Automated Dynamic AI Inference Scaling on HPC-Infrastructure: Integrating Kubernetes, Slurm and vLLM

Tim Trappen
Ruhr University Bochum
Bochum, Germany
tim.trappen@ruhr-uni-bochum.de

Robert Keßler
University of Cologne
Cologne, Germany
kessler@uni-koeln.de

Roland Pabel
University of Cologne
Cologne, Germany
pabel@uni-koeln.de

Viktor Achter
University of Cologne
Cologne, Germany
achter@uni-koeln.de

Stefan Wesner
University of Cologne
Cologne, Germany
wesner@uni-koeln.de

Abstract

Due to rising demands for Artificial Inteligence (AI) inference, especially in higher education, novel solutions utilising existing infrastructure are emerging. The utilisation of High-Performance Computing (HPC) has become a prevalent approach for the implementation of such solutions. However, the classical operating model of HPC does not adapt well to the requirements of synchronous, user-facing dynamic AI application workloads. In this paper, we propose our solution that serves LLMs by integrating vLLM, Slurm and Kubernetes on the supercomputer *RAMSES*. The initial benchmark indicates that the proposed architecture scales efficiently for 100, 500 and 1000 concurrent requests, incurring only an overhead of approximately 500 ms in terms of end-to-end latency.

Keywords

Inference, High-Performance Computing, Large Language Model

ACM Reference Format:
Tim Trappen, Robert Keßler, Roland Pabel, Viktor Achter, and Stefan Wesner. 2025. Automated Dynamic AI Inference Scaling on HPC-Infrastructure: Integrating Kubernetes, Slurm and vLLM. In *Proceedings of Next-Gen Middleware for MLOps in Distributed Systems (MIND '25).* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

Artificial Intelligence (AI), and in particular Large Language Models (LLMs), have become an integral factor in numerous areas of our lives. For instance, the use of LLMs in an university setting involves a wide range of user groups, including students, researchers, and even the administration. Students benefit from personalized education pathways, are provided with dynamic learning content and feedback, and thus can self-regulate their individual progress [2, 14]. The use of chatbots within e-learning platforms, such as MoodleBot, provides students with a personal tutor who is available 24/7, thereby reducing the administrative workload for teachers [15, 18]. Scientists are already making use of the increasingly powerful LLMs for their research work in various domains such as drug discovery or partial differential equations, where the requests range from simple literature search and summary to concept clarification, data analysis and methodology guidance [1]. However, the potential of LLMs goes way beyond this, meaning that in the future they could serve as a kind of AI research assistant, as envisaged by the AuroraGPT [1] project [5]. Even the administrative offices are increasingly using AI tools, for example to support the enrollment process or to provide better assistance with scholarship applications [12].

For higher education institutions, compliance and data governance pose the greatest challenges when adopting to AI. This creates the need for independent and sovereign AI infrastructure which provides easy to use and privacy preservering access to models. Universities that operate their own data centers are predestined to offer such services themselves, as they tend to possess the necessary computing power and can also guarantee the data privacy aspects. The computing systems operated by higher education institutions are often High-Performance Computing (HPC) systems, also known as supercomputers, which are used by scientists from a wide range of scientific disciplines to solve highly complex research problems and perform simulations, but tend to be underutilized [11]. However, the operating model of HPC systems is based on static allocation of computing resources, which is suitable for training LLMs but is not an inherent match for processing dynamic LLM inference requests [16]. To overcome this discrepancy, one can either *(i)* implement dynamic resource allocation in HPC systems from scratch, which requires at least adjustments at the level of the resource manager and programming models [20], or *(ii)* implement a scalable web API, that handles the dynamic management of HPC jobs and forwards the inference requests accordingly. Due to the manifold modifications required for approach *(i)*, this comprehensive implementation is very sophisticated, whereas approach *(ii)* offers an implementation using micro-service architecture, which can be deployed in a scalable manner using e.g. Kubernetes.

The remainder of this paper is structured as follows, in section 2 we present and discuss challenges and other approaches to solve the problem of serving dynamic AI inference workloads on HPC infrastructure. In section 3 we describe our own approach, evaluate its performance in section 4 and discuss eventual threats to validity

[1] https://www.anl.gov/cels/auroragpt-foundation-models-for-science

MIND '25, Nashville, TN, USA

ACM ISBN 978-x-xxxx-xxxx-x/YYYY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

in section 5. We conclude our paper with an outlook on the future work in section 6.

