JumpLM - LLM Benchmarking and Interactive Performance Monitoring for higher GPU Utilization

Lena Jurkschat
ScaDS.AI Dresden/Leipzig
CIDS, TU Dresden
lena.jurkschat@tu-dresden.de

Anton Rygin
ScaDS.AI Dresden/Leipzig
CIDS, TU Dresden
anton.rygin@tu-dresden.de

Elias Werner
ScaDS.AI Dresden/Leipzig
CIDS, TU Dresden
elias.werner@tu-dresden.de

Abstract

Benchmarking Large Language Model inference allows users to find its fastest configuration (e.g. model, batch size) tied to their application constraints and hardware resources. However, this does not necessarily correlate with a high and sustainable utilization of the available hardware, especially GPUs. Indeed, the hardware monitoring is usually detached from the inference benchmark, hampering the identification of resource- and latency-efficient LLM configurations. In this work, we present JumpLM, a tool that combines BALI – an inference efficiency benchmark with JUmPER – an interactive performance monitoring – integrated into Jupyter Notebooks. JumpLM provides a user-friendly way to deploy different LLM configurations for inference while monitoring the available hardware utilization. To this end, it combines hardware metrics (e.g. GPU utilization) with LLM inference metrics (e.g. tokens-per-second) and provides a novel visualization that gives actionable insights into the hardware usage and LLM performance. This helps users to tune LLM configurations according to the available resources.

1 Introduction

The deployment and use of large language models (LLMs) in HPC have expanded significantly over the past few years, either for direct HPC usage, such as chatHPC Yin et al. (2025) or leveraging HPC infrastructure in various research domains. Inference frameworks like Transformers HuggingfaceWolf et al. (2020) or VLLM Kwon et al. (2023) facilitate the deployment of LLMs for inference tasks. With those developments, the GPU accelerated inference of models has become a crucial research field, bringing the underlying GPU and inference efficiency into focus Wan et al. (2024). In contradictions to this, recent work has shown that a significant proportion of deep learning applications suffer from GPU underutilization (e.g. due to insufficient batch sizes) Gao et al. (2024). This negatively affects inference latency while wasting available hardware capacities. High GPU utilization means more useful computation is performed per unit of energy. This is especially important in HPC and cloud environments, where unused resources can be shared with other applications, or utilization can be increased if the user's own application is able to take advantage of it. Since the majority of applications today run in such shared environments, achieving high utilization is critical for maximizing efficiency, reducing operational costs, and enabling more sustainable AI systems Panwar et al. (2022); Suarez et al. (2025). We suspect, that monitoring the underlying hardware is a hurdle or unseen optimization factor within the LLM and AI community. Furthermore, deploying an inference test environment is time-consuming, as LLM benchmarks need to be set up first, do not include

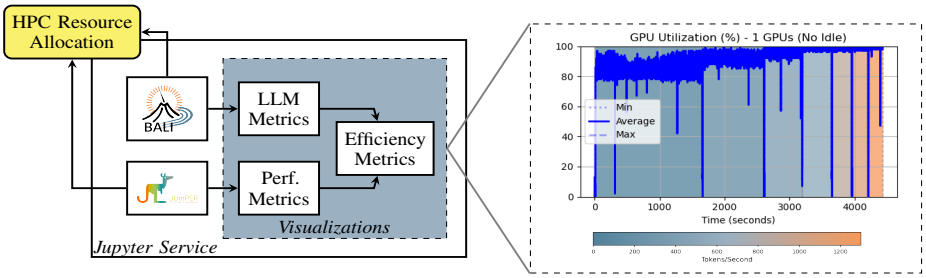


Figure 1: JumpLM - Integration of BALI and JUmPER in Jupyter Service running on HPC infrastructure.

hardware utilization metrics, and hardware monitoring - if available - is often decoupled from LLM execution. Therefore we introduce *JumpLM* that bridges this gap by merging these perspectives into a single tool, seeking to raise greater awareness within the LLM community for the underlying hardware usage.

