Qeios

Research Article

LLM as HPC Expert: Extending RAG Architecture for HPC Data

Yusuke Miyashita[1], Patrick Kin Man Tung[1], Johan Barthelemy[2]

1. Research Technology Services, University of New South Wales, Australia; 2. Nvidia (United States), Santa Clara, United States

High-Performance Computing (HPC) is crucial for performing advanced computational tasks, yet their complexity often challenges users, particularly those unfamiliar with HPC-specific commands and workflows. This paper introduces Hypothetical Command Embeddings (HyCE), a novel method that extends Retrieval-Augmented Generation (RAG) by integrating real-time, user-specific HPC data, enhancing accessibility to these systems. HyCE enriches large language models (LLM) with real-time, user-specific HPC information, addressing the limitations of fine-tuned models on such data. We evaluate HyCE using an automated RAG evaluation framework, where the LLM itself creates synthetic questions from the HPC data and serves as a judge, assessing the efficacy of the extended RAG with the evaluation metrics relevant for HPC tasks. Additionally, we tackle essential security concerns, including data privacy and command execution risks, associated with deploying LLMs in HPC environments. This solution provides a scalable and adaptable approach for HPC clusters to leverage LLMs as HPC expert, bridging the gap between users and the complex systems of HPC.

1. Introduction

HPC is powerful but often inaccessible to a broad range of users due to their inherent complexity. While graphical user interfaces (GUIs) such as Open OnDemand[1] have made HPC resources more accessible, users still need to interact with the system through command-line interfaces (CLIs) for resource availability, monitoring queue and software availabilities. This requirement places a steep learning curve on users unfamiliar with shell commands and the intricacies of the HPC environment. Furthermore, relying on HPC experts to answer user queries is not an efficient use of time, as their expertise is a valuable and limited resource.

Recent advances in large language models (LLMs)[2][3] have opened new possibilities for simplifying user interactions with HPC systems. Tools such as ShellGPT[4] and AI-Shell[5] have demonstrated the potential for LLMs to translate human-readable queries into executable shell commands, simplifying the interaction with complex systems. However, these systems lack the capability to integrate cluster-specific documentation or addressing the real-time user information, which is often essential for users to efficiently navigate HPC environments. LLMs directly putting commands in the terminal also poses some security risks.

Retrieval-Augmented Generation (RAG)[6] offers a solution by extending LLMs with domain-specific knowledge, dynamically incorporating specific HPC cluster information corresponding to specific HPC organization into the model's context.

Fine-tuned LLMs[7][8][9] have shown potential in aiding parallel programming and detecting race conditions, demonstrating their strength in coding parallel programs. However, this paper focuses on extending RAG to flexibly integrate additional cluster documentation and real-time user information.

We introduce HyCE, an extension of RAG that enriches LLMs with HPC-specific data, allowing them to serve as HPC experts HPC users. To evaluate this approach, we propose a novel framework where the LLM generates synthetic HPC-specific datasets, evaluates itself using defined metrics for HPC workflows, and acts as its own judge to assess performance. Additionally, we address critical security concerns, including safeguarding data privacy and ensuring command execution integrity, to facilitate safe deployment in HPC environments.

This approach not only simplifies access to HPC resources but also lays the foundation for a more interactive and intelligent interface that adapts to the specific needs of each HPC organization. To facilitate this, we open-source the code[1] of our extended RAG, enabling straightforward and secure deployment of LLMs with HPC context in any organization.

2. Related Work

2.1. LLMs in HPC Environments

As HPC applications grow in scale and complexity, there are examples of LLMs being adapted to address HPC-specific requirements in programming, code generation[9][3][8][10][11][12], and translation from natural language to commands[4][5].

HPC-GPT[7], a fine-tuned version of LLaMA[3] on HPC-specific datasets, optimizes tasks like data race detection in OpenMP parallel code, thereby enhancing productivity and accuracy in specialized coding. Similarly, HPC-Coder[8] pushes the boundaries of LLM utility in HPC by generating parallel, performance-optimized code. It assists in annotating code with OpenMP pragmas and forecasting the performance impacts of code modifications, providing expert-level support for complex, high-performance programming.