2 Related Work

As HPC and AI converge, this is reflected, for example, in the massive investments of the European Union (EU) in so-called *AI Factories*[2], which are intended to drive forward the development of trustworthy generative AI models in Europe. But also on a smaller scale, existing HPC infrastructures at higher education institutions are being used to run AI projects such as *Open Source-KI.nrw* [3]. This poses a certain contradiction, as AI environments are usually based on dynamic cloud-native solutions, whereas HPC systems are operated using batch processing and static resource allocation. Specifically, HPC systems are not inherently designed to support interactive applications, which is why Lopez et al. propose to rely on a sovereign dual-stack architecture for future systems, in order to combine the best of both worlds [16]. HPC systems have always been performance-sensitive, yet with the advent of AI workloads, it is becoming critical for cloud computing as well. In order to ensure sustainable performance of HPC systems, especially in the exascale era, accelerators such as Graphics Processing Units (GPUs) are essential, which in turn are also required for AI. The approach proposed by Hoefler et al., aims on the advatages of the HPC-Cloud convergence, by providing a simple and fast access to accelerators with their Acceleration as a Service (XaaS) model [10]. In turn, AI itself can be used in the HPC domain to efficiently generate high-performant parallel code, which subsequently improves not only conventional HPC applications but also the models themselves [6].

This kind of convergence does also pose certain challenges, including not only performance but also sustainability and scalability factors. Resch et al. therefore suggests that, in addition to the traditional TOP500 list of supercomputers, future evaluations should also focus specifically on AI workloads, emphasizing the time to deliver a solution or the complexity of finding a solution rather than pure Floating Point Operations Per Second (FLOPS) [19]. In order to overcome the challenges associated with realizing AI-coupled HPC workflows, the development of new middleware solutions is required to accommodate various execution motifs such as dynamic orchestration, multistage pipeline, and adaptive training [4].

To provide AI inference on a system, a variety of services must be operated and linked with each other, e.g., a user registry, service registry, and service identifier components, but also the support of scheduling and scaling functions. Guran et al. present a middleware solution that considers this and serves as a LLM gateway, but is lacking support for HPC environments [9]. The *FLEXI* project at FernUniversiät in Hagen implements on-premise hosting of open-source LLMs in an higher-education environment but also lacks support for HPC systems [22]. This shortcoming has been addressed by Luiz et al., who have developed a solution that implements a scalable microservice-based master-slave architecture that serves heterogeneous LLM models on HPC infrastructure leveraging the Slurm workload manager [17]. A similar implementation is offered by *Chat AI*, which operates its chat front-end as a web service in the Cloud , and then forwards the corresponding LLM requests via an API gateway to the respective model on the HPC infrastructure for being processed [7].

Despite all these efforts, resource and workflow managers still show a lack of support for the openness, transparency, and reusability of such workflows on HPC systems, meaning that the inherent dynamicity in such systems needs to be further fostered [8]. Furthermore, although attempts have been made to implement AI inference on HPC systems, there is a lack of systematic performance studies [3].

3 Architecture

Our approach separates various microservices into two layers, as illustrated in Figure 1. The first layer consists of *(i)* a Kubernetes-based web API that forwards inference requests from users to the corresponding models in the backend and manages the lifecycle of resources / models. The second layer consists of *(ii)* the LLM models themselves which are encapsulated into Slurm jobs.

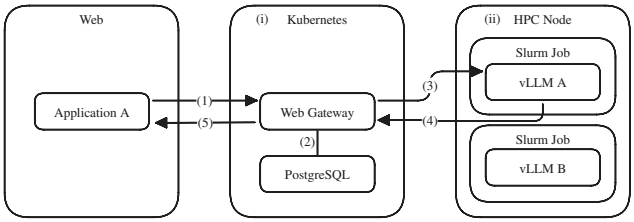

Figure 1: Visualization of the flow between layers for an incoming inference request.

To orchestrate the containers required for the services of our web API, we use Kubernetes[4] to automatically schedule, manage and scale the deployments. For the underlying container runtime we use Apptainer[5], since it is specifically designed for HPC environments. As resource manager for the allocation and deployment of the HPC nodes running the LLM endpoints, we use Slurm[6], as this is the de facto standard in this domain.

Central to almost all of the microservices is a single relational PostgreSQL[7] database running in Kubernetes. It holds two domains (): *(a)* authentication and *(b)* Slurm job management, as visualized in Figure 2. Authentication consists of a simple 1:N relation of `identity_tenants` and their API keys `identity_tenant_authentications`, which are stored in an encrypted format. Slurm job management consists of multiple 1:N relations, which have proven to provide consistency in an actual production scenario.

