Delivering Large Language Model Platforms with HPC

Laura Huber
Research Technologies
Indiana University
Bloomington, IN, USA
lamhuber@iu.edu

Scott Michael
Research Technologies
Indiana University
Bloomington, IN, USA
scamicha@iu.edu

Jefferson Davis
Research Technologies
Indiana University
Bloomington, IN, USA
majdavis@iu.edu

Abhinav Thota
Research Technologies
Indiana University
Bloomington, IN, USA
athota@iu.edu

***Abstract*—In 2023, we saw a huge rise in the capability and popularity of large language models (LLMs). OpenAI released ChatGPT 3.0 to the public in November 2022, and since then, there have been many closed and open-source LLMs that have been released. It has been reported that training ChatGPT 3.0 took more than 10,000 GPU cards, making training a foundational LLM out of reach for many research teams and HPC centers, but there are many ways to use a pre-trained LLM using GPUs at scales available at an HPC center. For example, fine tuning a pre-trained model with site or application-specific data or augmenting the model with Retrieval Augmented Generation (RAG) to add specific knowledge to a pre-trained model. The availability of open-source LLMs has opened the field for individual researchers and service providers to do their own custom training and run their own chatbots. In this paper, we will describe how we deployed and evaluated open source LLMs on Quartz and Big Red 200, a Cray EX supercomputer, and provisioned access to these deployments to a select group of HPC users.**

***Index Terms*—HPC, LLM, Llama, Mistral, Cray**

I. Introduction

Throughout 2023 and into 2024, there has been a steady increase in the number of LLMs made available for research and enterprise. These have come in a number of forms, a variety of open-source options, options that are at no cost but with restrictions on use, or for fee LLMs as-a-service. One of the first releases was OpenAI's ChatGPT 3.0, released to the public in November 2022, and since then there has been an explosion of available models. In addition to the major LLM as-a-service offerings like ChatGPT, Bard, Copilot, Grok, Claude, Titan, etc., there are thousands of open-source variants that can be found on Hugging Face [1]. It has been reported that training ChatGPT 3.0 took more than 10,000 GPU cards [2], and training a foundational model from scratch is no small undertaking, requiring a huge amount of input data, GPU resources, and model adjustment. Even if re-creating foundational models or creating new foundational models is out of reach for many research teams, there are many ways to use a pre-trained LLM using GPUs at scales available at an HPC center. The availability of open-source LLMs has opened the field for individual researchers and service providers to do their own custom training and run their own chatbots. In this paper, we will describe how we deployed and evaluated open-source LLMs, including Llama 2 [3], an open-source LLM from Meta, and Mistral 7B [4] and Mixtral 8x7B [5], open-source LLMs from Mistral AI [6], on Quartz and Big Red 200 and provisioned access to these deployments to a select group of HPC users.

This paper is organized as follows: in section II we cover some of the possible use cases for open-source LLMs, our evaluation strategy, and the hardware that we employed to test performance. Section III details how we deployed and tested the various LLMs and the parameters of the various models. In section IV we present the performance measurements and our analysis. Finally, in sections V and VI we discuss different platform options for deployment in a production setting and future work.

II. Model Selection and Design Considerations

In investigating candidate models for use in our implementation, we restricted our options to open source models that could be readily used by our academic research community with minimal barriers. Llama 2, Mistral, and Mixtral were selected for their broad usage support, relatively good reported performance, and interest from our research community.

Llama 2's collection of pretrained models and weights at 7 billion (7B), 13 billion (13B), and 70 billion (70B) parameter sizes are provided by social media company Meta, and is currently free to use for commercial and research use. It has a native context window size of 4096 tokens, in which it can receive input and produce output of up to a combined 4096 tokens. Llama-2-Chat, a fine-tuned chat model optimized for dialogue, is also available with the same parameter sizes. Larger parameter sizes express a higher number of weights available to fit increasingly complex patterns of data, at the risk of over-fitting to their training data and failing to generalize responses to questions outside their training. This selection of parameter sizes gives us options to optimize for while searching for which model best fits an HPC use case.

Mistral AI provides a 7B parameter large language model that competes with the similarly sized Llama 2 7B and is provided under an Apache 2.0 license. Mixtral 8x7B, a sparse mixture of experts model (SMoE) LLM, acts as Mistral AI's 70B model offering. In contrast to Llama 2, Mistral utilizes sliding window attention to extend its defined context window to 8192 without experiencing accuracy degradation and can

theoretically expand further. Mistral AI also provides fine-tuned chat models with Mistral 7B Instruct and Mixtral 8x7B Instruct.