To enhance user accessibility to HPC systems, tools like Shell-GPT[4] and AI-shell[5] have been developed to translate natural language queries into executable bash commands. While these tools demonstrate significant potential, academic research on leveraging LLMs to improve user interaction with HPC resources remains limited. This paper addresses this gap by introducing HyCE, a method that integrates HPC commands into a standard RAG architecture. Given the critical importance of secure command execution in HPC environments, this paper also explores the security implications of the proposed approach in a dedicated section.

2.2. Retrieval–Augmented Generation (RAG)

RAG[6] offers another method for incorporating supplementary information into pre-trained LLMs, presenting a flexible alternative to fine-tuning. Unlike fine-tuning, which modifies the model itself, which often is more expensive, RAG dynamically accesses up-to-date or domain-specific information. This dynamic nature makes RAG particularly suited for HPC environments where real-time data, such as user and system information, is essential. For instance, queries like "What GPUs are available to me?" or "What is the status of my program?" necessitate executing HPC commands and interpreting their output within the user's specific environment. By leveraging RAG, LLMs can respond more accurately, providing context-aware support in HPC contexts.

Among various RAG techniques, including query translation, indexing, and retrieval[13][14][15], our HyCE approach took inspiration from Hypothetical Document Embedding (HyDE)[16]. HyDE generates a hypothetical answer based on a query, using it to retrieve relevant contexts more accurately. In a similar way, HyCE utilizes hypothetical commands that are embedded into a vector space, which helps in retrieving similar real commands more effectively to enhance the semantic context retrieval.

Moreover, an automated and continuous evaluation workflow is essential for RAG applications in HPC settings. Recent research has introduced LLMs as evaluators[17][18][19], enabling a self-sustaining

process in which the LLM generates synthetic data for evaluation and serves as a "judge" to assess RAG performance. Custom evaluation metrics tailored to specific RAG tasks allow LLM evaluators to effectively assess various RAG implementations, including those involving HPC data.

3. RAG with HPC Data

3.1. Approach

In our approach expressed in Figure 1, we extend the standard RAG framework to perform HPC-specific question-answering by integrating data such as cluster documentation and command outputs. In a typical RAG workflow, chunks[20][2] are created from these data, embedded with a text embedding model, and used as context. Top K relevant chunks are retrieved with a bi-encoder[21] and ranked with a cross-encoder[22], then prompt is constructed for LLM to answer. Additionally, we incorporate HyCE into this standard RAG pipeline to accurately include command outputs in the LLM's context.
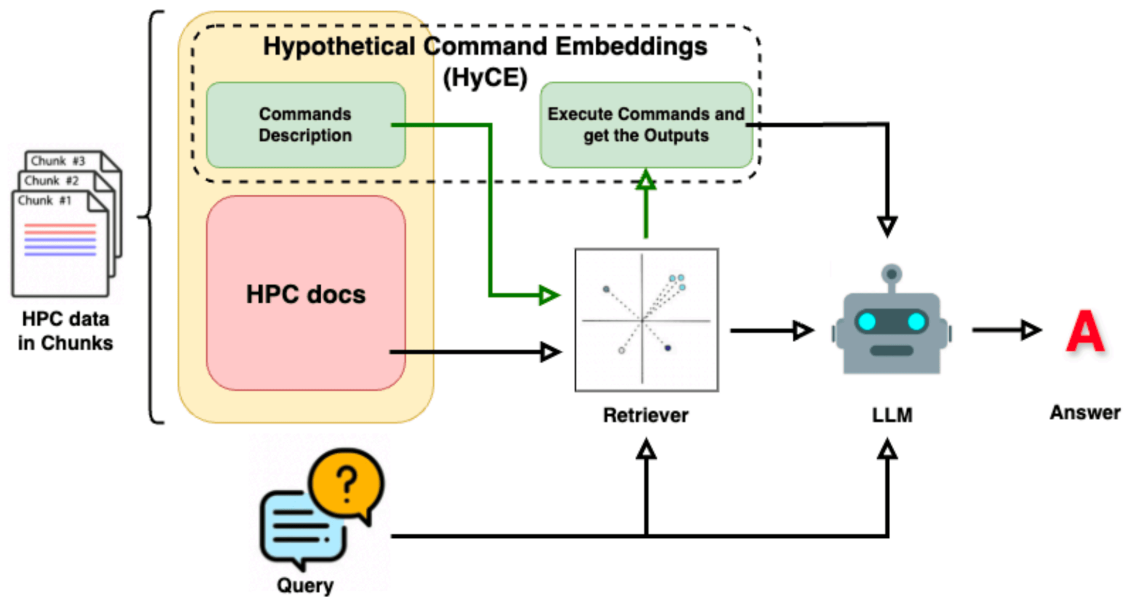

