From Chatbots to Interfaces: Diversifying the Application of Large Language Models for Enhanced Usability

Jonathan Sparks Hewlett Packard Enterprise Bloomington, USA jonathan.sparks@hpe.com Pierre Carrier Hewlett Packard Enterprise Bloomington, USA pierre.carrier@hpe.com Gallig Renaud Hewlett Packard Enterprise Grenoble, France gallig.renaud@hpe.com

Abstract—This paper explores the application of Large Language Models (LLMs) in three distinct scenarios, demonstrating their potential use to aid user experience and efficiency. Firstly, we examine the application of LLM models such as OpenAI's GPT -3 or Llama in chatbots to assist in programming environments, providing real-time assistance to developers. Secondly, we explore using LLMs and Python to search internal document corpus for performance engineering, significantly improving the retrieval of relevant information from extensive technical documentation. Lastly, we investigate using LLMs as an interface to system batch schedulers, such as Slurm or PBS, replacing domain-specific languages and prompts with natural text. This approach democratizes access to complex systems, fostering ease of use and enhancing the user experience. Through these use cases, we underscore the versatility and potential of LLMs, highlighting their role as an aid to system operation and user experience.

Keywords—LLM, RAG, assistant, chatbot, performance, development, scheduler, HPC, services

I. INTRODUCTION

New users may find using a supercomputer daunting; the system's complexity and diversity can overwhelm even the most knowledgeable person. To navigate the system, they often turn to support personnel, emails, instant message channels, or even "call a friend." This high entry bar can lead to frustration, prompting them to seek alternate systems with a lower barrier of entry or to create documentation on system usage. The challenge for users becomes even more problematic as these systems have different management and configurations, causing portability and time to solve difficulties. Imagine if we had a set of tools to guide the user through the system and provide a top-tier knowledgeable agent capable of answering and interacting with the system. This agent could assist the user on a wide range of topics, focusing on the architecture and environment the user encounters. We might now have a solution: generative AI and Large Language Models (LLMs).

The advent of Large Language Models (LLMs) has marked a paradigm shift in artificial intelligence, particularly in natural language processing (NLP). These sophisticated models, such as OpenAI's ChatGPT [1] and Llama [2], have been trained on extensive text corpora, enabling them to generate coherent and contextually relevant language outputs. The ability of LLMs to understand and produce human-like text has opened new avenues for human-computer interaction, making them a pivotal asset in the technological landscape.

We cannot overstate the importance of LLMs, as they play a crucial role in automating and enhancing tasks that require language understanding. LLMs are revolutionizing our approach to problem-solving and decision-making processes, from powering customer service chatbots to driving advanced analytics in data science [3]. Their versatility enables applications in various domains, delivering efficient but also intuitive and user-friendly solutions.

In the paper, we present the practical applications of LLMs using three exemplary use cases, each highlighting the use of LLMs to improve user experience and operational efficiency. We begin by examining the use of LLMs within specialized programming environments, where they provide developers with real-time assistance and information about the HPE programming environment and general HPC coding practices and system use, thereby streamlining application building and execution. The second scenario investigates the application of LLMs and Python interfaces [4] in searching internal document corpora for application performance engineering, where their ability to surface relevant information from a vast dataset quickly is invaluable. Lastly, we explore using LLMs as interfaces for complex systems, where they interpret natural text inputs, eliminating the need for users to learn domain-specific languages and commands. Through these use cases, we aim to demonstrate the transformative impact of LLMs on system operation and user experience, emphasizing their role in democratizing access to technology.

The following sections will offer an in-depth analysis of the challenges and methods to address the use case. Section II will establish the definitions and background knowledge essential for thoroughly comprehending this paper. Section III will furnish a comprehensive overview of the related works. Subsequently, in sections IV and V, we will explain the use cases and criteria by which we selected models and methods used in the proposed solutions and dive into our development specifics. Later, we will

explain our findings in Section VI and open problems in Section VII, respectively. Lastly, we will provide our concluding remarks in Section VIII.

