Advancing HPC skills by Developing Large Language Model Retrieval Augmented Generation (LLM-RAG) Systems

Julia Mullen
MIT Lincoln Laboratory
jsm@ll.mit.edu

Sam Corey
MIT Office of Research Computing and Data
secorey@mit.edu

Lauren Milechin
MIT Office of Research Computing and Data
milechin@mit.edu

Riya Tyagi
MIT Office of Research Computing and Data
riyaty@mit.edu

Daniel Burrill
MIT Lincoln Laboratory
daniel.burrill@ll.mit.edu

ABSTRACT

Large Artificial Intelligence (AI) and generative large language models (LLM) are key computational drivers. For researchers developing new tools or incorporating LLMs into their processing pipeline, the scale of data and models require supercomputing resources which can only be met through cloud or High Performance Computing (HPC) architectures. Many of these researchers have deep experience with AI, LLMs, and their research area but are new to HPC concepts, challenges, tools, and practices. To assist this researcher community, the Research Facilitation Teams at MIT Office of Research Computing and Data (ORCD) and the MIT Lincoln Laboratory Supercomputing Center (LLSC) have developed tutorial materials to teach researchers how to build their own Retrieval Augmented Generation (RAG) workflows. Selecting RAG systems as the project focus provides motivation for developing a wide range of skills necessary for efficiently working with LLMs on an HPC system while creating a useful application.

This work details LLM-RAG implementation concerns on two different systems, the design decisions associated with developing the examples, deployment of the workshop training, and the feedback received from the participants. Both the MIT ORCD and MIT LLSC systems are representative of HPC community systems and we plan to refactor the in-person and live virtual workshops into a micro-course built from online, self-paced modules that will be reusable across other HPC centers with slight modifications.

KEYWORDS

High Performance Computing, Artificial Intelligence, Generative AI, LLM, RAG, HPC-LLM Training Modules

1 INTRODUCTION

Large Artificial Intelligence (AI) and generative large language models (LLM) are key computational drivers. For researchers developing new tools or incorporating LLMs into their processing pipeline, the size and scale of data and models require supercomputing resources. The resource needs can be met through cloud architectures or High Performance Computing (HPC) systems. Many of these researchers have deep experience with AI, LLMs, and their research area but are new to HPC concepts, challenges, tools and practices. The paradigm shift from existing tutorials on YouTube and HuggingFace where examples are designed for personal systems to the tasks associated with using a community supercomputing system is massive. As part of this shift researchers need to learn how to install code and packages in user space, select the appropriate compute and memory resources, and run their applications through a scheduler. Additionally, they need to understand the basics of distributed computing in order to understand the challenges associated with running large applications that require multiple nodes for processing.

There are ways to lower these barriers to entry, including the use of containers, modules, well designed teaching examples, and user-guided tutorials. Based on researcher demand, the authors elected to explore approaches to supporting researcher efforts through the development of LLM Retrieval Augmented Generation (LLM-RAG) systems.

A cursory review of in-depth user guides and written tutorials provided by HPC centers highlights two broad approaches for creating LLM-RAG examples that researchers can emulate: provisioning containers [9] and LLM-RAG systems created with a combination of bash and python scripts [3, 5, 10]. In this paper we examine both approaches to building LLM Retrieval Augmented Generation (RAG) systems on an HPC system. The authors represent two distinct supercomputing centers with two distinct approaches to deploying LLM software and training for the research community. Note that when we refer to training in this paper we are referring to educational training.

2 TECHNICAL APPROACH TO EXAMPLE DEVELOPMENT

At a high level an LLM-RAG pipeline executes the following steps:

(1) Load the data set into a machine readable format
(2) Embed the data into a vector database, i.e. vectorstore
(3) Create a workflow that chains the embedding model, vectorstore, and language model together to read user inputs and use them to augment queries and return answers

https://doi.org/10.22369/issn.2153-4136/17/1/4

Table 1: Key Implementation Differences between MIT ORCD and MIT LLLSC

Configuration	MIT ORCD	MIT LLSC
Internet access from compute nodes	Yes	No
Offline model repository	No	Yes
Multi-node examples	No	Yes

The implementation of the LLM-RAG pipeline examples on the MIT ORCD and MIT LLSC systems is impacted by the differences in how these systems are configured and the policies that are applied. The key differences can be seen in Table 1.

In each case, the authors used the Python programming language and the Langchain [1] suite of packages to create the pipeline.