3.1 Inference Components

3.1.1 vLLM. Offering heterogeneous LLM serving and inference in a high-throughput scenario requires a fully scalable inference engine, such as the open-source software vLLM[8]. One of its major advantages is the introduction of *PagedAttention*, which splits the Key-Value Cache (KVC) into blocks and assigns them to logical

[2] https://digital-strategy.ec.europa.eu/en/policies/ai-factories
[3] https://www.oski.nrw
[4] https://kubernetes.io
[5] https://apptainer.org
[6] https://slurm.schedmd.com
[7] https://www.postgresql.org
[8] https://github.com/vllm-project/vllm

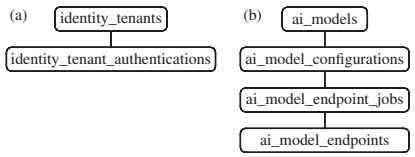

Figure 2: Visualization of the central database schema.

pages allowing for dynamic scaling of allocated GPU memory [13]. This mapping is handled via block tables, managed by a central KV cache manager. In doing so, blocks may be reused in single- and shared across multiple requests, leading to significant performance gains in scenarios where single prompts consist of long inputs, or a large amount of requests need to be served simultaneously. This also allows for models with weight sizes exceeding the memory capacity of a single GPU to run efficiently across multiple GPUs. vLLM implements an OpenAI-standard compatible FastAPI frontend for serving requests. If the number of requests received exceeds the system's concurrent throughput capabilities, a first-come, first-served scheduling policy is employed for all incoming requests.

3.1.2 Web Gateway. The flow of a single request from a user's application through the Web Gateway to the desired vLLM endpoint running inside a Slurm job is visualized in Figure 1. The system's primary entry point for client applications is the Web Gateway, which is responsible for authentication and the subsequent forwarding of incoming requests. Its API endpoints are OpenAI-standard compatible, making them agnostic to the consuming client application. Request properties are strongly typed and validated, adding an additional layer of robustness. Authentication is handled via long-lived bearer-tokens, stored in an encrypted format in the database and authenticated on each request. To minimise database load, a distributed memory cache stores recently authenticated API keys for a limited period. Subsequent to the authentication and validation process (1), the Web Gateway conducts a search for available endpoints corresponding to the requested LLM within the `ai_model_endpoints` table (2). It then forwards the request - along with all request parameters - to a matching node (3). The response flows from vLLM back to the Web Gateway (4) and then to the requesting application (5). If no matching vLLM endpoint ready for inference is found, custom HTTP status codes are returned.

3.2 Management Components

Figure 3 visualizes the flow between microservices and layers described in the following subsections, which are ordered in the way the components interact with each other, starting from the Job Worker.

3.2.1 Job Worker. The Job Worker functions as an intermediary between our Kubernetes-based microservices and the Slurm-managed HPC infrastructure. It is implemented as a long-running background process, responsible for scheduling new vLLM instances via Slurm and removing expired jobs. Each run, it compares the entries in the `ai_model_endpoint_jobs` table with the configurations specified in the `ai_model_configurations` table, for example number of requested model instances. If no matching

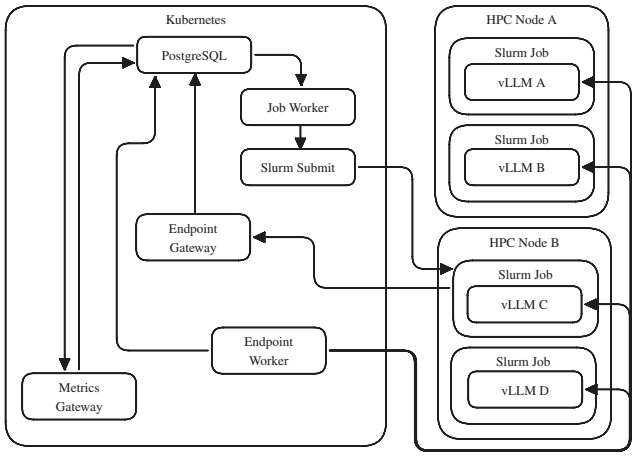

Figure 3: Visualization of the flow between layer for Slurm and vLLM management.

endpoint job is found, configuration parameters are passed to the Slurm Submit, which requests ressourcess for a job with the corresponding model. On success, an `ai_model_endpoint_jobs` entry is created, containing information such as the Slurm job id, timestamps for job submission, job registration and readiness of the contained vLLM server. To prevent simultaneous submission and subsequent startup of jobs, which could lead to inconsistent port mappings on compute-nodes, model configurations are iterated synchronously. The Job Worker waits for a specified timespan after a successful submit before proceeding to the next configuration.