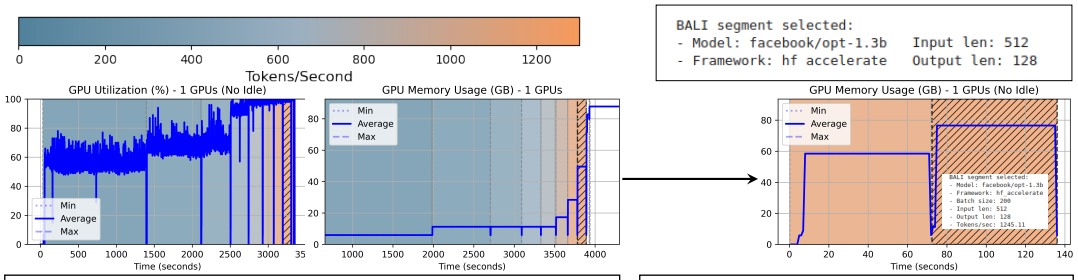
JumpLM provides hardware performance and LLM inference metrics through interactive visualizations within Jupyter Notebooks, enabling users to identify the most effective deployment configurations for their LLM workloads on HPC systems. *JumpLM* is built upon two foundational tools: JUmPER (Jupyter meets Performance) Werner et al. (2024) and BALI (Benchmark for Accelerated Language Model Inference) Jurkschat et al. (2025), integrating them through a dedicated Jupyter extension. This integration provides a new perspective on LLM performance by combining evaluation metrics and visualizations, and offers the following key contributions: ① Interactive inference testing and visualization in Jupyter notebooks with respect to available hardware ② Improved comprehension of hardware utilization driving factors within the LLM inference setup through performance graphs and ③ Explorative configuration tuning for resource-efficient and fast LLM inference deployments. With *JumpLM* we foster a sustainable hardware usage within the LLM community through an improved GPU utilization. To achieve this we provide a user-friendly test-interface that tackles inference speed and GPU usage as tightly coupled optimization factors. Through a conventional batch size scaling scenario, we provide a use-case, showing how the combined performance view can either reveal capacities for accelerating the inference further or indicate the use of a less powerful GPU.

2 JumpLM

JumpLM integrates the BALI benchmark for language model inference with JUmPER, a tool to enable performance measurements and visualization in Jupyter. This integration provides a novel interface for visualizing hardware performance data alongside relevant metrics from LLM deployment. By doing so, it supports identifying the best configuration for a specific LLM and the available HPC hardware. Next, we will introduce the two tools and demonstrate their interplay.

BALI is a benchmarking framework to address the growing need for standardized, comprehensive evaluation of LLM inference efficiency. As LLMs are increasingly used in real-time applications—such as chatbots, translators, and virtual assistants—efficient and fast inference becomes critical. However, comparing different inference frameworks has been challenging due to the lack of standardized benchmarks and the vast configuration space, including hardware, framework parameters, and dataset variations. BALI provides an interface to configure those LLM inference framework parameters comparably, tailored to the user's application constraints.
JUmPER Werner et al. (2024) provides a Jupyter kernel and IPython extension that support coarse-grained performance monitoring and fine-grained analysis of user code in Jupyter. The tool collects system metrics and stores them alongside the code that has been executed. Built-in Jupyter magic commands provide visualizations of the monitored performance data directly within the Jupyter. An description of the implementation details, the combined metrics, available functionalities and the error handling can be found in Appendix A.