II. BACKGROUND

Throughout the paper, we will interchangeably use the terms large language models (LLMs), Models, and service platforms, such as GPT-4 and OpenAI. Similarly, we will assume software developers and users are the same entity. The following defines short forms and standard definitions used throughout the paper.

A. Short Forms

- LLM Large Language Model
- SLM Small Language Model
- RAG Retrieval-Augmented Generation
- Agent An LLM-powered intelligent program
- Chatbot Interactive LLM program
- Retriever Unstructured data query interface
- Corpus A collection of public/private sources used to enhance the LLM.
- Vector store A database that stores vectors and associated data items.

B. Definitions

- Large Language Model Specialized Generative AI trained on extensive datasets can produce various artifacts from natural language prompts.
- Small Language Model A small language model is a lightweight generative AI model. It uses a small neural network, has significantly fewer parameters than a traditional LLM, and trains on tiny amounts of data.
- RAG Retrieval-Augmented Generation is a process that optimizes the response of a Large Language Model (LLM) by referencing a knowledge base outside of its training data sources before generating a response.
- Agent An LLM agent is an artificial intelligence system that utilizes a large language model (LLM) as its core computational engine to exhibit capabilities beyond text generation, including conducting conversations, completing tasks, reasoning, and demonstrating some autonomous behavior.
- Chatbot A computer program that simulates and processes human conversation.
- Retriever An interface that returns documents based on an unstructured query, making it a more general tool than a vector store. Unlike a vector store, a retriever does not need to be able to store documents. Instead, its primary function is to return or retrieve them.
- Corpus A corpus refers to the large body of text data on which the model is trained. This corpus can encompass various textual content, including books, articles, websites, and other written materials.
- Vector store A vector store, also known as a vector database, is a specialized database that can store

vectors (fixed-length lists of numbers) and other data items used in RAG tasks.

III. EXISTING WORKS

Since the inception of LLM models, researchers have continuously evaluated LLM use cases to aid the understanding of complex systems [5].

It is well-documented that performance correlates with their training data quality. In general, most of the training dataset comes from publicly available software sources [3], and often, this data can be of questionable quality or riddled with vulnerabilities. The table below is an excerpt from the above reference: Language Models are Few-Shot Learners. It shows the datasets used to train GPT-3., the base model for ChatGPT. This information was not easy to find, and I expect that this will get more difficult in time. It is also interesting to note that this is a small amount of data, approximately 570GB, after filtering. Similar corpora data breakdowns can be found for other LLMs, such as Aleph-Alpha Luminous [6].

Table 1 excerpt from "Language Models are Few-Shot Learners.".

Dataset	Quantity (tokens)	Weight in training mix	Epochs elapsed when training for 300B tokens
Common Crawl (filtered)	410 billion	60%	0.44
WebText2	19 billion	22%	2.9
Books1	12 billion	8%	1.9
Books2	55 billion	8%	0.43
Wikipedia	3 billion	3%	3.4

Table 2.2: Datasets used to train GPT-3. "Weight in training mix" refers to the fraction of examples during training that are drawn from a given dataset, which we intentionally do not make proportional to the size of the dataset. As a result, when we train for 300 billion tokens, some datasets are seen up to 3.4 times during training while other datasets are seen less than once.

In addition, LLMs cannot infer what they are producing. They produce an output that calculates the probability of each token generated next. Therefore, they are often impacted by bias. If the training data contains poor-quality references, it is unlikely that LLM will produce good results.

Furthermore, the unpredictable nature of LLMs indicates that they can produce varying outputs for identical prompts [7]. These variations are often more comprehensive than the semantic structure and the overall idea. The privacy implications concerning these models also require attention [8]. Developers are now concerned about public exposure to their questions and responses and the possible effect on system internals via security and Intellectual property leaks. In addition, the generated responses often do not consider the context of the question being asked, leading to user's mistrusting responses.