3.2.2 Slurm Submit. Slurm Submit is the service handling communication between the Job Worker and Slurm. It accepts a string from an SSH connection, which is forwarded to a bash script that actually deploys the job. A dedicated `munged` process inside the container provides authentication for the Slurm components. The comma-delimited parameters from the string are parsed and used to select a model-specific `.slurm` file from a folder mounted into the container, that in turn is used to run Slurm's `sbatch`. The file holds `#SBATCH` directives specifying hardware requirements of the requested node(s), among others. In addition, it sets environment variables for the vLLM container and configures a curl request to the Endpoint Gateway. For multi-node multi-GPU setups, required for very large models, it also contains logic for setting up head and worker nodes.

3.2.3 Endpoint Gateway. Upon successful submission of a Slurm job, a curl `POST` is initiated from the Slurm script to the API of the Endpoint Gateway. It contains the endpoint job id, Slurm job id, node id, model version, capabilities and bearer token for the vLLM server that is being spawned. Authentication with the Endpoint Gateway is facilitated by a bearer token that is passed to the script. The underlying business logic proceeds to check if the internal job id holds a corresponding job in `ai_model_endpoint_jobs` that has no endpoint attached. If that is the case, it compares the ports of all already existing endpoints on the supplied node and assigns a port $p = argmax(port) + 1$, which is passed back to the Slurm script in

the curl response to be used by the vLLM server. Finally, a new entry is created in `ai_model_endpoints`, holding information such as node, port, model version, bearer token and a nullable datetime field indicating readiness of the endpoint, which at that time is null.

3.2.4 Endpoint Worker. The Endpoint Worker is another long-running background service, responsible for managing endpoint health status. Each run, it iterates over all entries in the `ai_model_endpoint_jobs` table and sends a `GET` request to the `/health` endpoint of each job, which is provided by the vLLM server. Endpoint jobs not yet marked as ready but receive a response status-code `200` are updated accordingly by setting their corresponding fields to the current datetime. Once marked ready, the endpoint will be considered for forwarding requests by the Web Gateway. In cases where no response is returned, the Endpoint Worker differentiates two cases: (1) jobs that have been canceled or expired and (2) jobs that are still starting up. Since loading model weights can require a substantial amount of time, a configurable 30-minute timeout is currently implemented before a job is deemed cancelled or expired. Moving forward, this could be dynamically configured with an additional estimated load time on a per-model basis via the `ai_model_configurations` table. For cancelled or expired jobs, the Endpoint Worker removes the corresponding entries in `ai_model_endpoints` and `ai_model_endpoint_jobs`.

3.2.5 Metrics Gateway. The Metrics Gateway serves API endpoints for Prometheus, a time-series database, and Grafana, an open-source observability platform. The Prometheus endpoint returns a response with the required format for Prometheus' HTTP service discovery, which allows for dynamic (de-)registration of targets for metrics scraping. By accessing the `ai_model_endpoints` table, it converts necessary information like node id, port and bearer token of all running vLLM instances to the specified Prometheus template and allows for additional meta fields to be added, such as job id and Slurm job id. While metrics of our Kubernetes based microservices can be conveniently scraped with Prometheus Operator, obtaining metrics from the vLLM instances requires this workaround, as they are not part of the Kubernetes cluster and may change network adresses over time. The Grafana endpoints, in contrast, accept various `POST` request parameters, which are used to control the amount and type of running model instances.

3.3 Observability & Automated Dynamic Scaling

In addition to the fundamental inference and management components, an observability stack is utilized, comprising Prometheus, Grafana and Grafana Loki to facilitate visualisation, reporting and logging. This allows for real-time monitoring of critical metrics, such as the vLLM instance load or concurrent Web Gateway requests, hence providing insight into how frequently certain models are accessed. In combination with live logging, it provides the capability to efficiently identify and resolve issues almost instantaneously.

This stack also comprises the central part of the mechanism to dynamically scale up or down vLLM model instances inside Slurm jobs, based on GPU load. Utilizing Grafanas alert rules with its contact point webhook notifications, custom JSON payloads are automatically sent to the metrics gateway. Here, the business logic adjusts the number of model instances specified in the `ai_model_job_configurations` table. This triggers the Job Worker to automatically start the newly requested number of instances on its next invocation, currently every 15 seconds. Hence, the system scales by actual hardware load, since vLLM reports metrics like KVC utilization, queue times and token throughput. In our first tests, we utilized the vLLM queue time metric, where a queue time above 5 seconds over 30 sustained seconds triggered instantiation of an additional model instance (see subsection 4.2 for further explanation). Compared to scaling by number of requests, which in the context of LLMs is an arbitrary metric on its own given the lack of uniformity of the requests [21], for example input token count or hyperparameters like maximum generated tokens, this setup allows for maximizing GPU load and thus token throughput. In our first tests, we utilized the vLLM queue time metric, where a queue time above 5 seconds over 30 sustained seconds triggers instantiation of an additional model instance (c.f. subsection 4.2).