Figure 1 illustrates how both tools are integrated into a HPC Jupyter service. BALI and JUmPER are each available as Python packages and can be installed independently within any Python environment. This allows us to collect computational performance metrics using JUmPER and LLM-specific metrics using BALI in parallel. While analyzing the outputs of each tool separately already provides



Coarse-grained Batch Size (BS) Scaling (1-512)
Max BS: 128 → **1206 Token/s**
55% GPU Mem. Usage

Fine-grained BS Scaling (150,200)
Max. BS. 200 → **1245 Token/s**
85% GPU Mem. Usage

Figure 2: JumpLM batch size Scaling Use-Case for OPT 1.3B Zhang et al. (2022). Each coloured segment shows a BALI run using a different batch size. Left: coarse-grained batch size scaling from 1-512 (ascending order) in powers of two (left). Right: fine-grained scaling with a batch size of 150 and 200 based on the maximal feasible batch size of 128 from the coarse grained scaling experiment. The maximal feasible batch size is marked by the shaded area.

valuable insights into the efficiency of the LLM infrastructure, combining both sets of metrics can further facilitate their interpretation and increase their overall informativeness.

In JumpLM, we enhance the usability and integration of the tools: we have extended BALI from a Python module to an installable IPython extension making Jupyter a unified frontend for running benchmarks and visualizing performance data. Moreover, we have adapted JUmPER to integrate with BALI to display LLM-specific metrics alongside the collected performance data. Specifically, JumpLM now plots colourized segments for each benchmarked LLM configuration in a computational performance plot. The colour of the segment corresponds to the Tokens/Second metric as a heatmap allowing to seamlessly interact with the information from both tools.

3 Use Case

Choosing the LLM's size for a specific application comes mainly with considerations concerning the model's answer quality regarding specific downstream tasks. However, with the increasing model size comes a computational and, therefore, latency increase. Having an upper latency limit for inference within an application, the model's size and its configuration become crucial. From a sustainability perspective, monitoring GPU utilization is a vital task in this step. It enables the identification of underutilized computational resources, which can then be optimized in several ways: 1) by adjusting the model's configuration to improve utilization (if beneficial to the application), 2) by sharing the unused resources, or 3) by transitioning to alternative hardware to reduce economic costs.

Within this use case, we focus on the maximum batch size search for the lowest latency for the $1.3B$ model from the OPT family Zhang et al. (2022), using the Huggingface Transformers framework. At the same time, maximizing the GPU memory and its compute utilization supports this search for optimal inference efficiency and reveals further optimization potential. For our experiments, we set up OPT 1.3B with a fixed sample input length of 512 and output length of 128 tokens, a batch size range from $1-512$ in powers of two and executed the model on a single Nvidia H100 GPU (90GB). A single benchmark run was conducted on 1024 samples. Figure 2 displays *JumpLM*'s output after running the benchmark from the Jupyter environment. The left side shows two plots for GPU utilization (left) and GPU memory utilization (middle) with utilization on the y-axis and execution time of the benchmark on the x-axis. The single plots are segmented by different colours that correspond to the different batch size configurations for OPT 1.3B and their respective Tokens/Second metric as outlined in the colour scale at the top of the plot. The generation speed is colour coded from blue (low speed) to orange (high speed). Within the tool, the segments are interactively selectable, such that each configuration can be displayed in detail (see subsection A.1). Scaling the batch size in powers of two is a common practice, although it also exposes residual GPU memory capacity. Following this practice leads to the GPU running out of memory for a batch size of 256 (see GPU memory utilization plot, most right segment). Hence, a batch size of 128 was the last feasible batch size although only 55% of the GPU's memory has been utilized at that point. Therefore, JumpLM reveals that the GPUs memory capacity is not fully used. In a second BALI run (Figure 2 right), we