Figure 1. RAG Architecture with HPC Data. In addition to the conventional RAG architecture, it incorporates Hypothetical Command Embeddings (HyCE), which enhances command retrieval to accurately access user-specific HPC data. When a command description is retrieved (indicated by green arrows) based on the query, HyCE executes the command, and the resulting command output is fed into the LLM as context to answer user query

3.2. HPC data

In this paper, HPC data encompasses to cluster documentations and shell commands.

Cluster documentation serves as a comprehensive guide to navigating an HPC environment. Typically available online, it provides essential details such as the cluster's purpose, account setup procedures, and access methods. It also includes overviews of hardware, software, and storage resources, along with specific instructions for job submission using schedulers like SLURM or PBS. Furthermore, it outlines best practices for data management and performance optimization while addressing security policies, data protection guidelines, and compliance requirements. Additional resources, such as training materials, support contacts, sample job scripts, tutorials, and FAQs, help users effectively utilize HPC clusters for various tasks.

Shell commands, on the other hand, allow users to retrieve dynamic, real-time information specific to their needs. These commands provide insights into resource availability (e.g., GPUs or CPUs available), job queue status (e.g., running, queued, or held jobs), and software availabilities. Integrating these commands into the RAG framework enables the generation of tailored, context-aware responses, ensuring users receive precise and actionable information for their specific HPC workflows.

3.3. Hypothetical Command Embeddings (HyCE)

For handling terminal commands, we employ Hypothetical Command Embeddings (HyCE), an adaptation of the "HyDE" (Hypothetical Document Embeddings) approach. In HyDE, a hypothetical answer to the query is generated to match relevant contexts for retrieval. Similarly, HyCE leverages command descriptions to identify the most appropriate command to execute in response to a query. In this context, a "hypothetical command" refers to a descriptive explanation of what the command does. For example, for the command "nvidia-smi," the description might be "This command checks the GPU model, memory usage, and utilization rate in real time". After retrieving the relevant command, it is executed to generate output, which is then used as context for the LLM to formulate an answer to the query.

To measure the effectiveness of the HyCE method in retrieving HPC data, we compared the average text similarity between two setups, a cross-encoder which matches the query directly with command and a cross-encoder with HyCE which matches the query with a command description instead. The

results indicate the similarity score is consistently higher between the user query and the HyCE-augmented description than between the query and the raw command.

Matching the query to a command description proves more accurate than direct matching for several reasons. First, commands are often abbreviated, like "ls" for "list files," which lacks semantic meaning on its own, making direct semantic matching ineffective. Second, commands are syntactically structured and thus incompatible with the semantic search methods typically used in bi-encoder and cross-encoder models. By using descriptive explanations instead, HyCE makes the command more semantically aligned with the user's natural language query.

4. Automatic RAG evaluation with HPC data

Using the generated chunks and the extended RAG architecture, we expand the evaluation process to assess RAG's performance with HPC data automatically as described in Table 1

Cross-Encoders	Avg Top Sim Score Query vs Command Name	Avg Top Sim Score Query vs Command Description
nvidia/llama-3.2-nv-rerankqa-1b-v1	-2.8283	-2.7880
cross-encoder/ms-marco-MiniLM-L-12-v2	-9.0231	-5.1792
cross-encoder/stsb-roberta-large	0.0906	0.5285
dangvantuan/CrossEncoder-camembert-large	0.2501	0.5718
yunyu/cross-encoder-stsb-deberta-v3-large	0.1472	0.3981

Table 1. Comparison of the average similarity scores between the top-matching query and either the command or command description (HyCE setup). Higher similarity scores indicate greater textual similarity.