4 Performance Evaluation

4.1 Methodology

In this section, we provide preliminary performance results for our system via the serve-benchmark in vLLM using the *BurstGPT_without_fails_2* dataset [21].

We benchmark two system configurations: (1) *GPU-S*, which utilizes two NVIDIA L40S GPUs with an AMD EPYC 9654 CPU, and (2) *GPU-L*, which uses one NVIDIA H100 GPU and an AMD EPYC 9454 CPU. All configurations utilize 96GB RAM. Networking is based on Mellanox Infiniband HDR100 (100GBit/s), with two ports for redundancy on Kubernetes nodes and one port on compute nodes. The topology consists of 8 leaf-switches linked to 2 spine-switches via 5 HDR (200GBit/s) links each, resulting in a bisection bandwidth of 8000 Gb/s. The maximum blocking factor is 2.3:1.

For our baseline model, we use Mistral Small 3.2 24B Instruct 2506[9] with default parameters. The load is simulated and averaged over 50 runs for three scenarios: 100, 500 and 1000 concurrent requests. All runs are performed from a dedicated interactive node inside the HPC cluster, using the same vLLM container image running inside the Slurm jobs used for inference serving (v0.10.2). The seed is set to 0, to ensure that every benchmark run uses the same samples from the dataset. The vLLM node is not restarted or modified between runs; however, between each run, the KVC is fully cleared. All requests are handled using streaming.

In addition to the baseline benchmark, the system is also benchmarked including the Web Gateway. Before starting a full run, the vLLM serve-benchmark sends one initial request to the specified target, which triggers the caching of the authentication in the Web Gateway, causing it to only perform one database trip for each incoming request to look up the vLLM node endpoint.

4.2 Results

Results for the GPU-S and GPU-L configurations are shown in Table 1. The vLLM benchmark suite defines time to first token (TTFT) as a timespan between sending the request and receiving the corresponding first token. Similarly, end to end latency (E2EL) is a

[9] Mistral Small 3.2 24B Instruct 2506 on HuggingFace

Table 1: GPU-S and GPU-L configuration performance benchmarks for concurrent vLLM and Web Gateway requests.

Configuration	GPU-S						GPU-L					
Benchmark	vLLM Node			Web Gateway			vLLM Node			Web Gateway		
Concurrent Requests	100	500	1000	100	500	1000	100	500	1000	100	500	1000
E2EL Median (ms)	2307.96	7381.58	22385.04	2186.39	6331.33	22963.34	1195.80	3029.86	6190.10	1149.93	3076.11	9419.23
E2EL Std (ms)	5215.32	9316.33	20958.25	5151.45	9148.52	19999.29	2940.16	4984.30	8206.37	2981.23	4994.50	8465.61
Requests Total Duration (s)	30.01	48.02	84.76	30.03	46.95	79.75	17.20	25.94	34.81	17.31	26.28	37.25
Requests Total Input Tokens	77561.00	381456.00	768960.00	77561.00	381456.00	768960.00	77561.00	381456.00	768960.00	77561.00	381456.00	768960.00
Requests Total Output Tokens	7048.98	49763.68	141407.84	6976.34	49567.64	141313.34	6978.78	50126.12	142778.08	7027.58	49955.58	142664.64
TPOT Median (ms)	65.52	90.61	102.13	63.33	79.20	94.97	33.41	67.21	86.74	32.49	42.20	49.05
TPOT Std (ms)	6.36	13.07	18.00	5.52	6.92	10.50	2.34	12.19	29.23	2.08	3.16	5.58
TTFT Median (ms)	404.38	1849.79	8328.36	334.79	2412.93	6825.14	225.58	993.91	2190.97	207.35	1133.02	3109.98
TTFT Std (ms)	37.59	1477.46	12460.81	146.29	1950.86	13191.81	14.34	87.98	189.22	62.97	841.35	3642.25
Throughput Requests (req/s)	3.33	10.54	11.83	3.33	10.77	12.56	5.82	19.52	28.87	5.79	19.31	26.95
Throughput Token Output (tok/s)	234.91	1048.13	1672.62	232.51	1066.95	1775.28	405.79	1956.22	4121.70	406.30	1928.61	3844.97
Throughput Token Total (tok/s)	2820.34	9088.70	10768.98	2818.61	9282.65	11435.67	4916.07	16849.58	26322.19	4893.63	16664.20	24569.65

timespan between sending the request and receiving the corresponding last token. Therefore time per output token (TPOT) is defined as:

$$\texttt{tpot} = \frac{\texttt{e2el} - \texttt{ttft}}{\texttt{output\_len} - 1} \tag{1}$$