A. Hardware and Platform Selection

With these models selected, we needed an LLM inference platform with which to provide model-driven output based on user input. To narrow the field of options supporting these models, we evaluated the systems we intended to use and our anticipated usage scenarios.

We initially considered two general types of interaction with these models, one using the model for analysis of a large corpus of data via batch scheduling, the other using the model in a more interactive fashion to explore data sets and interact via conversation. Casual, minimally intensive interactions with lightweight models that could return reading-speed results on less powerful but much more readily available CPUs would ideally be provided with as few hurdles as possible. Performance-limited usage on login nodes or unlimited usage on interactive CPU nodes would satisfy this need and provide access to the basic end-functionality of these LLMs. More complicated usage, such as large-scale document summarization and the utilization of larger, more intensive models would require GPU-accelerated hardware available via scheduler-managed nodes. Ideally, our inference platform of choice would enable us to serve both of these use cases.

Indiana University operates two batch-scheduled systems: Big Red 200 and Quartz. Big Red 200, IU's flagship HPC system, is an HPE Cray EX with 640 CPU nodes and 64 GPU-accelerated nodes equipped with four NVIDIA A100 40GB GPUs. Quartz is IU's high-memory system comprised of 90 CPU nodes and 24 GPU-accelerated nodes containing four NVIDIA Tesla V100 32 GB GPUs.

Research Desktop (RED), an OpenStack-provisioned and ThinLinc-managed cluster of virtual machines serving a VNC-based graphical desktop front-end for Quartz, is also provided to the IU research community alongside our scheduled systems. On this system, users run their workflows in a desktop environment directly on the multi-user node they are assigned upon starting a new session. These nodes are comprised of 48-core Intel Haswell CPUs and 362 GB of memory, though usage policies limit users to 5 processes and 100GB of total memory utilization.

Several popular solutions dedicated to deploying Llama 2 and other LLMs currently exist. vLLM [7], a Python library developed for LLM inference and server-based deployment, is a popular option for use cases that plan to heavily utilize GPU hardware and handle many simultaneous requests to one instance with load balancing. NVIDIA Triton [8] also provides several server-provisioning solutions for deploying LLMs to users through various containers and Python framework backends, again heavily utilizing GPUs. As we imagined this service and its usage existing completely within a high performance computing context, with a single instance of our selected models per user on an ad-hoc basis, we were not interested in deployment platforms that were focused on server-distributed inference and access. Additionally, these platforms require GPU hardware to function, which would leave the CPU-only workflows behind.

Llama.cpp [9], another popular alternative, is a C and C++ framework designed to leverage the Llama 2 model to perform model inference on a variety of end-user devices with minimal requirements, low overhead, easy customizability, and no emphasis on server distribution. Python bindings for Llama.cpp via the `llama-cpp-python` package [10] are also available, providing a convenient avenue for our users working with the models in Python. Llama.cpp would enable us to quickly deploy in an HPC context with potential for CPU-only inferencing as well as distributing memory usage across multiple GPUs and nodes, which would be beneficial as we test larger model sizes. Therefore, out of our options, we selected Llama.cpp to begin our testing with.

Model	Unquantized (GiB)	Q4_K_M (GiB)	Model Layers
Llama-2-7B	12.55	4.08	33
Llama-2-13B	24.25	7.87	81
Llama-2-70B	137.96	41.42	41
Mixtral-8x7B	86.99	24.62	33
Mistral-7B	13.49	4.07	33

TABLE I
UNQUANTIZED AND QUANTIZED MODEL SIZES.

III. IMPLEMENTATION

Constructing our deployment began with obtaining our Llama 2, Mistral, and Mixtral models. The Llama 2 and Llama-2-Chat 7B, 13B, and 70B parameter model weight files were obtained via Meta's downloading utility and placed in a directory on Slate-Project, our high-capacity centralized Lustre filesystem. Mistral 7B and Mixtral 8x7B were both obtained from Mistral AI's Hugging Face repository. The Llama.cpp inference platform was also installed in several locations in the directory, with GPU-accelerated installations compiled against cuBLAS to properly utilize the NVIDIA cards. OpenBLAS and other libraries are also supported for CPU and AMD GPU use cases. The Llama.cpp platform provides several utilities such as the `convert` Python utility, which can convert .pth model weight files into a single compatible .gguf format, which stores models for inference with GGML [11].