Moreover, developing such AI tools and predicting their behaviors pose significant challenges. Unlike many traditional non-AI systems, AI systems rely on complex deep learningbased transformer models, which are inherently difficult to assess. In conventional software development, we typically utilize deterministic test cases to evaluate against specific and predictable behaviors. However, given LLMs' nondeterministic nature, testing them with such deterministic test cases is a significant challenge, as the LLM returned responses may vary between the same inquiries [9], [10].

IV. USE CASES

To investigate the promise of LLM augmented learning, we create three distinct use cases (UC_n) to illustrate the different approaches for using LLMs:

- UC₁: Create an assistant to help with questions on HPC best practices, such as HPE Programming Environment, virtualization, and workflow creation.
- UC₂: Create an assistant to support internal engineers' access to and retrieval of performance data on various topics from a center corpus.
- UC₃: Create an agent that understands an HPC system environment and can interact with the system in realtime to support workflow orchestration.

A. Corpus

As already mentioned, LLMs are trained on a variety of source materials. To augment this knowledge base, we use a collection of focused materials on various specialized subjects, programming models, such as C, C++, and Fortran, internal HPE training materials, open-source workflow systems, such as ExaWorks [11], Slurm [12], and virtualization systems and HPC application execution runtimes, such as Apptainer [13], Podman [14]. The following table, Table 2, lists the sources used for the specific use cases, UC₁₋₃.

Doc Name	No. Documents	Format	Public	Private
Cray Classic C and C++ Reference Manual	1	PDF	~	
HPE Cray Fortran Reference Manual	1	PDF	\checkmark	
OpenACC	1	PDF	~	
OpenMP-API-Specification	1	PDF	\checkmark	
HPE Parallel Programming Tutorial	16	PowerPoint		√
HPE Virtualization Presentations	164	Mixed ¹		√
HPC containers		Web	\checkmark	
Workflow environments		Web	\checkmark	
HPE Performance library		Web		~

Table 2 Corpus Sources.

B. Data Collection

The use cases use LangChain [14] infrastructure for data collection and processing. LangChain is an open-source framework for developing applications using language models. Using LangChain, it was possible to create interfaces to various LLMs and supporting functions to test the validity of different use cases, UC_{1-3} .

For UC_1 , we investigate different methods of augmenting LLM training data, and we look to understand how LLMs can provide correct responses and explanations to our dataset of test questions and to determine how the LLM performs, how accurate the responses, given the input queries are not part of the foundational models training input dataset? We aim to demonstrate methods to characterize the degree of correctness of the response given by the LLM. For UC₂, we study different data retrieval methods to augment LLM training data. We look to understand how LLM can provide correct answers and explanations to our test questions and to determine how the LLM performs when questions are asked in a single chat context one by one, compared to when they are asked in cached contexts. Another aspect of this use case is to investigate the optimal way to create the input to the corpus of knowledge and measure the consistency of responses. Lastly, for UC₃, we aim to determine whether the LLM can assist the user in navigating HPC system operations and usage by providing an agent acting as a human interface to system interactions.

We discuss the core technologies for (1) augmenting LLM training data, (2) retrieval mechanisms, (3) prompt engineering, (4) LLM-based agents for system interaction, and (5) LLM correctness test methodology.

1) Augmenting LLM training data: Commonly known as Retrieval-Augmented Generation (RAG), provides additional data for LLM reasoning, avoiding the expense of retraining the foundational model. RAG aims to improve the performance of language models in tasks that require access to external knowledge or information. RAG typically involves two main components: retrieval mechanisms and a generator. Figure 1 shows the processing of a user query using a local data store, passing the new context to the LLM and the response.

- A. Input query: Process an input query and prompt.
- *B. Query: retrieving relevant information from a data store seeded from a corpus.*
- C. Enhanced context: the information retrieved is now used as context for the generative component of the model. This context provides the model with the necessary background knowledge.
- D. The generator takes the retrieved information and the original input query to generate a response. This step is typically performed using a language model,
- *E.* The response from the language model is returned to the user.

