through efficient parameter tuning. Kotaemon serves as the boilerplate for our code development, where we integrate the outputs from PaperQA and apply configurations based on RagBuilder's results.

In parallel with code development, we are assembling a comprehensive knowledge dataset. This dataset already includes a substantial collection of scientific papers, such as those referenced in this document. We are also incorporating GitHub projects that offer valuable insights into prompting, as well as language model documentation and blogs that are recognized by the community through social media and influential newsletters.

For the evaluation of our framework, we plan to employ a combination of quantitative and qualitative metrics. Quantitative metrics include accuracy, contextual relevance, and efficiency, which will be benchmarked against existing industry-standard techniques and randomly generated prompts. Qualitative evaluation might involve gathering feedback from developers and LLM users through surveys and interviews. This feedback will likely focus on usability, clarity, and overall satisfaction with the prompt suggestions provided by our framework.

Our methodology emphasizes reproducibility, with plans to share all code and documentation in a GitHub repository. This will enable other researchers and developers to replicate and build upon our work.

4.1 Framework Architecture

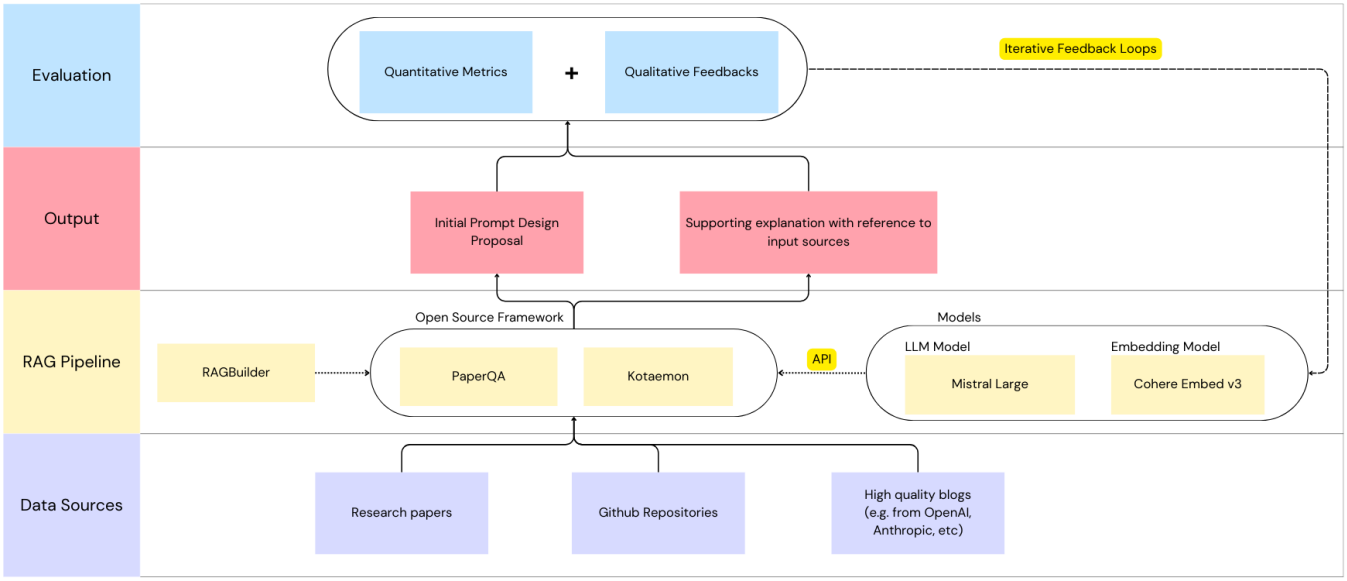

Figure 1: Framework Architecture: A flowchart illustrating the components and data flow of our Agentic RAG based prompt optimization system, from data sources through the pipeline to evaluation.

1. **Data Sources**: The framework accepts user inputs, including project files, use case descriptions, and optimization criteria. It also ingests data from selected papers, GitHub repositories, and blogs.

Papers are selected based on industry acclaim, citation counts, mentions in reputable blogs and newsletters, and endorsements from influential figures. They are chosen for their relevance to the research objectives, the novelty of techniques, and their potential impact on the field.

GitHub repositories are evaluated by the number of forks and stars, mentions in important newsletters and tweets, and overall community engagement. Selection criteria include the implementation of cutting-edge prompting techniques, code quality, and thorough documentation.

Blogs include high-quality expert articles that discuss state-of-the-art prompt design techniques and their practical applications. These sources provide additional insights into emerging trends and best practices in the field.