investigated the impact of more fine-grained batch sizes, specifically 150 and 200. This experiment revealed that utilizing a batch size of 200 results in 85% GPU memory usage, accompanied by an additional gain of approximately 40 Tokens/s. Notably, our results highlight the limitations of relying solely on GPU utilization as a metric, as it can be misleading. For instance, at a batch size of 16, the GPU utilization appears to be nearly 100% with 716 Tokens/Sec and only 11% of the GPU memory being used. This discrepancy underscores the importance of considering multiple metrics when evaluating performance. A reason for the high GPU utilization with low Tokens/Sec is that the utilization reveals the percentage of time one or more kernels have been running on the GPU without reflecting on the actual computations. Thus, GPU utilization can't guarantee efficient LLM inference on a GPU as a stand-alone metric. It's rather useful as a first impression on the inference efficiency, especially when users work with even smaller models than one billion parameters or tiny batch sizes. Hence, the combination of text generation speed (token/s), GPU utilization and GPU memory usage is a useful combination to reveal the maximum inference efficiency. Summarized, JumpLM reveals leftover hardware capacities from any LLM inference configuration in BALI, which can be addressed by the user e.g. through increasing the model size, input length or changing the GPU to less potent hardware like an Nvidia A100 GPU (40GB). This way, JumpLM creates a flexible inference benchmark interface while delivering hardware performance metrics at the same time, increasing the users awareness for HPC resource utilization, conceivably leading to a more sustainable GPU usage.

4 Related Work

Our work combines the benchmarking of LLM inference efficiency with hardware performance monitoring and combined visualization in an interactive environment supporting high-throughput text generation along with maximum hardware usage. Benchmarking LLM inference throughput as well as text generation speed is an increasingly emergent topic. Made efforts include *MLPerf* Reddi et al. (2019) for measuring maximum inference speed across deployment scenarios, *LLM-Inference-Bench* Chitty-Venkata et al. (2024) for evaluating the token throughput under varying hyperparameters and hardware, *LLM-Perf* Waleed Kadous, Kyle Huang, Wendi Ding, Liguang Xie, Avnish Narayasn and Ricky Xu (2023) for benchmarking LLM APIs, and *VLLM* Kwon et al. (2023), a widely used inference engine that provides a benchmark suite for testing VLLM configurations and hardware. Other work, like the work from Martinez Martinez (2025), elaborates on the throughput under different hyperparameters such as GPU count and generation as well as the batch size. *BALI*, a benchmark for accelerated inference Jurkschat et al. (2025), used in this work, allows for comparing various inference frameworks along with models and configurations on a set GPU. However, none of the above-mentioned work include hardware utilization metrics.
In HPC, a variety of tools are typically available for analysing application performance. Node-level monitoring solutions, such as Pika Dietrich et al. (2020) or TACC stats Evans et al. (2014),provide coarse-grained performance insights and performance analysis tools, such as Score-P Knüpfer et al. (2012), HPCtoolkit Adhianto et al. (2010) or framework-specific profilers like the PyTorch profiler offer detailed insights into an application's behaviour. However, these tools are not designed to fit into interactive and application specific workflows like LLM inference benchmarking. JUmPER Werner et al. (2024) provides performance insights in Jupyter notebooks and displays performance graphs in Jupyter. However, it does not reflect on application or LLM specific metrics.

5 Conclusion and Future Work

Investigating hardware usage and LLM specific metrics jointly is crucial for fast responses and resource-efficient LLMs deployment on HPC. To the best of our knowledge, no tool tailored for such an investigation was available within the HPC community. Our novel tool, *JumpLM*, supports HPC practitioners and computational scientists with a user-friendly Jupyter interface for benchmarking LLMs and displaying results with integrated visualisations of hardware usage and LLM text generation metrics for different LLM configurations. This allows users to easily identify suitable parameters for their LLMs and available hardware. For example, they can identify the configuration that achieves the highest number of Tokens/s for the required LLM, given the available GPU memory. We believe that *JumpLM* provides a valuable interface for resource-efficient deployments of LLMs. The source code publicly available via *JumpLM*[1].

[1] https://anonymous.4open.science/r/BALI-5585/README.md

In the future, we will also enable to normalize the text generation speed by the hardware utilization to further point out configurations, that do not keep the GPU sustainably busy. We additionally plan to enrich *JumpLM* with user recommendations for LLM configurations, helping to improve the hardware utilization and text generation speed. Furthermore, metrics like token throughput, time-to-first token or model loading time that are available within BALI will be included.