2.1 Deployment via Apptainer

ORCD deployed LLM-RAG using a world-readable Apptainer [11] image containing a prebuilt Anaconda environment to run on our Engaging cluster. We chose container technology over other approaches to minimize the setup time, allowing more class time to be spent interacting directly with the RAG pipeline. To further streamline our deployment, we distribute a shell script to run the pipeline, thus allowing users to interact with the LLM agent using a single command from the command line. The script uses the online ORCD documentation as a knowledge base by default, and with a second shell script users can create an alternative vector store with their own set of documents. The ORCD documentation exists in the form of Markdown files which are generally well suited for vector store creation.

By default, our pipeline uses Mistral AI's 8B Instruct LLM [6]. We chose a small model to reduce download time for our users, limit the GPU resource requirement, and save space on our systems. Furthermore, we found Mistral's models to be the most practical for our use case because they are open source, readily available through HuggingFace, and well-suited for LLM agents. While we explored saving the model in a global location on our cluster to be shared by all users to avoid multiple individual downloads, we found that the download process was a good opportunity for users to learn about storage and bandwidth. Many users might not understand the computational scale of LLMs, and witnessing how many resources are required even for relatively small models should influence their point of view as they develop their own AI workflows.

Our deployment allows for varying levels of engagement among our users. For those who simply want to interact with the LLM agent, they can do so with a single shell command. For users who want to easily customize the pipeline to fit their needs, we added flags that can be used when running the shell scripts, which allow users to set different knowledge bases or change the temperature of the LLM. Advanced users seeking deeper customization, such as enabling multi-GPU inference for larger models or experimenting with different LLM prompt contexts, can treat our code base as a flexible foundation. Its design makes it easy to adapt and extend for more complex use cases.

2.2 Deployment in Offline Mode

To avoid sharing data, queries, and results across the open internet, many organizations discourage or forbid access to public, open LLM tools for professional work. In these environments, researchers require offline models that can be used locally. Downloading models to a laptop quickly becomes infeasible, and researchers often turn to approved cloud providers or organizational clusters. While the use of organizational clusters enables research teams to work from a single system, resource requirements for ever larger models risk overwhelming a shared system. To support the requirements of Lincoln Laboratory projects engaged in LLM research, the LLSC provides a fully on-premise solution. Equiping a useful system for researchers requires providing access to an ever-expanding set of models and a means of serving the models to applications.

The LLSC provides access to open language models through a shared directory on the central file system. Publicly available models used by multiple research groups are downloaded and staged by the LLSC team. To avoid accidental deletion or corruption, the directory is set with read and execute permissions, while only members of the LLSC team have write permission. If a research team needs a model that is not open, they are advised to request a shared (Linux) group and stage the model in the group protected directory.

For models that are small enough to fit on a single node, providing users with the path to the models is enough to get them started. For larger models that require more GPUs than those on a single node, researchers need a way to serve a distributed model from multiple nodes to their application. To address these concerns, the LLSC has chosen to use vLLM [4] because vLLM

- handles communication between nodes,
- coordinates GPU memory through optimized CUDA kernels,
- offers fast model execution with CUDA/HIP graph,
- is well integrated with HuggingFace models, and
- provides an OpenAI compatible API server.

This approach provides enormous flexibility for the researchers allowing them to select a model at run-time by specifiying the model and resources requirements, e.g., number of GPUs, in the submission script. With templated scheduler submission scripts, users only need to change one or two parameters to select a new model for testing. However, one challenge with the LLSC's initial implementation of the vLLM approach was that users needed to develop a fairly deep understanding of Linux, bash scripting, and scheduler (resource manager) use in order to be fully effective. Through multiple iterations of the learning module these challenges have been simplified but not eliminated. The process of training researchers to migrate from more open environments to the on-premise solution is discussed in Section 3.

3 TEACHING AND TRAINING APPROACH

The choice of LLM-RAG provides the training team with the initial opportunity to teach researchers about user space, basic Linux commands, and the shared needs of community systems. The scope of models used in RAG systems motivates the need to better understand how to use GPUs efficiently in a shared community system. For both teams, the LLM-RAG pipeline created the opportunity to build HPC skills using a Just-In-Time approach.

3.1 Learning Objectives

We designed our educational material with two goals in mind: (1) to teach users how to run and customize an LLM agent on a computing cluster and (2) to demonstrate best practices for HPC more broadly through the use of an example application. This results in two sets of learning objectives for students, which we describe below.

