

# Energy-Efficient vLLM serving with Multi-Objective Bayesian Optimisation

Woon Yee Ng (wyn23)  
Peterhouse

December 31, 2025  
Word Count: 2458

## 1 Introduction

In the field of artificial intelligence, contrary to the common intuition that training consumes the most compute, reports indicate that inference accounts for about 80%-90% of the overall compute demand.[5, 8, 2, 13, 12]. Currently, most large language model inference serving systems[20, 19, 21, 1, 11, 15] focus on optimising throughput, without having an explicit objective of minimising energy consumption. Moreover, most systems use static configuration deployment, meaning that even if the workload changes throughout the day, no reconfiguration will be deployed to minimise energy consumption. A simple example is a deployment that uses the same number of GPUs, even though the LLM inference workload may be lower at night.

To address these problems and fully exploit the optimisation opportunities in the energy domain, this work employs multi-objective Bayesian optimisation (MOBO) to automatically tune the system to deploy an energy-efficient or throughput-maximising configuration, up to user preference. By treating the search space as discrete rather than continuous, we identified two Pareto optimal configurations compared to the baseline, optimising for throughput and energy, respectively. In the throughput-optimised configuration (ID 24), it improves throughput by 4.79% relative to the vLLM default baseline, whereas in the energy-optimised configuration (ID 11), it reduces energy consumption by 9.53%. Furthermore, we investigate the trade-offs between discrete and continuous search spaces in MOBO. Our results show that modelling the search space as discrete rather than continuous significantly improves the surrogate model’s accuracy, yielding lower RMSE

and higher  $R^2$  scores against actual validation benchmarks. In addition, MOBO with a discrete search space has a higher Pareto hypervolume than MOBO with a continuous search space. To investigate why MOBO chooses ID 24 and ID 11 as the Pareto frontier, this project also profiles these runs using NVIDIA’s Nsight System to understand the system-level code execution, revealing the importance of NCCL collective operations in tensor parallelism. All the code is documented in <https://github.com/woonyee28/R244-Mini-Project>.

## 2 Background

### 2.1 vLLM Architecture and Configuration Parameters

Initially, vLLM[10] was proposed to address memory fragmentation that frequently occurs in prior systems, such as Orca[20]. In transformer-based large language models[16], their auto-regressive nature promotes the use of a key-value (KV) cache[14], as generating new tokens depends on all previous tokens in the sequence. As sequence length increases, the size of the KV cache increases, and improper memory management induces memory fragmentation that limits batch size during processing[10], reducing GPU utilisation. To address this problem, vLLM leverages traditional virtual memory and paging techniques from operating systems and applies them to the KV cache, creating a KV cache manager with virtual block tables to reduce memory fragmentation.

vLLM has two deployment methods for prompt processing: batch and online. Batch inference uses the `LLMEngine` class [18] to process input sequences and generate output text. Online inference