Unintuitively, for GPU-S, total request duration and TPOT favors the Web Gateway setup, albeit with slightly increased E2EL in 100 and 1000 concurrent request scenarios. The 500 concurrent request scenario stands out particularly for TTFT, where sending requests to vLLM directly is noticeably faster by about 500ms (23.34%). GPU-L, in contrast, behaves more as expected, displaying slightly higher total request durations across all Web Gateway scenarios, with a higher E2EL and TTFT from 500 concurrent requests upward. Curiously, TPOT is significantly lower in these cases (37.21% / 43.45%). The reasons for this will need to be investigated in a more detailed system analysis in the future. Inspecting the vLLM logs, the GPU-S configuration starts queuing requests from 255 concurrent requests onward, while the GPU-L configuration does not appear to queue requests in any of our scenarios. While this could be an indicator for queueing impacting general performance, especially when done across multiple GPUs, it also leads to the assumption that once queueing has started, the slight latency introduced by the Web Gateway could buffer the impact of load put on the vLLM api server, 'masking' the queuing process. When compared to GPU-L (where no queueing occurs), we observe higher E2EL and TTFT as expected. However, at 500 and 1000 concurrent requests, MD and Std for TTFT deviates massively for the Web Gateway when compared to direct vLLM node access, with a nearly 1 second (41.95%) slower median TTFT.

This suggests that our current implementation of routing requests via the Web Gateway starts to bottleneck in scenarios where enough GPU compute is present to handle the incoming load.

5 Discussion

The presented architecture is still considered a work in progress, and this paper does not claim to represent best practices. Its purpose is to provide a starting point for other infrastructure projects, highlighting where the chosen implementation has succeeded, where it faced problems, and where it currently falls short.

From early friendly user tests with small groups that use various applications to access our Web Gateway, such as KI:Connect[10] or Open WebUI[11], we are confident that the foundation of our architecture - scaling LLM serving for synchronous applications in an HPC environment - is sound. Management of vLLM instances in Slurm jobs has run successfully in this small-scale production setting for multiple months, across a variety of tested models, covering single-node single-GPU, single-node multi-GPU, and even multi-node multi-GPU configurations.

Our benchmarks show that when scaling beyond a 1000 concurrent request scenario, improvements must be made to our benchmark setup and/or the Web Gateway. The increase in TTFT degrades end-user experience, signaling a longer wait time between the user sending a prompt and the model starting to return a response. However, TPOT remains consistent, which translates to smooth streaming and therefore 'typing'-animation for chatbot applications. We identify several key areas that require further investigation to solve the arising bottleneck:

Networking: since we run the vLLM benchmark suite from within a compute-node of the HPC system, high concurrency scenarios might be impacted by networking components, favoring a direct request from compute-node to compute-node. Alternatively, improper request handling in the Web Gateway may contribute.

Caching: as every request performs a database lookup for a matching model endpoint, implementing a caching mechanism could reduce database load and speed up request handling.

Scaling: horizontal or vertical scaling of either the Web Gateway or PostgreSQL via Kubernetes could improve performance.

Despite these necessary improvements, institutional provisioning of LLMs on sovereign infrastructure provides clear data privacy benefits, especially when handling sensitive data. However, preventing LLM-based attack vectors like prompt injections requires trained experts and domain specific knowledge, which must be seen as additional cost for said privacy benefits.

6 Conclusion & Outlook

In this work, we presented our approach to building a production-ready, modular architecture for highly scalable serving of LLM inference in an HPC context. By leveraging HPC-native components like Slurm and Apptainer, managed by custom tailored microservices and integrating them with cloud-native solutions for orchestration and observability like Kubernetes, Prometheus and Grafana, we implemented a robust solution to deploy and scale LLMs using

[10] https://kiconnect.pages.rwth-aachen.de/pages/

[11] https://openwebui.com

the open-source inference engine vLLM. In addition, we shared performance benchmarks for two distinct compute-node hardware configurations under different load scenarios, showcasing the viability of HPC infrastructure for synchronous workloads like web requests from AI chatbot applications. As our architecture represents a work in progress, our next steps will be directed toward a high-troughput production scenario in a higher education setting. Given the scarcity of usage data for this application, we hope to attain a better understanding of how far sovereign infrastructure for AI applications needs to scale. We will also explore the deployment of other generative ai modalities. With growing concerns about the sustainability of AI, ensuring efficient usage of compute resources will take high priority. Thus, further work on leveraging components like Slurm to balance compute during peak usage times of AI inference while allocating the unused ressources to research computations during off-hours should take priority. Furthermore, exploring hardware configurations from vendors other than NVIDIA seems advisable to prevent a vendor lock-in for datacenters.