C. Methodology

¹ Mixed documents: PDF, Markdown, Web, Confluence and Text



Fig. 1. Retrieval-Augmented Generation – RAG.

2) Retrieval mechanisms: To utilize RAG, we must bring appropriate information and insert the data into the model prompt. A typical RAG system has two main components: indexing, retrieval, and generation. Indexing is ingesting data from sources/corpus and indexing the data, typically using a vector store database. Indexing sub-tasks can be further characterized by loading documents using various document loaders, such as web-based, CVS, text, PDF, etc., and then splitting these documents into smaller chunks, as shown in Figure 2. This is useful for indexing data and passing it into a model since large chunks are more complicated to search and won't fit in a model's finite context window. Finally, the split data is stored for later search. This is often done using a vector store database with various embedding models.



3) Prompt engineering: Prompts are the inputs LLMs provide to produce a desired response. Well-designed prompts are crucial to get useful outputs from LLMs. Prompt engineering aims to construct prompts that deliver accurate, relevant, and helpful LLM responses. The simplest form of prompt engineering is where the developer crafts the user (query) and system (context) prompts to serve a better response and limit the ability of the LLM to hallucinate. By carefully crafting the system prompt, the developer can create a set of guardrails [15] to protect the application against hallucinating and help guide the responses to the developer's intent. Another viewpoint is that the user doesn't know what the query should be; by using a new concept, RAG-Fusion [16], we will show the model can produce better and more consistent answers to user

queries by using an LLM to generate similar queries based on the user input, and rank the responses, selecting the best fit. As shown Figure 3, RAG-Fusion will generate additional input queries based on the original input, pass all these queries to the LLM, and then rank-sort the responses, giving the topmost response back to the user.



In Figure 4, we use Phi-2, a small language model (SML) from Microsoft, to illustrate how different RAG retrieval methods can enhance the LLMs response to a question on which the foundation model has yet to be trained.



Fig. 4. SLM RAG Responses.

4) LLM-based agents for system interaction: An agent uses a language model as a reasoning engine to determine which actions to take and in which order. The agent executes these actions. We look to understand 1) the ability of the LLM to produce quality responses to the user's input query and whether the response accurately encapsulates the intent and 2) mechanisms to safeguard the agent execution of the code, such as sandboxing and Unix permissions.

5) LLM Correctness Test Methodology: We evaluate different strategies for evaluating LLM responses. Responses generated by LLM agents produce variable outputs, contexts, and meanings. The challenge is reconciling multiple responses and grading the responses to a known ground-truth answer. We could employ human evaluation, but this strategy would consume too many resources. Instead, we adopted a pragmatic approach. As illustrated in Figure 5, the evaluation framework depicts the evaluation pipeline using benchmark question datasets (GMS8k, Slurm, and HPE Development Environment).



Fig. 5. Multi-LLM Reasoning.

We use a question, q1, from several question datasets and compare the LLM answer a' to the ground truth answer, a1, as defined in the test dataset. For this comparison, we utilize an evaluator LLM (LLMe) to apply correctness criteria to grade the response from the LLM under test (GPT, Llama, etc.,) with the ground-truth answer,

a) Naïve Evaluation methodology: To verify the effectiveness of the LLM responses, we first adopted the LangChain LLM comparator functions to evaluate how close the answer is to ground truth, e.g. $a' \equiv a1$; these results were inconclusive on a sample set of GSM8K questions/answers.

Metric	True	False
cosine	0.0687042	0.0715222
euclidean	0.370687	0.378212
manhattan	11.5896	11.9183
chebyshev	0.0319455	0.0317866
mamming	1	1

Table 3. String Evaluation Criteria

Using these stock criteria methods, we could not assign a consistent value to the LLM response vs. ground truth, so we developed better test criteria, as documented here.

b) LLM Evaluation Methodology: We use an LLM evaluator (LLMe) that uses custom reasoning criteria (Table 5), which demonstrated better reasoning and evaluation for the sample set of LLM Q&A. Table 5 shows the custom accuracy criteria used to self-evaluate the LLM response to the input question. We take any score greater than 3 as the LLM has determined the positive response.