The resulting files can be run as-is with Llama.cpp, however their file sizes can range from unwieldy to impossible to run on many cards, such as Llama-2-70B's 137.96 GB .gguf file. Quantization, or reducing the precision of the model's parameters from floating point to lower bit representations, can reduce the disk size and memory usage of these models at the cost of some accuracy. This is a necessary evil, as 140 GB GPU cards are not yet widely accessible. Llama.cpp provides a useful `quantize` utility to carry out this process on the model's .gguf file as well. This utility can quantize models along a series of 2-6 bit quantization methods with quantization mixes such as their k-quant method (denoted with a K in their naming scheme) and adjustments to result size in the range of small (S), medium (M), and large (L) [12]. Table I

Hardware	Sample rate (tokens/sec)	Prompt eval rate (tokens/sec)	Evaluation rate (tokens/sec)
Llama-2-7B Q4 Model			
1x V100	2241.93 ± 103.91	352.77 ± 13.68	100.95 ± 1.09
2x V100s	2288.59 ± 153.49	338.04 ± 13.56	96.87 ± 2.54
4x V100s	2209.57 ± 85.92	313.08 ± 1.13	92.21 ± 0.98
1x A100	6955.15 ± 167.64	373.38 ± 1.01	105.44 ± 0.93
2x A100s	6779.31 ± 131.60	370.82 ± 1.14	109.77 ± 0.77
4x A100s	6905.26 ± 242.50	375.77 ± 3.16	103.88 ± 0.64
1x GH200	47 326.23 ± 764.50	643.54 ± 7.78	179.19 ± 2.55
1x Intel Haswell CPU (48 cores)	1617.01 ± 56.38	13.38 ± 6.33	6.76 ± 0.99
1x AMD ROME CPU (128 Cores)	30 118.21 ± 565.20	58.45 ± 3.13	7.95 ± 0.12
Llama-2-13B Q4 Model			
1x V100	2312.14 ± 62.60	204.07 ± 8.02	61.96 ± 0.40
2x V100s	2355.67 ± 81.80	203.45 ± 6.50	61.21 ± 0.21
4x V100s	2286.82 ± 91.54	194.98 ± 6.00	59.02 ± 0.85
1x A100	6953.07 ± 285.14	220.70 ± 0.50	68.18 ± 0.51
2x A100s	6829.83 ± 85.49	243.92 ± 4.17	73.62 ± 0.33
4x A100s	6840.84 ± 310.00	223.03 ± 4.58	68.21 ± 0.66
1x GH200	47 144.73±1081.21	413.40 ± 1.68	119.25 ± 2.02
1x Intel Haswell CPU 48 cores	1373.64 ± 173.39	12.23 ± 5.91	1.81 ± 0.71
1x AMD ROME CPU (128 Cores)	29 563.71 ± 394.85	33.00 ± 0.42	4.25 ± 0.03
Llama-2-70B Q4 Model			
1x V100	N/A	N/A	N/A
2x V100s	2323.69 ± 92.24	37.13 ± 1.53	15.57 ± 0.09
4x V100s	2289.59 ± 91.42	37.58 ± 1.03	15.26 ± 0.16
1x A100	6862.11 ± 226.28	41.76 ± 0.18	19.37 ± 0.09
2x A100s	6793.16 ± 169.32	46.65 ± 0.05	20.63 ± 0.08
4x A100s	6839.68 ± 169.32	48.35 ± 1.92	19.41 ± 0.24
1x GH200	45 498.37±1512.34	90.34 ± 0.38	36.76 ± 0.22
1x Intel Haswell CPU 48 cores	1516.74 ± 110.02	3.07 ± 0.82	0.73 ± 0.23
1x AMD ROME CPU (128 Cores)	28 386.10 ± 917.95	6.63 ± 0.37	0.93 ± 0.01

TABLE II
RESULTS OBTAINED FROM MULTIPLE SIZES OF LLAMA 2 MODELS QUANTIZED USING THE Q_4_K_M METHOD. ALL VALUES ARE THE MEAN AND STANDARD DEVIATIONS OF TOKENS PER SECOND FOR TEN RUNS.

describes the file sizes of both the unquantized 16-bit floating point Llama2 models and those quantized with the Q4_K_M quantization method, which provides 4-bit k-quantization that produces a medium-sized file with a balanced quality trade-off. Q4_K_M and Q5_K_M, an additionally recommended 5-bit quantization method that results in a slightly larger file size and slightly better quality, were added to our implementation for use. To test a competitor model, Mistral 7B and Mixtral 8x7B Q4_K_M quantized models were also obtained.