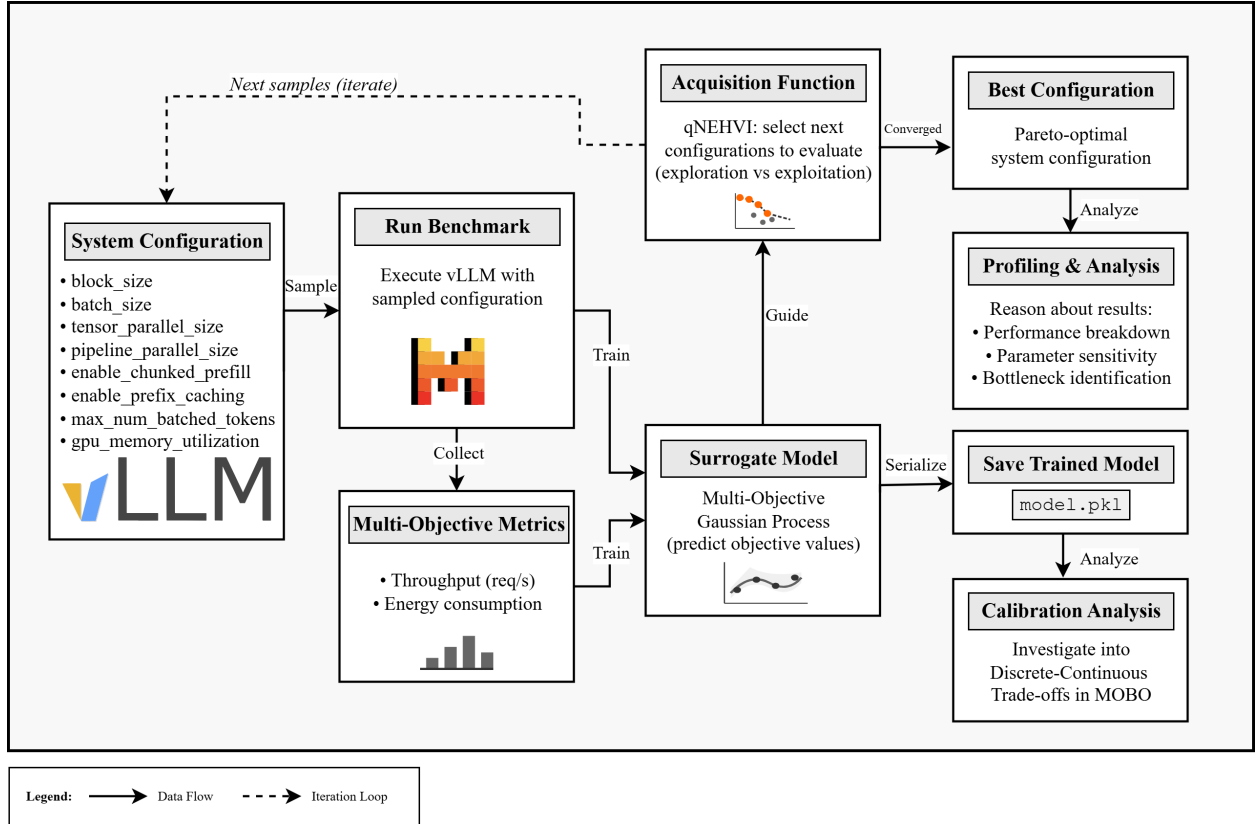


Figure 1: Full system design for this project.

uses AsyncLLM [17], which leverages the `asyncio` package to create a background loop that handles incoming requests. In our project, we focus on batch inference for reproducibility, as it is deterministic and repeatable, reducing variability in network latency and request arrival patterns that affect online inference serving. In the batch inference `LLMEngine` class, several parameters will impact the inference throughput (token/second) and energy consumption (Joule/token), the details are shown in Table 1.

Table 1: vLLM Configuration Search Space

Parameter	Range
<code>block_size</code>	[32, 128]
<code>max_num_seqs</code>	[64, 256]
<code>max_num_batched_tokens</code>	[4096, 12288]
<code>tensor_parallel_size</code>	[1, 4]
<code>pipeline_parallel_size</code>	[1, 4]
<code>enable_chunked_prefill</code>	{True, False}
<code>enable_prefix_caching</code>	{True, False}
<code>dtype</code>	{auto, fp16, bf16}

In our search space, `block_size` denotes the memory block size in the vLLM Paged Attention system, larger values can lead to high internal memory fragmentation, whereas smaller values can incur memory management overhead. Furthermore, `max_num_seqs` and `max_num_batched_tokens` are the maximum number of sequences per batch and maximum number of tokens per batch, respectively. `Tensor_parallel_size` splits the tensor into N chunks for multiprocessing, requiring all-gather operations at the end reconstruct the full tensor for the next computation. `Pipeline_parallel_size` splits the model into layers, requiring point-to-point communication for sending and receiving intermediate data. In the system, we can also indicate `chunked_prefill`[7, 1] and `prefix_caching`[10] for computation and memory optimisation. Additionally, `dtype` specifies the numerical precision for model weights and computation.

In vLLM deployment, these configurations are static once deployed. However, different workloads

may have different optimal configurations [9]. Hence, real-time changes in deployment workload often require a new set of optimal configurations to optimise energy efficiency and throughput.

## 2.2 Multi-Objective Bayesian Optimization (MOBO) and Acquisition Function

To find the optimal configuration in the search space shown in Table 1, research such as SCOOT[3] has employed multi-objective Bayesian optimisation (MOBO) to automatically tune system performance across various service-level objectives. The pipeline of MOBO is as follows: (1) Use Sobol sampling to generate configuration trials from the search space. (2) Run the benchmark using that configuration to observe the multi-objective metrics results. (3) Set up a surrogate model, such as a Gaussian Process, to predict the objectives. (4) Lastly, use the acquisition function to assess the current configuration and suggest the following configuration to benchmark.