Table 4: Reason Criteria

C 1	
Score 1	answer is entirely unrelated to the reference.
Score 3	answer has minor relevance but does not align with the
	reference.
Scare 5	answer has moderate relevance but may contain inaccuracies.
Score 7	answer aligns with the reference but has minor errors or
	omissions.
Score 10	answer is completely accurate and aligns perfectly with the
	reference.

Table 5 shows the results from two LLMs using the above reasoning criteria and samples from the GSM8K dataset.

Table 5: LLM Reasoning Score

GSM8K Question	GPT-4 Reason Score	Llama2 Reason Score
1	10	5
2	7	3

V. RESULTS

A. UC1: Evaluating the correctness of RAG-based queries against stock LLMs. The first evaluation is to determine the understanding of stock LLMs in the context of our use cases.

1) LLM performance: The results for "explanation of correctness." We use the GSM8K (Grade School Math 8K) dataset, containing 8.5K high-quality, linguistically diverse grade school math word problems. Methods used: GMS8K is a well-known benchmark dataset for evaluating LLM accuracy. We created several LLM Chains using different models, as shown below. The answers to the questions are then reevaluated using ground truth responses and another LLM to score the response. The algorithm yields a 1 for correct and 0 for incorrect. As expected, the larger commercial LLMs outperform the smaller Open LLM models. What is interesting is how well the newer Open LLM models, such as Mixtral MoE, are getting close to commercially available models.

2) Standard LLM Models GSM8K responses: We can use the above testing methodology to determine the scores for the UC1. Figure 6 illustrates how different LLM models responded to questions using the GSM8K benchmark.



3) Enhanced Retrieval Process: While RAG can improve the accuracy of LLM responses to user queries, additional optimization methods exist for the different stages of RAG pipelines. One area is semantic search. To improve the LLM response's quality further, we introduce the concepts of RAGfusion [16]. The theory behind RAG-fusion is to generate multiple versions of the user's original query using an LLM and then re-rank the results to select the most relevant retrieved parts. Again, we show a better return than standard RAG using the abovementioned dataset. Figure 7 shows the performance of using open-source LLM model's reasoning capability using a set of questions designed to test understanding of development concepts.



Figure 8 shows the performance of using open-source LLM model's reasoning capability using a set of questions designed to test understanding of Slurm concepts.



The next two figures, Figure 9 and Figure 10, show the performance of using open-source LLM model's reasoning utilizing RAG to enhance the accuracy of the LLMs response to the same set of questions.



Fig. 9. LLM/RAG Test Scores for HPE DE.



B. UC₂: Create an assistant to support internal engineers' access to and retrieval of performance data on various topics. For UC₂, we leveraged much of the work used to evaluate UC1. The difference for UC₂ is that we implemented a Chatbot-style interface using Streamlit API and offered several RAG mechanisms. This use case delves into fine-tuning document retrieval methodologies and shows the correct results for answer explanation based on private LLM instances. Given that we already have the expected text in the performance corpora, validation of answers is straightforward, albeit non-trivial. We implemented several retrieval schemas for this use case. Figures 11 and 12 show the difference in response between a standard foundational model and a model enhanced with RAG.

The question being asked was:

"describe how I would use Apptainer to launch an MPI application."

To launch an MPI application using Apptainer, you can follow these steps:

 First, make sure that you have Apptainer installed on your system. If you don't have it installed, you can download it from the official website.
Open a terminal window and navigate to the directory where your MPI

application is loaded. 3. Type "Apptainer" followed buy the name of your MPI application, and press enter. For example, if your application is named "my, mpi, app" you would typ

enter. For example, if your application is named "my_mpi_app", you would type "apptainer my_mpi_app". 4. Apptainer will then launch your application, Apptainer may prompt you to

select the number of CPUs or nodes you want to use for the application. 5.Once the application is launched, you can interact with it using standard input/output methods, such as stdin and stdout.