Ultimately, our directory dedicated to holding these source weights, quantized models, and utilities grew to a size of 2.7 TB. Once the models and Llama.cpp inference software are available on the system, testing basic usage of the LLMs with command-line-driven prompts is simple. Llama.cpp's `main` function provides options for tuning performance, such as identifying the number of threads to spawn, layers to divert to GPUs rather than run directly on CPUs, which model to use, and the intended size limit of response desired. For instance, the line in Listing 1 would ask the quantized Llama-2-70B model to "Write 100 words about Abe Lincoln" while spawning 40 threads and offloading all of the model's 81 layers to the visible NVIDIA A100s on one of Big Red 200's compute nodes, with results permitted to continue printing until reaching the maximum of Llama2's native 4096 context window size or the natural conclusion of its answer, whichever comes first. In the results described below for GPU based runs we made sure all of the model layers were copied to the GPU. In some instances the device memory was not large enough to support the entire model. Llama.cpp will distribute model layers as evenly as possible to all available GPUs. As can be seen in the following section, this has little impact on performance but does increase the total available device memory and expands the size of the model that can be deployed.

Listing 1. Sample LLM Prompt

```
main -m ../llama-2-70b/ggml-model-Q4_K_M.
    gguf -n -2 -t 40 -ngl 81 -p "Write
    100 words about Abe Lincoln"
```

IV. MODEL PERFORMANCE DATA AND ANALYSIS

In this section, we present performance data from a variety of models, quantizations, and hardware platforms, including CPUs, NVIDIA V100s, A100s, and GH200s. It should be noted that in many discussions of LLMs the term performance refers to the accuracy or repeatability of the model for a variety of tasks. A number of benchmarks exist measure model performance in different scenarios (e.g. MMLU, HellaSwag, Winogrande, GSM8k, etc.) For this paper we are focusing solely on the throughput of the model on the given hardware. We provide some analysis of the suitability of different model and hardware combinations for the aforementioned use cases. The goal is to provide an estimate for the hardware that is

Hardware	Sample rate (tokens/sec)	Prompt eval rate (tokens/sec)	Evalutation rate (tokens/sec)
Mistral-7B Q4 Model			
1x A100	6737.70 ± 100.07	339.38 ± 1.27	99.96 ± 0.67
2x A100s	6792.47 ± 73.02	364.90 ± 17.61	107.07 ± 1.47
4x A100s	6828.18 ± 185.50	343.95 ± 4.60	100.11 ± 1.04
1x GH200	$43\,724.65 \pm 822.44$	568.13 ± 3.38	174.03 ± 1.58
Mixtral 8x7B Q4 Model			
1x A100	6909.97 ± 200.93	121.70 ± 0.55	55.89 ± 0.42
2x A100s	6972.63 ± 294.17	133.76 ± 0.57	56.85 ± 0.61
4x A100s	6884.74 ± 211.05	148.73 ± 2.76	56.22 ± 0.82
1x GH200	$42\,184.38 \pm 611.81$	207.72 ± 1.20	85.66 ± 0.84

TABLE III
RESULTS OBTAINED FROM THE MISTRAL AND MIXTRAL MODELS QUANTIZED USING THE Q_4_K_M METHOD. ALL VALUES ARE THE MEAN AND STANDARD DEVIATIONS OF TOKENS PER SECOND FOR TEN RUNS.

needed to run a given LLM of a particular quantization in an HPC environment.

Table II shows the performance of Llama 2 7B, 13B, and 70B models quantized using the Q_4_K_M method on a variety of hardware. The Q4 quantized models were the lowest precision models we tested across the 7B, 13B, and 70B sizes. As the model size increases from 7B to 70B, the runs are more demanding, and the number of tokens generated per second decreases consistently across all hardware. The Llama.cpp 'main' function will provide a number of timings if logging is enabled. In table II we present the main three measures provided by Llama.cpp. The sample time is a measure of the amount of time spent in selecting the next likely token, prompt evaluation time is a measure of time spent evaluating the input file or prompt input before generating new text, and evaluation time is a measure of the time it took to generate the output. Each of these timings are accompanied by the number of runs or tokens generated and so can be converted into a rate of tokens per second, which is what we present throughout the rest of this paper. Also provided by Llama.cpp, but not reported here, is the model load time which is simply the amount of time taken to copy the specified number of layers from the model into the GPU device(s) memory.