2. **Retrieval-Augmented Generation (RAG) Pipeline**: The RAG system processes the ingested data and retrieves relevant information based on the user's inputs. Initially, RagBuilder optimizes the RAG configuration through Bayesian optimization, determining optimal parameters such as chunk sizes, embedding models, and retrieval strategies. These configurations are then applied to our Kotaemon-based implementation. The retrieval process is distributed across specialized components: PaperQA focuses on extracting relevant information from scientific papers, while Kotaemon handles retrieval from GitHub repositories and expert blogs. Retrieved content undergoes a re-ranking process to ensure the most pertinent information is prioritized. Finally, Mistral Large, our chosen LLM, processes this curated context to generate the output.

3. **Output**: Using the retrieved information and user inputs, the agent analyzes the most suitable prompting techniques for the given use case. It considers factors such as the complexity of the task, desired output format, and optimization criteria to suggest the optimal approach. Based on the recommended prompting technique, the generator creates an initial prompt proposal. It incorporates best practices and insights from the retrieved data to craft a prompt that aligns with the user's requirements. The relevant documents to the proposal are presented to the user on a side panel.

4. **Evaluation**: The framework's evaluation is planned to include both quantitative metrics, such as accuracy and response time, and qualitative feedback from developers. This evaluation phase will allow us to assess and improve the framework's effectiveness.

4.2 Designing Process

The initial phase of our research focused on implementing and evaluating RagBuilder [15], a toolkit for optimizing RAG configurations. We successfully deployed the framework locally, which required setting up both the RagBuilder application and a Neo4j database instance for GraphRAG capabilities. While the installation process presented several environmental and dependency challenges, we were able to establish a working setup. Our implementation approach prioritized practicality within resource constraints, particularly focusing on computational efficiency and API quota management, which led us to strategically select more cost-effective models like GPT-3.5 instead of larger counterparts.

The process involved data sampling and preprocessing techniques to manage the dataset's size, followed by custom RAG configurations optimized for our resource constraints. We deliberately limited the number of configurations being tested and avoided the "Evaluate All Possible Combinations" approach to maintain efficiency. While predefined templates showed limited success, our custom configurations demonstrated better adaptability to our specific use case. We generated a synthetic test dataset for evaluation, as this approach offered a practical balance between comprehensive testing and development time constraints.

5 Results and Analysis

Our initial evaluation of RagBuilder [15] yielded valuable insights for implementing the RAG mechanism in the Kotaemon application [16]. The tool evaluated multiple RAG configurations, with the best-performing setup achieving 96% answer correctness and 98% context precision and recall. This configuration utilized a RecursiveCharacterTextSplitter with 300-token chunks and 200-token overlap, combined with a hybrid retrieval approach that merged vector similarity and BM25 retrieval methods. The implementation leveraged OpenAI's text-embedding-3-large model for embeddings and included BAAI/bge-reranker-base for contextual compression. These findings provided a strong foundation for our RAG implementation, though we noted that the evaluation was limited to a smaller dataset due to API quota constraints. This initial phase helped establish the core RAG architecture while identifying key areas for optimization in the subsequent development of the Kotaemon application.

6 Discussion

7 Conclusion and Future Work

References

[1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2023. [Online]. Available: https://arxiv.org/abs/1706.03762

[2] J. Wei, Y. Tay, R. Bommasani, C. Raffel, B. Zoph, S. Borgeaud, D. Yogatama, M. Bosma, D. Zhou, D. Metzler, E. H. Chi, T. Hashimoto, O. Vinyals, P. Liang, J. Dean, and W. Fedus, "Emergent abilities of large language models," 2022. [Online]. Available: https://arxiv.org/abs/2206.07682

[3] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. Chi, Q. Le, and D. Zhou, "Chain-of-thought prompting elicits reasoning in large language models," 2023. [Online]. Available: https://arxiv.org/abs/2201.11903

[4] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao, "React: Synergizing reasoning and acting in language models," 2023. [Online]. Available: https://arxiv.org/abs/2210.03629

[5] N. Shinn, F. Cassano, E. Berman, A. Gopinath, K. Narasimhan, and S. Yao, "Reflexion: Language agents with verbal reinforcement learning," 2023. [Online]. Available: https://arxiv.org/abs/2303.11366

[6] A. Madaan, N. Tandon, P. Gupta, S. Hallinan, L. Gao, S. Wiegreffe, U. Alon, N. Dziri, S. Prabhumoye, Y. Yang, S. Gupta, B. P. Majumder, K. Hermann, S. Welleck, A. Yazdanbakhsh, and P. Clark, "Self-refine: Iterative refinement with self-feedback," 2023. [Online]. Available: https://arxiv.org/abs/2303.17651

[7] S. Yao, D. Yu, J. Zhao, I. Shafran, T. L. Griffiths, Y. Cao, and K. Narasimhan, "Tree of thoughts: Deliberate problem solving with large language models," 2023. [Online]. Available: https://arxiv.org/abs/2305.10601