3.1.1 LLM-RAG learning objectives. The learning objectives associated with developing and running LLM-RAG pipeline include theoretical concepts, practical implementation concerns and troubleshooting skills. In particular, the teams designed the training so that learners completing the workshops, would

- understand the components of LLM-RAG pipelines, including vector stores, embedding models, and LLM context tuning,
- be familiar with implementing a RAG pipeline using provided code
- be able to modify the RAG pipeline based on individual needs,
- be able to create a vector store based on any set of documents,
- be able to pre-process a set of documents to minimize troubleshooting of vector store creation,
- understand how to launch a vLLM server through an HPC scheduler (MIT LL), and
- be able to use LLMs on a supercomputing/HPC system using GPUs.

3.1.2 HPC learning objectives. While the set of learning objectives associated with running a LLM-RAG pipeline on a supercomputing system were the key drivers of the training, success required that learners develop an understanding about HPC systems and the skills to use them. In practice this meant that the training teams had an opportunity to include a number of general HPC skills in the training. Thus, at the end of the workshop, not only did the learners develop the skills in the previous section, but they also

- understood what containers were and when to use them,
- were able to use containers in an HPC setting (MIT ORCD)
- understood the role of an HPC resource manager (scheduler) (MIT LL),
- were able to submit jobs and monitor jobs through the resource manager (MIT LL),
- gained greater familiarity with Linux command line use,
- had experience monitoring GPU utilization for memory and compute,
- had experience configuring and confirming Python environments in user space,
- had experience configuring user shell environments more broadly, and
- understood best practices for file system use when downloading data and models.

In the next section we discuss how these learning objectives were woven into the course material, through the lecture/presentation and the hands-on practice.

3.2 Workshop Design and Delivery

3.2.1 ORCD workshop. ORCD offered an in-person, two-hour course on running the RAG pipeline as part of the summer HPC course series presented to interested learners in the MIT community. The course was accompanied by a "How-To" guide included in our online documentation. Based on a pre-course survey, learners came from a variety of disciplines at MIT and mostly consisted of graduate students and postdoctoral scholars.

The ORCD team designed our workshop to be highly practical, focusing on hands-on application rather than delving deeply into the theoretical mechanics of LLM agents. That said, we did provide a brief overview of key concepts, including visual explanations of RAG pipeline architecture and the process of embedding documents to build a vector store. This was intended to provide our users with just enough context so that they could make any necessary customizations to the pipeline later on.

The workshop was also highly interactive, with most of the of the teaching taking the form of live coding, which allowed students to follow along on their own. Frequent pauses were taken to ensure that students were not left behind and personal assistance was provided to students as needed. We aimed for a low instructor-to-student ratio, with one teacher and two helpers for every 20 students. Helpers were present to provide one-on-one support to students who had user-specific issues that may not apply to the rest of the class, such as unique environment configurations or storage limitations.

There were a few setup steps that users were required to follow before they could start interacting with the LLM-RAG agent. We informed users of some of these steps in the days leading up to the workshop, but still went over them in person. Users needed to create an account on HuggingFace and generate a user token in order to gain access to the pre-trained LLMs and embedding models used in our setup. Then, from the command line, we went over saving the tokens to individual shell initialization scripts, which we used as an opportunity to teach about shell configuration on shared systems.

We then walked users through requesting necessary resources to run the pipeline. Users connected to our cluster using the shell access to login nodes available through our Open OnDemand web portal [2]. We chose this route over SSH connection to limit the barrier to accessing our systems and ensure a consistent experience among our users, circumventing any individual differences in operating systems and SSH clients. After connecting to a login node, we showed users how to start an interactive job using a GPU. Our Engaging cluster has L40S GPUs, each with 46GB of memory, which is enough to run inference on the Mistral 8B model.

The overall flow of the workshop was intended to start as simple as possible and increase in complexity as time went on. After setup, the first exercise that students performed was running the single shell command that allowed them to interact with the LLM-RAG agent right away. Students initially had to wait for the model to download, during which time we walked through the code and general architecture of the pipeline.

Then, we evaluated the agent by posing different types of prompts. The goal in this portion was to teach the students how to assess the agent's ability to pull information from the knowledge base. In RAG, it is sometimes difficult to know when the agent is generating responses based on parametric memory (i.e., knowledge encoded in the parameters of the LLM) or non-parametric memory (i.e., knowledge gained from external sources of information, such as a vector database of documents). To illustrate this concept, we

asked the agent questions that were deliberately absent from the knowledge base (such as, "When did the United States declare its independence?"). The agent would provide a response but would not always elaborate on whether relevant information was present in the provided documents. In the class, we facilitated student input on how to manage this issue.