Algorithm 1 Automatic RAG Evaluation

Input: HPC data chunks $\mathcal{D}$
Output: RAG Evaluation Score

$\quad (Q_{hyp}, A_{hyp}) \leftarrow$ `LLM_Synthetic_Data_Gen`$(\mathcal{D})$ $\quad \triangleright$ Generate synthetic Q&A pairs

$\quad (Q'_{hyp}, A'_{hyp}) \leftarrow$ `LLM_Filter`$(Q_{hyp}, A_{hyp}, \mathcal{D})$ $\quad \triangleright$ Filter pairs

$\quad A_{pred} \leftarrow$ `RAG`$(Q'_{hyp}, \mathcal{D})$ $\quad \triangleright$ Generate predicted answers

$\quad$ `Eval_Score` $\leftarrow$ `LLM_Eval`$(A'_{hyp}, A_{pred}, \mathcal{D})$ $\triangleright$ Evaluate answers

$\quad$ **Return** `Eval_Score`

4.1. Question & Answer Generation and Filtering

To comprehensively evaluate the RAG system, we utilize chunks of HPC-specific information to generate hypothetical question-answer (Q&A) pairs. These pairs simulate potential user queries and their corresponding answers based on the provided context, creating a benchmark for testing the system's performance.

The HPC data chunks comprise two key sources: online documentation and shell commands. For Q&A generation based on command chunks, the commands are executed, and both the command description and output are used in LLM to create the Q&A pairs. This ensures the generated pairs accurately reflect the practical use cases and specific details of the HPC environment.

In this study, a total of 100 Q&A pairs were generated: 90 derived from cluster documentation and 10 from shell commands. To ensure the quality and relevance of these pairs, the LLM evaluates them using three criteria aligned with typical HPC usage:

- **Groundedness:** Ensuring the question can be answered unambiguously using the available context.
- **Relevance:** Verifying the question addresses practical and commonly encountered issues in HPC environments.
- **Standalone:** Confirming the question is understandable without requiring additional background information.

Each response is evaluated using a binary scoring system (1 for success, 0 for failure), and only Q&A pairs scoring 1 on all criteria are included in the evaluation set. This rigorous filtering process ensures the Q&A pairs are both realistic and applicable, providing a robust foundation for assessing the RAG system. Detailed prompts and output formats for this process are provided in the Appendix.

4.2. Evaluation of RAG via LLM as a Judge

Following the generation of hypothetical Q&A pairs, the RAG system generates responses for each question. These responses are evaluated by comparing them to reference answers using defined scoring criteria:

- **Correctness:** Ensuring that the response accurately matches the reference answer.
- **Faithfulness:** Verifying that the response is free from errors, does not hallucinate, and aligns closely with the given context.

This evaluation framework assesses both the factual accuracy and the overall reliability of the RAG system's responses, emphasizing their applicability to HPC users. Unlike the direct scoring approach used for filtering synthetic Q&A pairs, this stage employs a reference-based evaluation approach[19]. Each generated answer is scored against the reference answer using binary metrics, assigning a score of 1 for success and 0 for failure for each criterion.

This process provides a structured and rigorous means of evaluating the RAG system, ensuring its responses meet the practical needs of HPC environments. The full prompt can be found in the Appendix.

5. Experiment and Discussion

While the extended RAG system is designed for deployment as a chatbot integrated into HPC GUIs like Open OnDemand to enable seamless and intuitive user interactions, this experiment evaluates its functionality in a command-line environment. Specifically, the RAG evaluation was conducted on the terminal of Katana[23], an on-premises HPC cluster managed by Research Technology Services at UNSW Sydney. For LLM inference in the RAG, Nvidia NIM microservices and OpenAI were utilized. Details of the RAG hyperparameters and models used in the experiment are provided in the Appendix.

5.1. Qualitative Analysis of RAG with HPC Data

We conducted a qualitative analysis to evaluate the impact of incorporating shell command data via HyCE. For instance, when asked, "What GPUs do I have access to?", an LLM relying solely on cluster documentation provided only generic information about GPU types. In contrast, our RAG setup with HyCE dynamically retrieved user-specific context by executing relevant commands, accurately identifying the user's available GPUs, such as Nvidia V100 and A100.

This demonstrates how the integration of shell commands enables RAG to generate precise, tailored responses, enhancing the relevance and utility of its outputs for HPC users. Additional examples are shown in Figure 2.

Question 1: What GPUs are available for me?