Throughout this process, we avoid running expensive evaluations for every possible configuration, instead, we use a Gaussian Process to predict the best configuration given the current workload. For a multi-objective surrogate model, there are two ways of implementation: One surrogate by aggregating objective functions and multiple surrogates for each objective function[4]. For the acquisition function, we can use qNEHVI [6] that works with the independent posterior predictions from each surrogate to identify Pareto-optimal solutions.

## 3 Experimental Design and Methodology

### 3.1 System Design

The complete system design of the project is shown in Figure 1. First, I define the search space within the vLLM system configuration, then run the benchmark. Each benchmark is measured in terms of throughput (tokens / second) and energy consumption (Utilised Codecarbon for energy tracking: CPU energy measured by Intel RAPL, GPU energy measured by nvidia-smi, and RAM energy measured by psutils). Next, the multi-objective metrics will be fed into a surrogate model for training, and the surrogate’s

predictions will guide the qNEHVI acquisition function in selecting the following configuration to evaluate. In the experiment, the number of initial Sobol samples is set to 10, and the subsequent MOBO acquisition-function samples are set to 30 to train the surrogate model, all of which is running on 1-4 L4 GPUs. The model I used in the experiment is *mistralai/Mistral – 7B – Instruct – v0.1*, and the dataset is *Open – Orca/OpenOrca* from Hugging Face. After all runs, the Pareto-optimal system configuration is determined and profiled. The saved trained models will also be used in calibration analysis to investigate the discrete-continuous search space trade-off in LLM inference multi-objective optimisation.

### 3.2 Discrete-Continuous Trade-offs in MOBO

In MOBO, parameters such as *block\_size*, *tensor\_parallel\_size*, *pipeline\_parallel\_size*, *max\_num\_seqs* are typically power-of-2 values, enabling computation coupled with hardware tiling, alignment with GPU memory cache lines, and CUDA optimisation to maximise performance in system-algorithm co-design. In Ax’s MOBO implementation, we could define these parameters as a *choice*, which represents a discrete search space, or as a *range*, which represents a continuous search space. Under the hood, both use the same Matern 5/2 kernel within a Gaussian Process. The difference lies in input transformation and candidate generation. A *range* parameter works in continuous, unconstrained space so that the surrogate model (Gaussian Process) is smoothly differentiable in proposing a new candidate point for the next iteration. In contrast, a *choice* parameter discretises and projects the continuous space to the discrete space, creating a mismatch between the Gaussian Process’s smoothness assumption, but reduces the search space dramatically, as vLLM often only accepts certain values. For example, it only accepts multiples of 16 for *block\_size*.

To quantify the discrete-continuous trade-offs in MOBO, this project will save the trained multi-objective Gaussian Process models for future inference performance prediction. We sample 10 held-out configurations that is not present in the training process, then we run Gaussian process

inference to predict the throughput and energy consumption. We also run actual vLLM benchmark on these 10 held-out configurations to serve as ground truth. By calculating the Root Mean Square Error and the Coefficient of determination ( $R^2$ ) between actual and predicted performance for different Gaussian Processes (trained in continuous vs. discrete search spaces), we will quantify the effectiveness and trade-offs of each. Additionally, we evaluate the optimisation performance by comparing the Pareto hypervolume achieved by each approach under a fixed evaluation budget (i.e., the same number of trials). Thus, we will be able to assess whether the improved prediction accuracy translates into better optimisation outcomes.

## 4 Results and Performance Analysis

### 4.1 Calibration Analysis for Discrete-Continuous Trade-offs

Table 2: Calibration Analysis of Continuous vs. Discrete GP on 10 held-out configurations

Metric	Cont.	Disc.	Better
Throughput RMSE	378.73	<b>52.48</b>	Discrete
Throughput $R^2$	-4.71	<b>0.89</b>	Discrete
Energy RMSE	1.23	<b>0.60</b>	Discrete
Energy $R^2$	-51.44	<b>-11.26</b>	Discrete

MOBO with a discrete search space is shown to be better in our calibration analysis, as shown in Table 2. Among all the 10 held-out configurations, MOBO with a discrete search space achieves lower throughput RMSE compared to the ground truth (metrics collected through actual benchmarking), and higher  $R^2$  in throughput calibration analysis. Additionally, the throughput  $R^2$  performance is 0.89, close to 1, showing its good performance in predicting the throughput. In energy prediction, MOBO with a discrete search space also achieved a better result by having a lower RMSE and a higher  $R^2$  result compared to MOBO with a continuous search space. However, MOBO with a discrete search space shows poorer performance in predicting the energy consumption. In most of the configurations, we notice that both Gaussian Processes tend to overestimate the

energy consumption for each run.