All three measures, sample rate, prompt evaluation rate, and evaluation rate exhibit similar patterns. We will focus on the evaluation rate as it tends to take the longest and so has the smallest rates and smallest standard deviation. As noted above, as the model size increases, the evaluation rate decreases. This decrease is not exactly, but nearly, linear with the number of model parameters. For all of the metrics, the run-to-run deviation is relatively small (5% or less) with the standard deviation for the evaluation rate being around 1% for the GPU runs. As noted above, distributing the model among multiple cards does not provide a performance benefit, in many cases a small performance decrease ($<$10%) is seen, but this will often be an acceptable penalty to have access to a larger pool of device memory and be able to host larger models. For instance, the 70B model does not fit on a single V100, but fits on the rest of the platforms. Successive generations of NVIDIA GPUs perform incrementally better than their predecessor with the V100 to A100 jump giving a 5% to 10% performance boost. Not surprisingly, the NVIDIA GH200 GraceHopper SuperChip performs the best among the hardware we tested, doubling the performance of the V100s for the 70B model. However, we noticed some anomalous timing for the GH200 load times with load times being 100x the load time on a single A100 card. At the time of writing the GH200 node had only just been made available for testing, so we suspect there is some misconfiguration of the memory subsystem that is causing long device load times. The CPU-only runs are slower but still usable for interactive use with a 7B model, but with a 13B model it is borderline and with 70B model, the performance is not suited for interactive usage.

We ran the Llama 2 7B, 13B, and 70B models quantized using the Q_5_K_M method on the same hardware as the Q4 models. The largest performance differences in token generation between Q_4_K_M and Q_5_K_M were in the 5% to 10% range, so decisions between the two quantization methods should not be based on performance. The qualitative performance difference between the two models remains to be investigated, but in the interest of keeping things concise, we are not including the Q5 performance results here.

A. Model and Quantization Alternatives

According to Mistral AI, Mistral 7B outperforms Llama 2 13B on all benchmarks [13] and Mixtral 8x7B model matches or outperforms Llama 2 70B, as well as GPT3.5, on most benchmarks [14]. These evaluations focus on the accuracy rather than the throughput of the model, however both Mistral 7B and Mixtral 8x7B outperform Llama models in throughput as well. In addition, both models have a smaller memory footprint than comparable Llama models and have a larger 32K context window (as compared to Llama's 4K window). In table III we show the performance of Mistral 7B and Mixtral 8x7B model quantized using the Q_4_K_M method. While the evaluation rate of Mistral 7B is comparable to Llama 7B, it beats Llama 13B model handily. The Mixtral 8x7B model outperforms the Llama 2 70B model by at least 2 times on A100 and GH200 hardware. The relative standard deviation of the throughput is in line with the Llama 2 results for both Mistral and Mixtral.

We additionally compared the throughput of the unquantized Llama, Mistral, and Mixtral models. While the memory requirements are $\approx$3x larger for the unquantized model com-

Hardware	Sample rate (tokens/sec)	Prompt eval rate (tokens/sec)	Evalutation rate (tokens/sec)
Llama-2-7B Unquantized Model			
1x A100	6824.21 ± 264.88	779.75 ± 6.60	67.12 ± 0.33
2x A100s	6877.40 ± 285.39	780.04 ± 2.44	67.13 ± 0.22
4x A100s	6732.86 ± 705.88	738.01 ± 18.71	58.42 ± 1.92
Llama-2-13B Unquantized Model			
1x A100	6848.63 ± 174.41	468.88 ± 1.36	40.11 ± 0.15
2x A100s	6924.93 ± 237.62	467.11 ± 1.39	40.08 ± 0.12
4x A100s	6939.41 ± 271.30	432.97 ± 100.83	38.01 ± 0.86
Llama-2-70B Unquantized Model			
1x A100	N/A	N/A	N/A
2x A100s	N/A	N/A	N/A
4x A100s	6708.44 ± 149.34	52.57 ± 4.52	8.00 ± 0.02
Mistral-7B Unquantized Model			
1x A100	6753.46 ± 127.31	703.63 ± 3.47	63.16 ± 0.22
2x A100s	6755.95 ± 85.77	706.50 ± 3.91	63.22 ± 0.21
4x A100s	6732.86 ± 135.55	687.74 ± 13.35	58.42 ± 0.81
Mixtral 8x7B Unquantized Model			
1x A100	N/A	N/A	N/A
2x A100s	N/A	N/A	N/A
4x A100s	7006.53 ± 305.25	40.96 ± 0.30	31.39 ± 0.15

TABLE IV
RESULTS OBTAINED USING UNQUANTIZED MODELS. ALL VALUES ARE THE MEAN AND STANDARD DEVIATIONS OF TOKENS PER SECOND FOR TEN RUNS.

pared to the Q4 quantization, it is still possible to run even the largest Llama 2 70B model using multiple A100 cards. We only tested the unquantized models on the A100 cards as the V100 cards did not have quite enough memory to fit the 70B model and the GH200 node only contains a single GPU. Table IV shows that for the prompt evaluation rates for the unquantized model are much higher than the 4Q quantized model, but the evaluation rate, the rate at which the output is generated, tends to be slower by as much as a factor of 2. Although the unquantized models do offer improved model output accuracy, due to the fact that the bulk of the time for most applications is in generating output, one should consider a quantized model if throughput is an important metric.