After experimenting with the LLM-RAG agent on the default ORCD online documentation, we showed students how to create an alternative knowledge base from on another set of documents, and then run the pipeline using the newly created vector store. The example document we used for this portion was a PDF of MIT's Wikipedia page, chosen for its simplicity and relevance. We walked students through uploading files to our computing cluster and running the vector store generation script with necessary flags that point to the new documents. We ran further example prompts using this new agent. The prompts revealed that the agent was less adept at pulling information from the new vector store, suggesting a difference in the embedding model's capacity to encode information from PDFs versus Markdown files.

At the end of the class, students were given the opportunity to try running the LLM-RAG agent using their own documents. Many students came to class prepared with PDFs of academic articles, technical documentation, or other domain-specific sources of information. This portion of the class was highly interactive and collaborative, with many students needing assistance understanding the filesystem on our cluster as well as curating their documents to fit the pipeline. This portion created opportunities to discuss various HPC concepts with students, including memory, GPU utilization, and storage.

3.2.2 LLSC workshop. The LLSC HPC-RAG training module was initially delivered as part of an 8 week virtual professional education course on Large Language Models. Additional offerings included a hybrid short course on the theory and practice of LLMs and a second run of the 8 week virtual LLM course. In each case, the goal of the HPC-RAG module was to provide students with hands-on exposure to LLMs on an HPC system prior to the start of their student-defined, mentor-guided project. We note that the mix of students included many who haven't written code in years and others with deep backgrounds in LLMs but limited experience with HPC systems.

The long courses were designed so that each module, or week, included online asynchronous learning material: videos, articles, and hands-on practice along with a 90 minute live, synchronous session. When designing the RAG module we reviewed the skills and knowledge that students needed to work independently, the information required for setting up their HPC accounts to work through the RAG example, and the skills required to be successful with the example. Based on the review, the asynchronous online preparatory work included

- an introduction to HPC including a short video about the components of HPC systems and how they differ from a laptop,
- information on accessing the HPC system through the web portal including a short question to confirm access,
- information on Jupyter Lab and Jupyter Notebooks including how to start them on the system and how to use them, and
- best practices for Jupyter Notebook use.

Similar to the ORCD workshop, the live, synchronous session included a presentation introducing and explaining RAG pipelines, their architecture, and the hands-on RAG activity. The hands-on portion of the session used live coding extensively within Jupyter Lab running on the HPC system. As with ORCD's workshop, this approach was chosen to ensure access and a consistent environment for all students. To work with the RAG example, students needed to first start a terminal in Jupyter Lab and submit a job to the scheduler to start the vLLM server. Once the server was running, they used the Python scripts provided to set up and run the RAG pipeline. To provide additional assistance for students during the hands-on activity, LLSC staff from the Computational Science and Engineering team joined the session and used breakout rooms to work with students. Student feedback on the module was very good and students suggested that a re-run of the course should include more interactive modules.

While the students were successful during the synchronous session, when they started working on their guided projects, their first challenge was the lack of internet connectivity from the compute nodes. Most researchers are used to using "sudo" or "pip" to install packages as needed, sometimes even as they start their application, so the lack of internet access caused some confusion. The work around is to set up the compute environment from either a login or download node before starting work. Students struggled, though documentation was provided. To address this the training team re-evaluated the learning objectives in preparation for updating the module prior to the second run of the 8 week course.

The "Introduction to LLMs: Theory and Practice" short course provided an ideal opportunity to refactor the initial module and test it with a live class. There were two significant changes in the learning material. The first resulted from re-factoring the way that we addressed the challenges associated with the lack of internet connectivity from the compute nodes. The new version included content about required tasks resulting from the constraint. In particular, the newly created training materials built on an analogy to traditional HPC processing where applications require completing pre-processing, processing, and post-processing tasks. In the context of LLMs or interpreted packages in general, the pre-processing stage (Phase 1) is the creation and confirmation of the user's computing environment. The processing stage aligns with the execution of the LLM pipeline and application. Students in the short course found this separation of the two phases clear and easy to follow. They also understood that this approach would be required anytime there were changes in the environment, such as new or updated packages. This refactoring has the additional benefit of motivating the need to understand Linux environments, how to use modules, and how to troubleshooting environment issues.