[8] S. Hong, M. Zhuge, J. Chen, X. Zheng, Y. Cheng, C. Zhang, J. Wang, Z. Wang, S. K. S. Yau, Z. Lin, L. Zhou, C. Ran, L. Xiao, C. Wu, and J. Schmidhuber, "Metagpt: Meta programming for a multi-agent collaborative framework," 2024. [Online]. Available: https://arxiv.org/abs/2308.00352

[9] C. Qian, W. Liu, H. Liu, N. Chen, Y. Dang, J. Li, C. Yang, W. Chen, Y. Su, X. Cong, J. Xu, D. Li, Z. Liu, and M. Sun, "Chatdev: Communicative agents for software development," 2024. [Online]. Available: https://arxiv.org/abs/2307.07924

[10] Q. Wu, G. Bansal, J. Zhang, Y. Wu, B. Li, E. Zhu, L. Jiang, X. Zhang, S. Zhang, J. Liu, A. H. Awadallah, R. W. White, D. Burger, and C. Wang, "Autogen: Enabling next-gen llm applications via multi-agent conversation," 2023. [Online]. Available: https://arxiv.org/abs/2308.08155

[11] O. Khattab, A. Singhvi, P. Maheshwari, Z. Zhang, K. Santhanam, S. Vardhamanan, S. Haq, A. Sharma, T. T. Joshi, H. Moazam, H. Miller, M. Zaharia, and C. Potts, "Dspy: Compiling declarative language model calls into self-improving pipelines," 2023. [Online]. Available: https://arxiv.org/abs/2310.03714

[12] S. Schulhoff, M. Ilie, N. Balepur, K. Kahadze, A. Liu, C. Si, Y. Li, A. Gupta, H. Han, S. Schulhoff, P. S. Dulepet, S. Vidyadhara, D. Ki, S. Agrawal, C. Pham, G. Kroiz, F. Li, H. Tao, A. Srivastava, H. D. Costa, S. Gupta, M. L. Rogers, I. Goncearenco, G. Sarli, I. Galynker, D. Peskoff, M. Carpuat, J. White, S. Anadkat, A. Hoyle, and P. Resnik, "The prompt report: A systematic survey of prompting techniques," 2024. [Online]. Available: https://arxiv.org/abs/2406.06608

[13] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W. tau Yih, T. Rocktäschel, S. Riedel, and D. Kiela, "Retrieval-augmented generation for knowledge-intensive nlp tasks," 2021. [Online]. Available: https://arxiv.org/abs/2005.11401

[14] M. D. Skarlinski, S. Cox, J. M. Laurent, J. D. Braza, M. Hinks, M. J. Hammerling, M. Ponnapati, S. G. Rodriques, and A. D. White, "Language agents achieve superhuman synthesis of scientific knowledge," 2024, accessed: 2024-04-27. [Online]. Available: https://arxiv.org/abs/2409.13740

[15] KruxAI, "Ragbuilder: A toolkit for creating optimal production-ready retrieval augmented generation (rag) setups," 2024. [Online]. Available: https://github.com/KruxAI/ragbuilder

[16] Cinnamon, "Kotaemon: An open-source rag-based tool for chatting with your documents," 2024. [Online]. Available: https://github.com/Cinnamon/kotaemon

[17] X. Xu, C. Tao, T. Shen, C. Xu, H. Xu, G. Long, J. guang Lou, and S. Ma, "Re-reading improves reasoning in large language models," 2024. [Online]. Available: https://arxiv.org/abs/2309.06275

[18] S. Li, Y. He, H. Guo, X. Bu, G. Bai, J. Liu, J. Liu, X. Qu, Y. Li, W. Ouyang, W. Su, and B. Zheng, "Graphreader: Building graph-based agent to enhance long-context abilities of large language models," 2024. [Online]. Available: https://arxiv.org/abs/2406.14550

[19] P. Lewis, A. Piktus, F. Petroni, V. Karpukhin, B. Oguz, S. Min, W. tau Yih, T. Rocktäschel, S. Riedel, and D. Kiela, "Retrieval-augmented generation for knowledge-intensive nlp tasks," 2020. [Online]. Available: https://arxiv.org/abs/2010.08191

[20] V. Karpukhin, B. Oguz, S. Min, P. Lewis, L. Wu, S. Edunov, K. Clark, and W. tau Yih, "Dense passage retrieval for open-domain question answering," 2020. [Online]. Available: https://arxiv.org/abs/2004.04906

[21] S. Borgeaud, A. Mensch, J. Hoffmann, T. Cai, E. Rutherford, K. Millican, G. van den Driessche, J.-B. Lespiau, B. Damoc, A. Clark, and Others, "Improving language models by retrieving from trillions of tokens," 2022. [Online]. Available: https://arxiv.org/abs/2112.04426

[22] D. Yun, R. Kumar, A. Singh, and H. Smith, "Optimizing retrieval-augmented models for efficiency and domain adaptation," 2023. [Online]. Available: https://arxiv.org/abs/2304.12345