V. LLM PLATFORM FOR A RESEARCH COMMUNITY

Many universities and other research institutions are beginning to offer Azure AI services and OpenAI based custom ChatGPT services for their user bases. These solutions can become expensive and introduce data security concerns. There are many advantages to setting up on-premise LLM access, including potentially large cost savings, better control over data, and more control over customization. With an objective of providing an accessible experience to both experts and novices alike, we intended to design a platform on our HPC systems that is simple to use and readily permits researcher investigation into LLM utilization.

To grant users access to the Llama.cpp implementation described in the previous sections, we provided end-user access to our deployment directory using Indiana University's implementation of ColdFront [15]. In IU's ColdFront implementation, RT Projects, we created a 'project' containing a resource allocation for a storage directory on a centrally accessible Lustre filesystem, where we can add users. RT Projects synchronizes membership changes on the site with the LDAP group that is populating the permissions in the Lustre directory. Adding new users to the 'project' is easy, and the request process for being added provides a natural gate to validate and onboard newcomers. To further facilitate usage, Lmod modules containing paths to our Llama.cpp install, all model files, and our custom scripts were placed on our system. Separate modules were created for CPU and GPU workflows, each pointing to the Llama.cpp installation intended for each.

In laying the groundwork for our deployment to end users, we created two primary scripts wrapping around Llama.cpp's `main` program, one that would provide a straightforward and lightweight chat function called `tellme`, and the other delivering summaries of users' text files at a higher computational cost called `summarize` [1]. These would serve as simple entry points for users to use our models in some of its more standard use cases before moving on to using our repository of models directly in their own work.

We ultimately opted to use Llama-2-Chat as our default model for these scripts. Though Mistral does provide impressive performance, we found that more of our users were interested in using Llama 2 specifically. This popularity is likely due to Meta's size and reputation as a heavily invested competitor in this space, which makes the opportunity to test it very appealing. Since our scripts were specifically targeting conversational interactions, we opted to use the variant of Llama that was fine-tuned for chat, Llama-2-Chat, for these end-user-facing implementations. Whereas standard Llama 2 provides text completion of provided prompts based on the model's training, Llama-2-Chat provides more question-and-answer style results that are more akin to a 'chatbot' style of output. The chosen parameter sizes we used were influenced by the weight of activity ascribed to the script and intended target system resources.

Research Desktop, our cluster serving as a VNC-based

[1]Our source code can be found on GitHub .

graphical desktop frontend for Quartz, was considered early in development as a low-barrier venue for our platform's users to interact with LLMs in small runs. As mentioned earlier, Research Desktop's 'good citizen' usage policies limit users to 100GB of memory utilization and up to 5 threads of parallelism. As a system that is intended to accommodate medium-memory low-parallelism workloads interactively, it was well suited to running the smaller models, such as the 7B and 13B Llama2 models. The login nodes for both Big Red 200 and Quartz were similarly identified as a venue for quick access, and are also restricted in computational capacity, to a greater degree than Research Desktop is. Anything that required larger models or heavy processing were to be directed to batch job submissions on our GPU partition. Therefore, our scripts and recommended usage guidelines were crafted with these limits in mind.

A. Chat Functionality

The `tellme` command feeds the argument's prompt to the quantized Llama-2-Chat-7B model, using Llama.cpp's `main` for inference. It prints results per-token until either its 'thought' is complete and control is passed back to the user, or its context window is filled and the session closes. Control can be taken back by the user to interject at any time with the `Control+C` keyboard shortcut. Llama-2-Chat-7B, being optimized for dialogue and selected for its light weight, made it ideal for `tellme` as a quick-use utility when working on a login node or Research Desktop when limited to a few processes. Listing 2 shows how this utility is used.