6. When you're done with the application, you can type "exit" to terminate it

That's it! With these steps, you should be able to use Apptainer to launch your MPI application in a containerized environment. If you have any further questions or need additional assistance, feel free to ask.

The response from a standard LLM is very generic and misleading. Meanwhile, the RAG response, as shown in Figure 12, is augmented using data from HPE training documentation.

To launch an MPI application using Apptainer, you can use the following command:

apptainer run <container-name> <MPI-command>

For example, if your container is named mycontainer and you want to run the mpirun command with 4 processes, you would use the following command:

apptainer run mycontainer mpirun -n 4 /path/to/your/application

This will launch the MPI application in your container using the MPI implementation available on the host.

Note that you need Apptainer installed on your system and have access to the container you want to use. You also need to make sure that the MPI implementation is available on the host and that the container has access to it.

Fig. 12. LLM/RAG response to question

The same approach created an interface to search HPE propriety information. The difference was how we constructed the local document vector store from confluence pages rather than documents and public web sources. Confluence access tokens are supplied to enable the retrieval of private data using the RAG methods. As before, results obtained using RAG were more concise than those without.

C. UC₃: Create an agent that understands an HPC system environment and can interact with the system in real-time to support workflow orchestration.

For UC₃, we further expand on the previous use cases $(UC_{1,2})$, and we apply the lessons learned to develop a shellbased assistant that interacts with the user and transforms LLM responses to system inputs. We evaluate the correctness across a sample set of answer-explanation pairs and investigate the security and efficacy of such operations.

This use case created a CLI agent to assist the users in navigating a complex HPC system environment. The agent was evaluated to solve human-system interaction in the context of workload management. The agent was often able to propose plausible action and execute both individual and multiple-step commands.

In this section, we describe the agent's ability to solve some of the more challenging tasks in the suite to illustrate a range of capabilities and limitations. Figure 13 illustrates how the agent can transform a user's question into system commands and execute them. The response from the LLM is returned in a format that is easily understandable to the users. This simplifies the current process, where users have to write a series of commands and scripts to obtain the same information.

_> list all nodes in the cluster The nodes in the cluster are: o184i[027,038-039,060-061,073,077- 078,080,083,086-087], o185i[086-087,084,164-167], o186i[180-187,190- 197,225,233-234], o184i[110-125], o186i[130-137].
-> show all slurm running jobs There are currently two jobs running on the cluster. The first job, with ID 140610, is named 'det-ai_c' and is running in the 'champollion' partition on node ol86i225. The second job, with ID 140613, is named 'bash' and is running in the 'misc_gpus' partition on node ol85i087.
_> What is the node configuration of o184i082? Node o184i082 has 2 sockets, each with 64 cores. It has a total of 128 CPUs and 1TB of memory. The node is currently idle and has 8 AMD MI210 GPUs available.
> show all nodes which have a GPU The nodes with a GPU are o1851086, o1851087, o1841082, o1841084, and o1841085.

Fig. 13. Assistant CLI

To mitigate user concerns about security and limit the scope of interactions, we implemented several "guardrails" to impose secure use boundaries. We implemented guardrails using three gates: (A) sandbox, (B) validity, and (C) authentication.

- A. Sandbox: Only run commands in a sandbox environment, limiting the LLM from generating commands outside the restricted availability set.
- B. Validity: Limit the command set to known utilities. For Slurm, these would be srun, sbatch, etc.
- C. Authentication: Use access to run commands based on Linux access permissions.

If the LLM-generated command passes the guardrails, the command is executed in the user's context, and the response will be returned to the agent, parsing the output and returning an LLM representation to the user. The output returned is not the same as the output from the actual Slurm command; instead, it is a transformed representation. As can be seen in Figure 14, responses to the user question.