RAG with just HPC docs: The documents do not provide specific information about the types or models of GPUs available. They mention the high demand for GPU nodes and the process for accessing them, such as waiting in a common queue or purchasing a GPU node for priority access. If you need precise details about specific GPU models or availability, you may need to consult additional resources or contact the provider directly.

RAG with HPC docs and HyCE: You have V100 and A100 GPUs available.

Question 2: What is the status of my job?

RAG with just HPC docs: To determine the status of your job, you should run the `qstat` command. Look for the second last column in the output:

- If you see an "R", it means your job is currently running.
- If you see an "H", it means your job is on hold and cannot currently run.

You can also use the command with your ZID or the environment variable `$USER` to list your jobs specifically. Adding the `-s` flag will give you more detailed status information.

RAG with HPC docs and HyCE: Based on the provided documents, it seems your job is currently queued, waiting for a suitable compute node to become available. You can check its status using the `qstat` command and look for a "Q" in the second last column to confirm it is in the queue.

Figure 2. Example of comparing of two RAG responses. Unlike the ambiguous response provided by RAG with just HPC documents, RAG combined with HyCE can address user specific HPC questions more precisely.

5.2. Orthogonality of HyCE to Other RAG Improvement Methods

Our approach to extending RAG with HPC-specific data complements other RAG improvement methods such as prompt engineering using Chain of Thought(CoT)[24] and better models. The results of automatic evaluation in Table 2 demonstrate that the incorporation of these components into the pipeline proportionally enhances the performance of our HPC-augmented RAG system. For instance, while HyCE alone improved the baseline RAG by 4.66%, the combination of CoT and better models effectively leveraged the shell commands' output provided by HyCE, resulting in incremental performance gains.

Changes	Eval Score	∆Eval Score
RAG baseline	77.67%	-
+ HyCE	82.33%	4.66%
+ Prompt Engineering(CoT)	83%	0.67%
+ Better Retrieval, Re-Rank models and LLM	86%	3%

Table 2. Automatic RAG evaluation with incremental changes

In addition, our framework is designed for adaptability, enabling users to integrate fine-tuned models optimized for generating parallel code in HPC environments or to adopt open-source models for enhanced security. This flexibility underscores the robustness of our approach across diverse HPC scenarios, allowing users to tailor the system to their unique needs and security requirements.

Furthermore, the RAG architecture can be seamlessly integrated with analytics tools to enhance user support. For example, analyzing user interactions can help identify frequently asked questions, driving iterative improvements to user documentation. These enhancements, in turn, enrich the RAG's generative capabilities, creating a "spiral of improvement" where both user document and RAG evolve to better serve users.

5.3. Limitations of the Automatic RAG Evaluation

Despite its utility, our automatic RAG evaluation method has some limitations. This approach relies on data chunks to create hypothetical questions and answers, which means that it cannot effectively measure performance outside of the provided documentation chunks or commands. Consequently, the system may exhibit hallucinations when responding to queries that fall outside the scope of the user documentation or command output in the real world. Future work could explore evaluation methods that assess RAG's ability to generalize beyond specific chunks to further enhance robustness of the RAG system.

While context retrieval quality could also be evaluated through metrics such as accuracy, precision, F1 score and AUC, this aspect remains consistent with conventional RAG implementations and thus is not a primary focus in our project. While these metrics may provide additional insights, they fall outside the scope of the HyCE-specific extensions in our RAG approach.

6. Security Considerations

6.1. Data Privacy

Protecting sensitive HPC information is paramount, especially when interacting with external servers (e.g., OpenAI). Solutions include using enterprise and local models. For enterprise deployments, privacy can be maintained if the provider does not record input-output data and does not train their LLMs with user data, thereby ensuring secure usage. Additionally, future encryption technologies may allow data to be encrypted before being sent externally and processed by the LLM. Alternatively, hosting local models entirely within the HPC environment enhances security by eliminating the need to transmit data externally. However, this approach has challenges, as local model hosting requires substantial compute resources and can reduce available HPC resources for other tasks. Thus, finding a balance between privacy and efficient HPC utilization remains essential.