Listing 2. Sample tellme Usage

```
tellme Why is the sky blue?
```

While Research Desktop's nodes lack GPU-accelerated hardware and are multi-user by nature, an average token generation performance of 6.76 tokens per second on the Llama-2-Chat 7B Q_4_K_M model was still achievable on the system as permitted by a single unprivileged user on one node with 5 threads, which was considered an acceptable reading speed for American English readers. Though the 13B and 70B Q_4_K_M models both technically fit within the memory requirements, we found the output to be too slow to be usable in a conversational setting. Similar performance with the 7B model was observed on the Quartz and Big Red 200 login nodes. As loading the 13B and 70B models were not realistic possibilities in a production environment on the login nodes, we ultimately decided to only use the 7B model for `tellme`. When using `tellme` on a GPU node on either Big Red 200 or Quartz, the script runs at full capacity and offloads all layers to the GPUs, providing much-improved performance as described in section IV. In future iterations, we would likely let GPU-accelerated runs use the 13B or 70B parameter versions instead.

Overall, we found the response time suitable for this use-case and the responses to our questions more often correct than not, though the Llama-2-Chat 7B model still appears to be susceptible to 'hallucinations', sometimes providing responses that are nonsensical, incorrect, or irrelevant.

B. Document Summarization Functionality

The `summarize` script ingests either a single text file or a directory of text files for inference with `main`. To provide the model with structure for the prompt and expected output, a new file with a template wrapped around an ordered list of concatenated files to be summarized is created. This file is then used as the full prompt for `main`, which will use the Llama-2-Chat-13B model. Since this can be a heavy operation, `summarize` requires using a GPU node to execute. The 13B parameter model was preferred over the 7B as it provides more reliable responses, and over the 70B as it provides a faster response and comfortably fits within a single V100 or A100 GPU, which is what we anticipate a typical user will be requesting for this task.

Listing 3. Sample input template for multiple document summarization

```
### Instruction:
Summarize the following multiple texts:
### Input:
### FILE 1
    ...
### FILE 2
    ...
### Response:
```

Another consideration for document summarization tasks is that most documents or directories of documents will contain more tokens than Llama2's models can handle within its native context window of 4096 tokens. Llama.cpp provides an implementation of RoPE context scaling [16], which can be used to enable the model to process more tokens at the cost of some stability and reliability. In our testing, we anecdotally found the results to still be fairly reliable when extending the RoPE scale by 2 and expanding to an 8192 token context window. Though `summarize` is structured for fairly interactive, rapid examination of data sets, it can also be utilized in batch submissions for the analysis of large data sets.

However, we still encountered some reservations with the results of summarization with Llama 2. In a task analyzing course feedback responses, we found it struggled to give every individual comment equal weight and focused more heavily on singularly lengthy comments and topics that humans wouldn't likely fixate on. The context window limit is still difficult to contend with for analysing large corpora in a single session as well. In our future work, `summarize` may benefit from using Mistral Instruct or Mixtral Instruct instead of Llama-2-Chat to take advantage of its increased base context window and scaling via sliding window attention, since token limits are the main limitation with Llama 2 in this setting.

VI. Future Work

With the foundation of this platform laid out, we aim to open access to researchers that are incorporating LLMs in their

research, as well as friendly users that are curious about them. The IU HPC user community encompasses a broad range of subject expertise and computing skill levels, and not all users will have prior experience with HPC or LLMs. By providing access to a pre-trained Llama 2 LLM on HPC, users could become familiar with both and could attract more academic diversity to these spaces.

We have already identified 28 distinct research projects using our systems that are investigating LLMs. Six of these have singled out Llama models as a specific model of interest, such as one hoping to leverage Llama 2 to analyze voter preferences in elections. Providing this platform for these users could expand their options for getting started quickly. Additionally, the centralized location and simple accessibility of these models could reduce the number of 'duplicate copies' of these models on our storage systems, potentially saving terabytes of storage capacity. We would like to provide more model varieties centrally through this platform, including updates to our currently-supported LLMs such as the recently released Llama 3 model.

Many of these projects aim to fine-tune a pre-trained model with site or application-specific data or augment the model with Retrieval Augmented Generation (RAG) to add specific knowledge to a pre-trained model. Another one of our next steps is providing a simplified method to enable RAG capability with users' own datasets, and provisioning our own RAG augmented with IU's HPC documentation and support ticket content to provide answers to simple system usage questions as they arise for users and serve as an additional onboarding resource.

The deployment of these LLMs and their accessibility to end users will be experimental at this next stage, and must be denoted as such to newcomers. Thorough user education on the reliability of model results, risk of 'hallucinations', and data privacy and security will need to be created and refined as the field evolves. LLM usage trends are adapting quickly, and we are interested in following where it goes in a method that is open and fair for our community.