The material in Phase 1 required students to use the Linux command line, load module files, install packages in user space, confirm package installation, and test the environment. In some cases, students had to start the Python interpreter at the command line to download or update additional packages. The ability to do all this highlighted their HPC skills and prepared them for their individual projects in addition to the RAG pipeline example.

The second major change was to convert the scripts associated with setting up and running the RAG pipeline into cells within a Jupyter Notebook. This new approach meant that students were still required to use the Linux command line to submit a job to the scheduler to start the vLLM server, but once started they were able to explore the pipeline in a Notebook. One advantage of this approach was the ease with which new cells could be added and used to experiment with a range of prompts. The output from previous cells was still available in the Notebook making it easy to compare the prompts and outputs.

To increase the usability of the RAG example, the second 8 week course was augmented with an extra short lab in Module 3, the week prior to the RAG pipeline (Module 4). The augmented lab required students to complete the Phase 1 steps during the Week 3 asynchronous work. In addition, they were asked to upload documents related to their projects. During the synchronous session for Module 3, students used and modified a Jupyter Notebook to load their documents, select an embedding model and embed their documents in a vector store. This redesign had multiple benefits: (1) because some documents had errors during the embedding it prompted a discussion about data preparation, (2) at the end of the session student documents were embedded in a vector store and ready to use for the RAG example, and (3) we uncovered and resolved issues that would impact the success of the RAG example the following week.

4 LESSONS LEARNED

The MIT ORCD and LLSC teams took away many lessons from the RAG training workshops—some related to provisioning the tools and others related to training.

4.1 Training

The examples created by the MIT ORCD and MIT LLSC teams were developed and tested across multiple training events. Through this process, the authors learned:

- **Providing templates is helpful, but users still need context.**
  The shell scripts administered by ORCD and the Jupyter Notebook administered by LLSC allowed students to run things quickly, but a more thorough understanding of the pipeline is necessary for a student to personalize things to fit their own use case.
- **Providing editable scripts that are very similar to default scripts encourages users to experiment in a scaffolded environment.**
- **Converting demonstration examples to teaching examples requires explaining what the steps are and why they are necessary.**
- **Teaching proper HPC resource utilization cannot be ignored.**
  Students who are new to HPC or LLMs may not have a sense for the resources required to run these models. Many students are eager to start experimenting with larger models, but without understanding their computational cost, they may begin to monopolize resources that they do not need. For example, an idle Jupyter Notebook that has multiple GPUs allocated has a negative impact on other researchers in the community who need to run their own computational tasks.
- **Unique use cases may cause the pipeline to break.**
  It is important to schedule in extra time for debugging individual use cases. In our classes, example documents that some students brought did not perfectly integrate with the pipeline that we had built. Furthermore, for the ORCD course, some students did not have enough home directory storage to download the LLM, so we had to walk them through additional steps to save the files elsewhere.
- **Tailoring material for multiple audiences is complicated.**
  Our pre-workshop survey revealed that students joined the course for various reasons. Some were trying to develop an LLM-RAG pipeline of their own, but others attended to learn about containers, and others simply wanted more experience using an HPC cluster. While this was in accordance with our goal—to teach various concepts through a single example—this made it difficult to choose which aspects of the course to cover in more detail.
- **Low teacher-to-student ratios are vital.**
  Due to the interactive nature of the course, students learned through trial and error. We found it essential that enough instructors were present to provide one-on-one assistance to students.
- **Students are interested in agent effectiveness.**
  Many students sought a practical application of this pipeline to fit their needs as researchers. As a result, they needed to know how accurate the agent is at summarizing information to be reliable for their research. In the future, students would benefit from a more formal teaching section on evaluating the LLM-RAG agent.

4.2 Provisioning Resources

In addition to the lessons learned from teaching, the teams took away a number of lessons associated with provisioning the systems for these workshops:

- **The LLSC team noticed a peculiar usage pattern with vLLM.**
  A number of nodes were running jobs that had very spiky usage, idle for long periods with intermittent workloads. While this behavior is common for LLMs, it is particularly problematic with multi-node models and made worse when users do not shutdown the vLLM server. Various solutions were applied including a time limit on the vLLM server and configuring the vLLM server so that research groups could share a single vLLM server instance.
- **The pipeline should be configured to use optimal storage spaces.**
  The ORCD team's setup involved users downloading models to their home directory, which take up at least 15GB of space. Because not all users have that much available space, we encouraged them to choose alternate locations, such as temporary scratch storage.