6.2. Command Execution Security

To maintain security in HPC environments, we implement several layers of safeguards to prevent direct command execution by the LLM:

- **LLM Safety against Prompt Injection:** LLMs are usually pre-trained to counter prompt injection attacks, ensuring that queries with harmful prompts and outputs are detected, keeping the LLM's responses within safe operational bounds. From the perspective of building RAG, developers can ensure this safety aspect when choosing the LLM model.
- **Predefined Command Retrieval:** In HyCE, RAG is restricted to retrieving only predefined, validated commands, preventing it from executing arbitrary commands autonomously. This reduces the risk of unintended or unsafe system actions.
- **Restricted User Privileges:** The LLM operates under user-level privileges, limiting access to sensitive system functions. This minimizes the potential impact of any unintended commands,

restricting the LLM to non-administrative interactions.

- **Containerization:** Deploying the RAG application in a container isolates LLM processes from the core HPC environment, ensuring that any unexpected behavior remains contained and does not impact overall system stability.

These layers of security collectively create a secure framework, allowing safe LLM-assisted interactions within HPC environments.

7. Conclusion

In this paper, we introduced HyCE, a novel approach to extending RAG with HPC-specific data, transforming LLMs into effective HPC expert assistants. By incorporating cluster-specific documentation and command outputs, our method enables LLMs to provide contextually accurate, user-tailored responses that address the unique needs of HPC users. Our evaluation framework further strengthens this system, utilizing the LLM as a judge to automate performance assessments and support continuous improvement. This setup, while robust and adaptable, includes layered security safeguards to prevent unauthorized command execution and maintain data privacy, making it suitable for deployment in an HPC environments.

Appendix A. RAG Hyperparameters and Models used

Hyperparameters	
Parameter	**Value**
Chunk Size	Determined by LLM
Top K Retrieval	20
Top K Re-rank	5
Max Context Size	128k tokens (limit of LLM)
LLM Temperature	0
Max Output Tokens	4096
Models	
Category	**Model Name(increasing order of performance in the RAG)**
Retrieval Model	nvidia/llama-3.2-nv-embedqa-1b-v1
	sentence-transformers/multi-qa-MiniLM-L6-cos-v1
Re-rank Model	nvidia/llama-3.2-nv-rerankqa-1b-v1
	cross-encoder/ms-marco-MiniLM-L-12-v2
LLM	meta/llama-3.1-405b-instruct
	gpt-4o-2024-08-06

Appendix B. LLM Prompt Templates

We list the prompt templates for LLM Q&A generation, Q&A filtering, LLM answer, and LLM judge.

Please refer to our GitHub repository[2] for full details.

```
system_message = "You are a helpful assistant that generates factoid questions and answers."

user_message = f"""
Your task is to write a factoid question and an answer given a context.
Your factoid question should be answerable with a specific, concise piece of factual information from the context.
Imagine you are assisting a new user to High Performance Computing (HPC), and generate a question that a new user to HPC might ask.
Your factoid question should be formulated in the same style as questions new users to HPC would ask.
This means that your factoid question MUST NOT mention something like "according to the passage" or "context".

Provide your answer in the following JSON format:
```json
{{
  "question": "Your generated question here",
  "answer": "Your answer to the question here"
}}
```

Now here is the context.
Context: {context}
"""
```

Figure 3. The prompts for RAG Q&A generation

```
system_message = "You are a helpful assistant that evaluates questions."

user_message = f"""
You will be given a context and a question.

Your task is to evaluate the question based on the following criteria:
1. **Groundedness**: The question can be answered unambiguously with the given context.
2. **Relevance**: The question addresses a common, practical problem or concept that HPC users are likely to encounter given the context.
3. **Standalone**: The question is understandable without additional context.
For each criterion, provide a 'total rating' of **1** (meets the criterion) or **0** (does not meet the criterion).

Provide your answer in the following JSON format:
```json
{{
  "evaluation": "Your rationale for the ratings",
  "groundedness_score": 0 or 1,
  "relevance_score": 0 or 1,
  "standalone_score": 0 or 1
}}
```

You MUST provide values for all keys in your answer.
Now here are the question and context.
Question: {question}
Context: {context}
"""
```

Figure 4. The prompts for RAG Q&A filtering

```
system_message = "You are an assistant that provides accurate and concise answers based on the provided documents."

user_message_cot = f"""
Answer the following question based on the provided documents.
Question: {question}
Documents: {context}

Provide your thinking for each step, and then provide the answer in the following **JSON format**:
```json
{{
  "reasons": "Your reasons to the question here",
  "answer": "Your answer to the question here"
}}
```
 """

user_message_no_cot = f"Answer the following question based on the provided documents.
Question: {question}
Documents: {context}"
```