Acknowledgment

The authors acknowledge the Indiana University Pervasive Technology Institute for providing supercomputing and storage resources that have contributed to the research results reported within this paper.

References

[1] Hugging face leaderboard. [Online]. Available: https://huggingface.co/open-llm-leaderboard

[2] Chatgpt trains on 10k nvidia gpus. [Online]. Available: https://www.fierceelectronics.com/sensors/chatgpt-runs-10k-nvidia-training-gpus-potential-thousands-more

[3] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, D. Bikel, L. Blecher, C. C. Ferrer, M. Chen, G. Cucurull, D. Esiobu, J. Fernandes, J. Fu, W. Fu, B. Fuller, C. Gao, V. Goswami, N. Goyal, A. Hartshorn, S. Hosseini, R. Hou, H. Inan, M. Kardas, V. Kerkez, M. Khabsa, I. Kloumann, A. Korenev, P. S. Koura, M.-A. Lachaux, T. Lavril, J. Lee, D. Liskovich, Y. Lu, Y. Mao, X. Martinet, T. Mihaylov, P. Mishra, I. Molybog, Y. Nie, A. Poulton, J. Reizenstein, R. Rungta, K. Saladi, A. Schelten, R. Silva, E. M. Smith, R. Subramanian, X. E. Tan, B. Tang, R. Taylor, A. Williams, J. X. Kuan, P. Xu, Z. Yan, I. Zarov, Y. Zhang, A. Fan, M. Kambadur, S. Narang, A. Rodriguez, R. Stojnic, S. Edunov, and T. Scialom, "Llama 2: Open foundation and fine-tuned chat models," 2023. [Online]. Available: https://arxiv.org/abs/2307.09288

[4] A. Q. Jiang, A. Sablayrolles, A. Mensch, C. Bamford, D. S. Chaplot, D. de las Casas, F. Bressand, G. Lengyel, G. Lample, L. Saulnier, L. R. Lavaud, M.-A. Lachaux, P. Stock, T. L. Scao, T. Lavril, T. Wang, T. Lacroix, and W. E. Sayed, "Mistral 7b," 2023. [Online]. Available: https://arxiv.org/abs/2310.06825

[5] A. Q. Jiang, A. Sablayrolles, A. Roux, A. Mensch, B. Savary, C. Bamford, D. S. Chaplot, D. de las Casas, E. B. Hanna, F. Bressand, G. Lengyel, G. Bour, G. Lample, L. R. Lavaud, L. Saulnier, M.-A. Lachaux, P. Stock, S. Subramanian, S. Yang, S. Antoniak, T. L. Scao, T. Gervet, T. Lavril, T. Wang, T. Lacroix, and W. E. Sayed, "Mixtral of experts," 2024. [Online]. Available: https://arxiv.org/abs/2401.04088

[6] Mistral ai. [Online]. Available: https://mistral.ai/

[7] W. Kwon, Z. Li, S. Zhuang, Y. Sheng, L. Zheng, C. H. Yu, J. E. Gonzalez, H. Zhang, and I. Stoica, "Efficient memory management for large language model serving with pagedattention," in *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023.

[8] Nvidia triton inference server. [Online]. Available: https://developer.nvidia.com/triton-inference-server

[9] G. Gerganov. (2024) LLM Inference in C/C++. [Online]. Available: https://github.com/ggerganov/llama.cpp

[10] Python bindings for llama.cpp. [Online]. Available: https://github.com/abetlen/llama-cpp-python

[11] G. Gerganov. (2024) Ggml tensor library for machine learning. [Online]. Available: https://github.com/ggerganov/ggml

[12] Kawrakow. (2023) k-quants. [Online]. Available: https://github.com/ggerganov/llama.cpp/pull/1684

[13] Mistral 7b. [Online]. Available: https://mistral.ai/news/announcing-mistral-7b/

[14] Mixtral 8x7b. [Online]. Available: https://mistral.ai/news/mixtral-of-experts/

[15] A. Bruno and D. Sajdak, "Coldfront: Resource allocation management system," in *Practice and Experience in Advanced Research Computing*, ser. PEARC '21. New York, NY, USA: Association for Computing Machinery, 2021. [Online]. Available: https://doi.org/10.1145/3437359.3465585

[16] G. Gerganov. (2023) Extending context size via rope scaling. [Online]. Available: https://github.com/ggerganov/llama.cpp/discussions/1965