Acknowledgments

The work presented in this paper is the result of the joint project by Ruhr-University Bochum and University of Cologne: 'Open Source-KI.nrw'. The project is fully funded by the Ministry of Culture and Science of the State of North Rhine-Westphalia.

References

[1] Microsoft Research AI4Science and Microsoft Azure Quantum. 2023. The Impact of Large Language Models on Scientific Discovery: A Preliminary Study Using GPT-4. arXiv:2311.07361 [cs] doi:10.48550/arXiv.2311.07361

[2] Michael E. Bernal. 2024. Revolutionizing eLearning Assessments: The Role of GPT in Crafting Dynamic Content and Feedback. *Journal of Artificial Intelligence and Technology* 4, 3 (May 2024), 188–199. doi:10.37965/jait.2024.0513

[3] Wesley Brewer, Greg Behm, Alan Scheinine, Ben Parsons, Wesley Emeneker, and Robert P. Trevino. 2020. Inference Benchmarking on HPC Systems. In *2020 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, Waltham, MA, USA, 1–9. doi:10.1109/HPEC43674.2020.9286138

[4] Wes Brewer, Ana Gainaru, Frédéric Suter, Feiyi Wang, Murali Emani, and Shantenu Jha. 2025. AI-coupled HPC Workflow Applications, Middleware and Performance. arXiv:2406.14315 [cs] doi:10.48550/arXiv.2406.14315

[5] Franck Cappello. 2024. AuroraGPT: Exploring AI Assistant for Science. In *2024 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, San Francisco, CA, USA, 1–1. doi:10.1109/IPDPS57955.2024.00009

[6] Le Chen, Nesreen K. Ahmed, Akash Dutta, Arijit Bhattacharjee, Sixing Yu, Quazi Ishtiaque Mahmud, Waqwoya Abebe, Hung Phan, Aishwarya Sarkar, Branden Butler, Niranjan Hasabnis, Gal Oren, Vy A. Vo, Juan Pablo Munoz, Theodore L. Willke, Tim Mattson, and Ali Jannesari. 2024. The Landscape and Challenges of HPC Research and LLMs. arXiv:2402.02018 [cs] doi:10.48550/arXiv.2402.02018

[7] Ali Doosthosseini, Jonathan Decker, Hendrik Nolte, and Julian M. Kunkel. 2024. Chat AI: A Seamless Slurm-Native Solution for HPC-Based Services. arXiv:2407.00110 [cs] doi:10.48550/arXiv.2407.00110

[8] Jorge Ejarque, Rosa M. Badia, Loïc Albertin, Giovanni Aloisio, Enrico Baglione, Yolanda Becerra, Stefan Boschert, Julian R. Berlin, Alessandro D'Anca, Donatello Elia, François Exertier, Sandro Fiore, José Flich, Arnau Folch, Steven J. Gibbons, Nikolay Koldunov, Francesc Lordan, Stefano Lorito, Finn Løvholt, Jorge Macías, Fabrizio Marozzo, Alberto Michelini, Marisol Monterrubio-Velasco, Marta Pienkowska, Josep de la Puente, Anna Queralt, Enrique S. Quintana-Ortí, Juan E. Rodríguez, Fabrizio Romano, Riccardo Rossi, Jedrzej Rybicki, Miroslaw Kupczyk, Jacopo Selva, Domenico Talia, Roberto Tonini, Paolo Trunfio, and Manuela Volpe. 2022. Enabling Dynamic and Intelligent Workflows for HPC, Data Analytics, and AI Convergence. *Future Generation Computer Systems* 134 (Sept. 2022), 414–429. doi:10.1016/j.future.2022.04.014

[9] Narcisa Guran, Florian Knauf, Man Ngo, Stefan Petrescu, and Jan S. Rellermeyer. 2024. Towards a Middleware for Large Language Models. arXiv:2411.14513 [cs] doi:10.48550/arXiv.2411.14513

[10] Torsten Hoefler, Marcin Copik, Pete Beckman, Andrew Jones, Ian Foster, Manish Parashar, Daniel Reed, Matthias Troyer, Thomas Schulthess, Dan Ernst, and Jack Dongarra. 2024. XaaS: Acceleration as a Service to Enable Productive High-Performance Cloud Computing. arXiv:2401.04552 [cs] doi:10.48550/arXiv.2401.04552

[11] Robert Keßler, Simon Volpert, and Stefan Wesner. 2024. Towards Improving Resource Allocation for Multi-Tenant HPC Systems: An Exploratory HPC Cluster Utilization Case Study. In *2024 IEEE International Conference on Cluster Computing Workshops (CLUSTER Workshops)*. IEEE, Kobe, Japan, 66–75. doi:10.1109/CLUSTERWorkshops61563.2024.00019