Figure 5. The prompts for RAG answering

```
system_message = "You are a helpful assistant that evaluates answers."

user_message = f"""
You will be given a question, a generated answer, and a reference answer.

Your task is to evaluate the generated answer based on the following criteria:
1. **Correctness**: The answer is correct and accurate based on the reference answer.
2. **Faithfulness**: The answer does not hallucinate or contradict any factual information presented in the context.
For each criterion, provide a 'total rating' of **1** (meets the criterion) or **0** (does not meet the criterion).

Output your reasoning for the ratings in the evaluation field and provide your scores in the following JSON format:
```json
{{
  "evaluation": "Your detailed feedback explaining the evaluation based on the criteria.",
  "scores": {{
    "Correctness": 0 or 1,
    "Faithfulness": 0 or 1,
  }}
}}
```

Now, here are the inputs:
Question: {question}
Generated Answer: {generated_answer}
Reference Answer: {true_answer}
"""
```

Figure 6. The prompts for RAG evaluation

Acknowledgements

Footnotes

1 https://github.com/Yusuke710/llm_rag_eval_hpc

2 https://github.com/Yusuke710/llm_rag_eval_hpc

References

1. <u>∧</u>*Hudak DE, Johnson D, Nicklas J, Franz E, McMichael B, Gohar B (2016). "Open OnDemand: Transformi ng Computational Science Through Omnidisciplinary Software Cyberinfrastructure". In Proceedings of t he XSEDE16 Conference on Diversity, Big Data, and Science at Scale (Miami, USA) (XSEDE16). New Yor k, NY, USA: Association for Computing Machinery. Article 43, 7 pages. doi:<u>10.1145/2949550.2949644</u>.*

2. <u>a</u>, <u>b</u>*Khattab O, Zaharia M (2020). "ColBERT: Efficient and Effective Passage Search via Contextualized L ate Interaction over BERT". arXiv. <u>arXiv:2004.12832 [cs.IR]</u>.*

3. <u>a</u>, <u>b</u>, <u>c</u>**Dubey A, et al. (2024). "The Llama 3 Herd of Models". arXiv. <u>arXiv:2407.21783 [cs.AI]</u>.**

4. <u>a</u>, <u>b</u>, <u>c</u>**TheR1D (2024). Shell GPT. Available from: <u>https://github.com/TheR1D/shell_gpt</u>. Accessed: 2024- 11-24.**

5. <u>a</u>, <u>b</u>, <u>c</u>**Builder.io (2023). AI Shell. Available from: <u>https://github.com/BuilderIO/ai-shell/tree/main</u>. Acces sed: 2024-11-24.**

6. <u>a</u>, <u>b</u>*Lewis P, Perez E, Piktus A, Petroni F, Karpukhin V, Goyal N, Küttler H, Lewis M, Yih W, Rocktäschel T, Riedel S, Kiela D (2021). "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks". arXi v. <u>arXiv:2005.11401 [cs.CL]</u>.*

7. <u>a</u>, <u>b</u>*Ding X, Chen L, Emani M, Liao C, Lin P-H, Vanderbruggen T, Xie Z, Cerpa A, Du W (2023). "HPC-GP T: Integrating Large Language Model for High-Performance Computing". In: Proceedings of the SC '23 Workshops of The International Conference on High Performance Computing, Network, Storage, and An alysis (SC-W 2023). ACM. p. 951–960. doi:<u>10.1145/3624062.3624172</u>.*

8. <u>a</u>, <u>b</u>, <u>c</u>*Nichols D, Marathe A, Menon H, Gamblin T, Bhatele A (2024). "HPC-Coder: Modeling Parallel Pro grams using Large Language Models". In ISC High Performance 2024 Research Paper Proceedings (39t h International Conference). IEEE; 2024. p. 1–12. doi:<u>10.23919/isc.2024.10528929</u>.*