VI. DISCUSSION

In this study, we examined and demonstrated the potential of using LLMs as assistants to aid knowledge discovery and as a tool to navigate complex systems. Furthermore, we created several closed systems, using open-source models to provide both security and mobility of the solutions. We found that standard foundational model LLMs can provide correct or partially correct answers of a sample from a standardized benchmark GPT4:72%, GPT3:58%, luminous:0.0%, llama2:38%, mixtral:74%, Phi-2:50%. In the Luminous case, the model comprehension of the GSM8K questions was not performing well, and we still need to understand why the text caused the model issues today. Case in point: the reasoning of the following question

"Janet's ducks lay 16 eggs per day. She eats three for breakfast every morning and bakes muffins for her friends every day with four. She sells the remainder at the farmers' market daily for \$2 per fresh duck egg. How much in dollars does she make every day at the farmers'."

The luminous model interpretation was:

"A; First find the total number of eggs Janet eats: 16 eggs/day * 3 breakfasts/day = <<16*3=48>>48 eggs Then find the total number of eggs she sells: 48 eggs - 4 eggs = <<48-4=44>>44 eggs Then multiply the number of eggs by the price per egg to find the total revenue: 44 eggs * 2/egg = <<<44*2=88>>88 #### 88"

The model misinterpreted the "eats three" as a multiplication.

Use prompt engineering, guardrails, and RAG retrieval methods as chatbots for the HPE Development environment, as shown in Table 6.

Table 6: HPE DE model correctness scores.

Model	LLM Correctness	RAG Correctness
Phi-2	68	68
Llama2	74	74
Mistral	78	90
Mixtral-MoE	84	95

And Table 7, Workload manager responses.

		Table	7:	Slurm	model	correctness	scores
--	--	-------	----	-------	-------	-------------	--------

Model	LLM Correctness	RAG Correctness
Phi-2	11	11
Llama2	44	55
Mistral	77	88
Mixtral-MoE	100	100

As expected, using RAG helped correct the LLM response when answering domain-specific questions. Furthermore, we investigated using agents as system actors to aid users in navigating system configurations and interactions. We found that the use of LLMs can provide offload knowledge capabilities. In applying the agent, we noticed the security challenges of allowing an autonomous agent, which required special consideration and additional safeguards around executing within a secure jail. For this use case, the goal was to create an expert system to translate natural text to schedulerspecific commands and respond with translated output. Said differently, natural text processing for batch system interface. The interface was implemented as a shell interface to mimic the typical system use pattern. With all agent implementations that interact with a system, security is a significant concern, as the LLM is responsible for creating the command and options autonomously. A level of trust is being granted to the agent, which needs to be understood and limited. We implemented several guardrails as an initial measure against rogue commands and a mechanism to limit capabilities.

Our results for the agent also showed that we required a model with significant training and modality, meaning language support. The bigger the model, even with RAG, the better the user comprehension and interaction were. For instance, switching the language from English to French was trivial and the reverse. A notable observation was how the agent was chaining together the workstream. There are two invocations of the LLM: the translation from the user query to the Slurm command and the translation of the Slurm output back to the user. This second invocation took significant time; the larger the LLM, the faster the second invocation became.

VII. CONCLUSION

This work delves into the use and accuracy of LLM-based agents, focusing on three distinct use cases. Initially, we assessed various LLM providers and models to show the correctness of LLM responses to specific HPC-related scenarios. Subsequently, we investigated multiple methods to improve generated responses augmented by relevant materials and adopted different retrieval methods to explore the accuracy and correctness of responses. Finally, capitalizing on the prior work, we conducted a comprehensive evaluation and discussion to scrutinize the use of interactive agents to assist the user in system use. This analysis provides valuable insights and directions for future research endeavors in the domain of LLMpowered assistants in HPC systems.