References

Laksono Adhianto, Sinchan Banerjee, Mike Fagan, Mark Krentel, Gabriel Marin, John Mellor-Crummey, and Nathan R Tallent. 2010. HPCToolkit: Tools for performance analysis of optimized parallel programs. *Concurrency and Computation: Practice and Experience* 22, 6 (2010), 685–701.

Krishna Teja Chitty-Venkata, Siddhisanket Raskar, Bharat Kale, Farah Ferdaus, Aditya Tanikanti, Ken Raffenetti, Valerie Taylor, Murali Emani, and Venkatram Vishwanath. 2024. LLM-Inference-Bench: Inference Benchmarking of Large Language Models on AI Accelerators. arXiv:2411.00136 [cs.LG] `https://arxiv.org/abs/2411.00136`

Robert Dietrich, Frank Winkler, Andreas Knüpfer, and Wolfgang Nagel. 2020. Pika: Center-wide and job-aware cluster monitoring. In *2020 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 424–432.

Todd Evans, William L Barth, James C Browne, Robert L DeLeon, Thomas R Furlani, Steven M Gallo, Matthew D Jones, and Abani K Patra. 2014. Comprehensive resource use monitoring for HPC systems with TACC stats. In *2014 First International Workshop on HPC User Support Tools*. IEEE, 13–21.

Yanjie Gao, Yichen He, Xinze Li, Bo Zhao, Haoxiang Lin, Yoyo Liang, Jing Zhong, Hongyu Zhang, Jingzhou Wang, Yonghua Zeng, Keli Gui, Jie Tong, and Mao Yang. 2024. An Empirical Study on Low GPU Utilization of Deep Learning Jobs. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering* (Lisbon, Portugal) *(ICSE '24)*. Association for Computing Machinery, New York, NY, USA, Article 96, 13 pages. `https://doi.org/10.1145/3597503.3639232`

Lena Jurkschat, Preetam Gattogi, Sahar Vahdati, and Jens Lehmann. 2025. BALI-A Benchmark for Accelerated Language Model Inference. *IEEE Access* (2025).

A. Knüpfer, C. Rössel, D. Mey, S. Biersdorff, K. Diethelm, D. Eschweiler, M. Geimer, M. Gerndt, D. Lorenz, A. Malony, et al. 2012. Score-p: A joint performance measurement run-time infrastructure for periscope, scalasca, tau, and vampir. In *Proc. 5th Int. Workshop on Parallel Tools for HPC*. Springer, 79–91.

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*.

Matias Martinez. 2025. The Impact of Hyperparameters on Large Language Model Inference Performance: An Evaluation of vLLM and HuggingFace Pipelines. In *Proceedings of the 33rd ACM International Conference on the Foundations of Software Engineering* (Clarion Hotel Trondheim, Trondheim, Norway) *(FSE Companion '25)*. Association for Computing Machinery, New York, NY, USA, 1672–1678. `https://doi.org/10.1145/3696630.3728704`

Suraj Singh Panwar, Man Mohan Singh Rauthan, and Varun Barthwal. 2022. A systematic review on effective energy utilization management strategies in cloud data centers. *Journal of Cloud Computing* 11, 1 (2022), 95.