9. <u>a</u>, <u>b</u>*Chen L, Ahmed NK, Dutta A, Bhattacharjee A, Yu S, Mahmud QI, Abebe W, Phan H, Sarkar A, Butler B, Hasabnis N, Oren G, Vo VA, Munoz JP, Willke TL, Mattson T, Jannesari A (2024). "The Landscape and Challenges of HPC Research and LLMs". arXiv. <u>arXiv:2402.02018 [cs.LG]</u>.*

10. <u>∧</u>*Kadosh T, Hasabnis N, Vo VA, Schneider N, Krien N, Wasay A, Ahmed N, Willke T, Tamir G, Pinter Y, Ma ttson T, Oren G (2023). "Scope is all you need: Transforming LLMs for HPC Code". arXiv. <u>arXiv:2308.094 40 [cs.CL]</u>.*

11. <u>∧</u>*Chen L, Lin PH, Vanderbruggen T, Liao C, Emani M, de Supinski B (2023). "LM4HPC: Towards Effectiv e Language Model Application in High-Performance Computing". In: OpenMP: Advanced Task-Based,*

Device and Compiler Programming. Springer Nature Switzerland. pp. 18–33. doi:10.1007/978-3-031-40744-4_2.

12. ^*Kadosh T, Hasabnis N, Vo VA, Schneider N, Krien N, Capota M, Wasay A, Ahmed N, Willke T, Tamir G, Pinter Y, Mattson T, Oren G (2024). "MonoCoder: Domain-Specific Code Language Model for HPC Codes and Tasks". arXiv. arXiv:2312.13322 [cs.PL].*

13. ^*Asai A, Wu Z, Wang Y, Sil A, Hajishirzi H (2023). "Self-RAG: Learning to Retrieve, Generate, and Critique through Self-Reflection". arXiv [cs.CL]. arXiv:2310.11511.*

14. ^*Rackauckas Z (2024). "RAG-Fusion: A New Take on Retrieval Augmented Generation". International Journal on Natural Language Computing. 13 (1): 37–47. doi:10.5121/ijnlc.2024.13103.*

15. ^*Trivedi H, Balasubramanian N, Khot T, Sabharwal A (2023). "Interleaving Retrieval with Chain-of-Thought Reasoning for Knowledge-Intensive Multi-Step Questions". arXiv. arXiv:2212.10509 [cs.CL].*

16. ^*Gao L, Ma X, Lin J, Callan J (2022). "Precise Zero-Shot Dense Retrieval without Relevance Labels". arXiv. arXiv:2212.10496 [cs.IR].*

17. ^*Zheng L, Chiang WL, Sheng Y, Zhuang S, Wu Z, Zhuang Y, Lin Z, Li Z, Li D, Xing EP, Zhang H, Gonzalez JE, Stoica I (2023). "Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena". arXiv. arXiv:2306.05685 [cs.CL].*

18. ^*Shahul Es, Jithin James, Luis Espinosa-Anke, Steven Schockaert (2023). "RAGAS: Automated Evaluation of Retrieval Augmented Generation". arXiv. arXiv:2309.15217 [cs.CL].*

19. a, b*Adlakha V, BehnamGhader P, Lu XH, Meade N, Reddy S (2024). "Evaluating Correctness and Faithfulness of Instruction-Following Models for Question Answering". arXiv. arXiv:2307.16877 [cs.CL].*

20. ^*Günther M, Mohr I, Williams DJ, Wang B, Xiao H (2024). "Late Chunking: Contextual Chunk Embeddings Using Long-Context Embedding Models". arXiv. arXiv:2409.04701 [cs.CL].*

21. ^*Devlin J, Chang MW, Lee K, Toutanova K (2019). "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". arXiv. arXiv:1810.04805 [cs.CL].*

22. ^*Reimers N, Gurevych I (2019). "Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks". arXiv [cs.CL]. Available from: https://arxiv.org/abs/1908.10084.*

23. ^**University of New South Wales** *(2014). Katana. doi:10.26190/669x-a286. Accessed: 2024-11-24.*

24. ^*Wei J, Wang X, Schuurmans D, Bosma M, Ichter B, Xia F, Chi E, Le Q, Zhou D (2023). "Chain-of-Thought Prompting Elicits Reasoning in Large Language Models". arXiv. arXiv:2201.11903 [cs.CL].*

Declarations

Potential competing interests: No potential competing interests to declare.