[12] Suleman Ahmad Khairullah, Sheetal Harris, Hassan Jalil Hadi, Rida Anjum Sandhu, Naveed Ahmad, and Mohammed Ali Alshara. 2025. Implementing Artificial Intelligence in Academic and Administrative Processes through Responsible Strategic Leadership in the Higher Education Institutions. *Frontiers in Education* 10 (Feb. 2025). doi:10.3389/feduc.2025.1548104

[13] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient Memory Management for Large Language Model Serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles* (Koblenz, Germany) *(SOSP '23)*. Association for Computing Machinery, New York, NY, USA, 611–626. doi:10.1145/3600006.3613165

[14] Fangfang Liu, Yiyun Wang, Qiuling Feng, Linkai Zhu, and Guang Li. 2024. Optimizing E-Learning Environments: Leveraging Large Language Models for Personalized Education Pathways. In *2024 5th International Conference on Education, Knowledge and Information Management (ICEKIM 2024)*. Atlantis Press, Chengdu, China, 811–817. doi:10.2991/978-94-6463-502-7_86

[15] Rongxin Liu, Carter Zenke, Charlie Liu, Andrew Holmes, Patrick Thornton, and David J. Malan. 2024. Teaching CS50 with AI: Leveraging Generative Artificial Intelligence in Computer Science Education. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1 (SIGCSE 2024)*. Association for Computing Machinery, New York, NY, USA, 750–756. doi:10.1145/3626252.3630938

[16] Pedro Garcia Lopez, Daniel Barcelona Pons, Marcin Copik, Torsten Hoefler, Eduardo Quiñones, Maciej Malawski, Peter Pietzutch, Alberto Marti, Thomas Ohlson Timoudas, and Aleksander Slominski. 2025. AI Factories: It's Time to Rethink the Cloud-HPC Divide. arXiv:2509.12849 [cs] doi:10.48550/arXiv.2509.12849

[17] Anderson de Lima Luiz, Shubham Vijay Kurlekar, and Munir Georges. 2025. Scalable Engine and the Performance of Different LLM Models in a SLURM Based HPC Architecture. arXiv:2508.17814 [cs] doi:10.48550/arXiv.2508.17814

[18] Alexander Tobias Neumann, Yue Yin, Sulayman Sowe, Stefan Decker, and Matthias Jarke. 2025. An LLM-Driven Chatbot in Higher Education for Databases and Information Systems. *IEEE Transactions on Education* 68, 1 (Feb. 2025), 103–116. doi:10.1109/TE.2024.3467912

[19] Michael M. Resch, Johannes Gebert, and Dennis Hoppe. 2025. The Future of HPC and AI. In *2025 International Conference on Intelligent Control, Computing and Communications (IC3)*. IEEE, Mathura, India, 897–904. doi:10.1109/IC363308.2025.10956516

[20] Ahmad Tarraf, Martin Schreiber, Alberto Cascajo, Jean-Baptiste Besnard, Marc-André Vef, Dominik Huber, Sonja Happ, André Brinkmann, David E. Singh, Hans-Christian Hoppe, Alberto Miranda, Antonio J. Peña, Rui Machado, Marta Garcia Gasulla, Martin Schulz, Paul Carpenter, Simon Pickartz, Tiberiu Rotaru, Sergio Iserte, Victor Lopez, Jorge Ejarque, Heena Sirwani, and Felix Wolf. 2024. Malleability in Modern HPC Systems: Current Experiences, Challenges, and Future Opportunities. *IEEE Transactions on Parallel and Distributed Systems* 35, 9 (2024), 1–14. doi:10.1109/TPDS.2024.3406764

[21] Yuxin Wang, Yuhan Chen, Zeyu Li, Xueze Kang, Yuchu Fang, Yeju Zhou, Yang Zheng, Zhenheng Tang, Xin He, Rui Guo, Xin Wang, Qiang Wang, Amelie Chi Zhou, and Xiaowen Chu. 2025. BurstGPT: A Real-World Workload Dataset to Optimize LLM Serving Systems. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2 (KDD '25)*. ACM, Toronto, ON, Canada. doi:10.1145/3711896.3737413

[22] Torsten Zesch, Michael Hanses, Niels Seidel, Piush Aggarwal, Dirk Veiel, and Claudia De Witt. 2024. Flexible LLM Experimental Infrastructure (Flexi) – Enabling Experimentation and Innovation in Higher Education Through Access to Open LLMs. In *2024 21st International Conference on Information Technology Based Higher Education and Training (ITHET)*. IEEE, Paris, France, 1–8. doi:10.1109/ITHET61869.2024.10837635