Vijay Janapa Reddi, Christine Cheng, David Kanter, Peter Mattson, Guenther Schmuelling, Carole-Jean Wu, Brian Anderson, Maximilien Breughe, Mark Charlebois, William Chou, Ramesh Chukka, Cody Coleman, Sam Davis, Pan Deng, Greg Diamos, Jared Duke, Dave Fick, J. Scott Gardner, Itay Hubara, Sachin Idgunji, Thomas B. Jablin, Jeff Jiao, Tom St. John, Pankaj Kanwar, David Lee, Jeffery Liao, Anton Lokhmotov, Francisco Massa, Peng Meng, Paulius Micikevicius, Colin

Osborne, Gennady Pekhimenko, Arun Tejusve Raghunath Rajan, Dilip Sequeira, Ashish Sirasao, Fei Sun, Hanlin Tang, Michael Thomson, Frank Wei, Ephrem Wu, Lingjie Xu, Koichi Yamada, Bing Yu, George Yuan, Aaron Zhong, Peizhao Zhang, and Yuchen Zhou. 2019. MLPerf Inference Benchmark. arXiv:1911.02549 [cs.LG]

Estela Suarez, Hendryk Bockelmann, Norbert Eicker, Jan Eitzinger, Salem El Sayed, Thomas Fieseler, Martin Frank, Peter Frech, Pay Giesselmann, Daniel Hackenberg, et al. 2025. Energy-aware operation of HPC systems in Germany. *Frontiers in High Performance Computing* 3 (2025), 1520207.

Waleed Kadous, Kyle Huang, Wendi Ding, Liguang Xie, Avnish Narayasn and Ricky Xu. 2023. *Reproducible Performance Metrics for LLM inference*. Technical Report. Anyscale.

Zhongwei Wan, Xin Wang, Che Liu, Samiul Alam, Yu Zheng, Jiachen Liu, Zhongnan Qu, Shen Yan, Yi Zhu, Quanlu Zhang, Mosharaf Chowdhury, and Mi Zhang. 2024. Efficient Large Language Models: A Survey. *Transactions on Machine Learning Research* (2024). `https://openreview.net/forum?id=bsCCJHbO8A` Survey Certification.

Elias Werner, Anton Rygin, Andreas Gocht-Zech, Sebastian Döbel, and Matthias Lieber. 2024. JUmPER: Performance Data Monitoring, Instrumentation and Visualization for Jupyter Notebooks. In *SC24-W: Workshops of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2003–2011.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. 2020. HuggingFace's Transformers: State-of-the-art Natural Language Processing. arXiv:1910.03771 [cs.CL] `https://arxiv.org/abs/1910.03771`

Junqi Yin, Jesse Hines, Emily Herron, Tirthankar Ghosal, Hong Liu, Suzanne Prentice, Vanessa Lama, and Feiyi Wang. 2025. chatHPC: Empowering HPC users with large language models. *The Journal of Supercomputing* 81, 1 (2025), 194.

Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, Todor Mihaylov, Myle Ott, Sam Shleifer, Kurt Shuster, Daniel Simig, Punit Singh Koura, Anjali Sridhar, Tianlu Wang, and Luke Zettlemoyer. 2022. OPT: Open Pre-trained Transformer Language Models. arXiv:2205.01068 [cs.CL]

A JumpLM Implementation Details

A.1 Visualizaton Interface

The *JumpLM* interface enables visualization and analysis of model execution and system performance within Jupyter environments (see Figure 3). Through the `%perfmonitor_plot` command, the recorded hardware activity is displayed. The hardware metric is selected via a drop down menu, while the inference performance of the underlying *BALI* run is represented by a colorbar indicating text generation speed in Tokens/second. The BALI segments display can be interactively turned on and off via a dedicated control element. To facilitate comparability between different `%bali_run` executions, the colorbar's minimum and maximum values can be adapted to enforce consistent scaling. Periods of GPU idling can be explicitly displayed, thereby revealing erroneous BALI run segments together with their corresponding error outputs (see subsection A.4). For more detailed inspection, individual BALI run segments can be selected to expose their configuration parameters as well as their exact tokens per second and overall latency.