- **A vector database may be edited if a different LLM is used.** ORCD's configuration stored its default vector database in a global, read-only location on our computing cluster. This became problematic when users attempted to experiment with alternative LLMs, as the pipeline tries to write to the vector store during execution if the LLM is changed. To resolve this, we modified the pipeline to automatically copy the vector database to the user's home directory, ensuring write permissions and preserving flexibility in model experimentation.

5 SUMMARY AND NEXT STEPS

The authors developed LLM-RAG tutorials and used them to successfully train researchers who were new to using supercomputing systems. These learning modules should be reusable at other HPC centers with slight modifications.

The ORCD team in particular hopes to make our approach more generalizable by utilizing alternative software that helps make the building of LLM agents more streamlined, especially on a shared computing cluster. These tools, such as Ollama, vLLM, or Morphik, are greatly valuable for scaling up workflows and simplifying the development process. Furthermore, ORCD hopes to continue collaboration with MIT undergraduate students, who have been integral to this project so far. We will continue to offer this course to our community, constantly iterating and improving upon our setup to make this a more valuable resource. Our course materials, including our "How-To" guide as well as our GitHub repository, are available public access online [8] [7].

The LLSC team is in the process of converting the individual modules and hands-on exercises to a short "Building LLM pipelines on HPC systems" course that should be easily modifiable for other centers. The goal is to provide learners with a skeleton they can modify and tweak to explore LLM pipelines along with a guide to understand the general approach. As a result, the LLSC approach doesn't use any special tools, beyond vLLM, and should be transferrable to other HPC systems. In addition, the authors would like to create a teaching guide that can be used by other members of the education and training community.

ACKNOWLEDGEMENTS

DISTRIBUTION STATEMENT A. Approved for public release. Distribution is unlimited. This material is based upon work supported by the Department of the Air Force under Air Force Contract No. FA8702-15-D-0001 or FA8702-25-D-B002. Any opinions, findings, conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Department of the Air Force. JM would like to acknowledge the MIT SuperCloud and LLSC teams for assistance with hosting and deploying the workshop materials and facilitating the training. SC and LM would like to acknowledge the ORCD team, especially Jonathan Murray for helping out during the workshop.

REFERENCES

[1] Harrison Chase. 2022. LangChain. https://github.com/langchain-ai/langchain Release date: 2022-10-17.

[2] Jeremy S. Hudson, Joel Welling, Alan Chalker, Trey Dockendorf, Jeff Ohrstrom, Andreas Dilger, and Boyd Wilson. 2018. Open OnDemand: A web-based client portal for HPC centers. *Journal of Open Source Software* 3, 25 (2018), 622. https://doi.org/10.21105/joss.00622

[3] Owain Kenway. 2023. *Running a Large Language Model on our HPC systems.* Retrieved March 12, 2025 from https://www.youtube.com/watch?v=taUixaGpSbs

[4] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient Memory Management for Large Language Model Serving with PagedAttention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles.*

[5] Marco Magliulo. 2025. *Inference of Hugging Face's Pre-Trained LLMs on HPC Systems.* Retrieved March 12, 2025 from https://www.youtube.com/watch?v=0YAyy3ACYek

[6] Mistral AI. 2024. Ministral-8B-Instruct-2410. https://huggingface.co/mistralai/Ministral-8B-Instruct-2410. Fine-tuned instruction-following language model with 8B parameters, released under the Mistral AI Research License.

[7] MIT Office of Research Computing and Data. 2025. *orcd-rag.* Retrieved September 9, 2025 from https://github.com/mit-orcd/orcd-rag

[8] MIT Office of Research Computing and Data. 2025. *Running Your Own Retrieval-Augmented Generation (RAG) Model.* Retrieved September 9, 2025 from https://orcd-docs.mit.edu/recipes/rag/

[9] NYU HPC. [n.d.]. *LLM of HPC.* Retrieved March 12, 2025 from https://sites.google.com/nyu.edu/nyu-hpc/training-support/general-hpc-topics/ai-at-hpc-tips/llm-on-hpc

[10] QC University eResearch. [n.d.]. *Large Language Models/Natural Language Processing on HPC System CPU or GPU modules.* Retrieved March 12, 2025 from https://eresearch.cqu.org.au/high-performance-computing/hpc-user-guides-and-faqs/running-llms-on-the-hpc-system/

[11] Singularity Developers. 2021. Singularity. https://doi.org/10.5281/zenodo.1310023