Furthermore, our study demonstrated that open-source RAGenabled models perform well against commercial LLMs, with the added benefit of privacy and the use of a focused knowledge base. One surprising observation was using augmented small language models, such as Microsoft Phi-2. We believed or hoped that by providing enough local documents, these models would, at minimum, give the same responses as the larger LLM or the MoE LLMs. Sadly, this research did not prove this, and these SMLs didn't have enough reasoning capabilities required by this investigation.

As more focus is placed on using LLMs in autonomous agents, further study is required, especially around security and capabilities. This study also found that LLMs can generate bogus information, commands, and options; guardrails can mitigate command exposure by limiting which commands the agent can access. Considering the diverse nature of system setups and configurations, having an agent that can translate free-form text to scheduler context was very helpful within the test environments. One could see a future where agents are a general onboarding tool to assist users in navigating HPC systems for all aspects of system use, development, knowledge transfer, execution, and system operations.

ACKNOWLEDGMENT

The authors thank John Levesque and HPE leadership for supporting this study.

REFERENCES

 ChatGPT. (GPT-4). OpenAI. Accessed: Mar. 26, 2024. [Online]. Available: <u>https://chat.openai.com/chat</u>

- [2] Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., Rodriguez, A., Joulin, A., Grave, E., & Lample, G. (2023). LLaMA: Open and Efficient Foundation Language Models. ArXiv. /abs/2302.13971
- [3] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017). Attention Is All You Need. ArXiv. /abs/1706.03762
- [4] "LangChain application from prototype to production", https://www.langchain.com. [Accessed: Jan. 29, 2024]
- [5] Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., . . . Amodei, D. (2020). Language Models are Few-Shot Learners. ArXiv. /abs/2005.14165
- [6] "Aleph-alpha Luminous Training Data", https://docs.alephalpha.com/docs/introduction/model-card/#training-details [Accessed: Apr. 5, 2024]
- [7] "Output from AI LLMs is Nondeterministic. What that means and why you should care.", https://www.sitation.com/non-determinism-in-ai-llmoutput [Accessed: Mar. 7, 2024]
- [8] Yifan Yao, Jinhao Duan, Kaidi Xu, Yuanfang Cai, Zhibo Sun and Yue Zhang (2024). A Survey on Large Language Model (LLM) Security and Privacy: The Good, the Bad, and the Ugly. arXiv:2312.06717v2
- [9] T. H. Kung, M. Cheatham, A. Medinilla, ChatGPT, C. Sillos, L. De Leon, C. Elepano, M. Madriaga, R. Aggabao, and G. Diaz- Candido, "Performance of ChatGPT on USMLE: Potential for AI- Assisted Medical Education Using Large Language Models," medRxiv, 2022.
- [10] J.H.Choi,K.E.Hickman,A.Monahan,andD.B.Schwarcz,"ChatGPT Goes to Law School," SSRN, 2023.
- [11] Exaworks. (2024, Apr. 7). Technologies for Composable and Scalable HPC Workflows [Online]. Available: <u>https://exaworks.org</u>
- [12] Slurm. (2024, Apr. 7). Slurm workload manager [Online]. Available: https://slurm.schedmd.com
- [13] Podman. (2024, Apr. 7). A tool for managing OCI containers and pods [Online]. Available: <u>https://podman.io</u>
- [14] LangChain. (2022, Oct 17). LangChain is the platform developers and enterprises choose to build AI apps from prototype to production [Online]. Available: <u>https://github.com/langchain-ai/langchain</u>
- [15] Guardrails. (2024, Apr. 7). How to implement LLM guardrails [Online]. Available: <u>https://cookbook.openai.com/examples/how_to_use_guardrails</u>
- [16] Rackauckas, Z. (2024). RAG-Fusion: A New Take on Retrieval-
- Augmented Generation. ArXiv.https://doi.org/10.5121/ijnlc.2024.13103
- [17] GSM8K (Cobbe et al., 2021). Grade School Math [Online]. Available: https://github.com/openai/grade-school-math