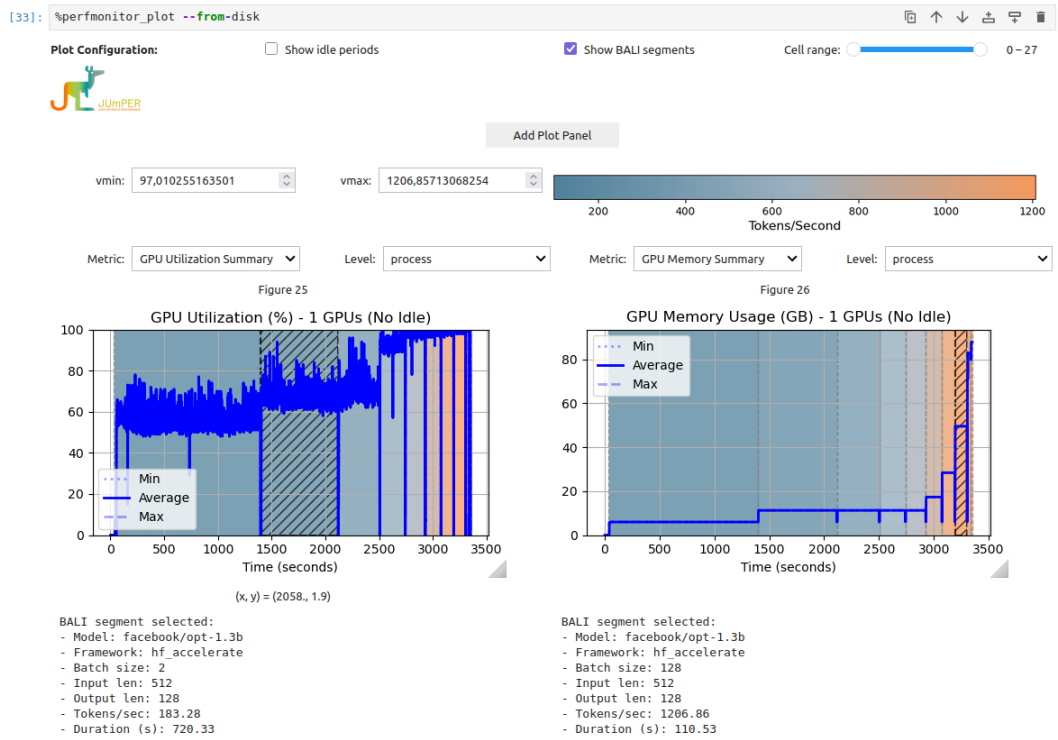


Figure 3: Visualization Interface of JumpLM in a Jupyter Notebook

A.2 Magic commands - Integration of Both Tools

Since BALI has been extended from a Python module to an installable IPython extension, it can be loaded preliminarily into the notebook along with JUmPER. Both extensions provide the following new or adapted magic commands:

`%bali_config`	Set up BALI configurations, e.g., list of models and frameworks.
`%bali_run`	Run BALI across all combinations of configurations.
`%bali_plot`	Plot inference velocity heat maps for the executed BALI runs.
`%perfmonitor_plot`	Extended JUmPER command, includes individual BALI configuration runs as segments into the JUmPer performance visualization.

Along with the benchmark execution, the extension redirects its output into a folder named with the PID of the current IPython session, which enables JUmPER to easily discover BALI data.

A.3 Combined Metrics

In JUmPER, the `%perfmonitor_plot` command has been extended. By default, it draws the time intervals of the executed cells on the background of the plot. Along with the BALI integration, this can be changed to drawing segments, which correspond to individual benchmark runs for each benchmarked BALI configuration. BALI segments on the plot are interactive. Selecting them will display its respective configuration parameters below the plot. Segments themselves are coloured with respect to their average text generation velocity in tokens per second, extracted from the BALI results.

A.4 Error Handling

We also implemented an error handling, e.g. if LLM configurations are not feasible to run on the selected GPU (e.g. out of memory errors). When an individual BALI configuration fails during it execution, its time stamps are collected and shown in the combined visualization. The specific error message can be displayed through the interactive interface, similar to the run configuration parameters.