For MOBO with a continuous search space, we observe that both throughput and energy prediction have negative  $R^2$ , indicating that the Gaussian Process model performs worse than a mean predictor. These observations suggest that the projection to a discrete search space improves the prediction even though the smoothness assumption in the Gaussian Process is mismatched.

### 4.2 The Pareto-Optimal Frontier

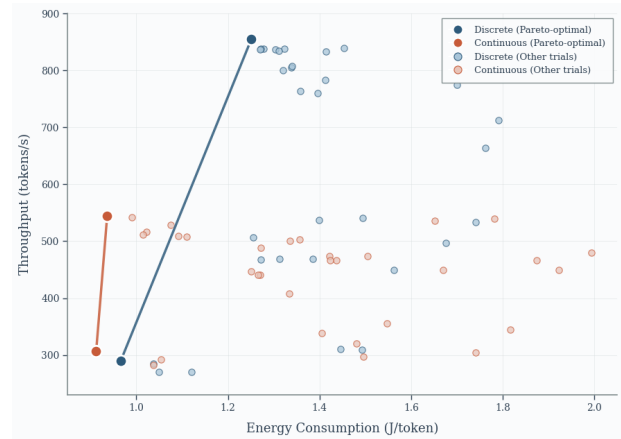


Figure 2: Pareto frontiers for discrete vs. continuous hyperparameter search spaces, showing the throughput–energy trade-off. Each dot represents a specific configuration with its metrics observed during the MOBO run.

For both MOBO problems with continuous and discrete search spaces, we identified two Pareto frontiers, as shown in Table 3. In our experiment, to calculate the hypervolume of the Pareto configurations found in MOBO, we used the worst observed throughput (200 tokens/second) and the worst observed energy consumption (2 J/token) as reference points. As a result, calculations show that MOBO with a discrete search space has a higher hypervolume (466.64) than MOBO with a continuous search space (369.54). However, in Figure 2, we can see that MOBO with a continuous search space covers more volume in the low-energy objective region, but fails to explore the high-throughput region, thus contributing to a lower hypervolume score, which causes it to underperform the discrete search space in

Table 3: Pareto-Optimal Configurations from Discrete and Continuous Search Spaces, configurations were identified during the MOBO search process. Values for throughput and energy represent the mean of 5 independent benchmark runs.

Space	ID	block size	seqs	tokens	tp	pp	chunked	prefix	dtype	Tput (tok/s)	Energy (J/tok)
vLLM Default	–	16	1024	8192	4	1	T	F	auto	805.5	0.986
Discrete	24	32	256	12288	4	1	T	T	auto	844.1	0.952
Discrete	11	32	128	12288	1	1	F	T	auto	291.0	0.892
Continuous	30	80	154	9273	1	3	T	F	auto	543.2	0.756
Continuous	24	48	91	6596	1	1	T	T	auto	299.2	0.846

general.

Furthermore, to ensure statistical robustness and mitigate the impact of system variance, we report the averaged throughput and energy for the Pareto-optimal configurations from 5 additional independent runs in Table 3. When we compare the vLLM default performance characterised by its engine default parameter, we observed that both Pareto frontiers obtained by MOBO’s continuous search space are optimising for energy consumption, where both have lower J/tokens, ranging from 0.756 to 0.846 J/tokens, compared with the default of 0.986 J/tokens, indicating a 14-23% energy reduction in batch inference. For the Pareto frontiers discovered by the MOBO with a discrete search space, we have two configurations that further exemplify the throughput–energy trade-offs in LLM inference serving. For configuration ID 24, the MOBO with a discrete search space is optimising for throughput, achieving 844.1 tokens/second, a 4.79% improvement over the baseline of 805.5 tokens/second, with its energy consumption being slightly lower than the vLLM default. For configuration ID 11, it is optimising for energy, achieving a low energy consumption of 0.892, 9.53% lower than the vLLM default. Our Table 3 also showed that instead of using 4 GPUs, we could use only 1 GPU to lower the energy consumption per token, which is crucial for responsible inference serving. These results show that, although vLLM has well-tested default parameters, workloads that differ from vLLM’s internal testing require a different optimal configuration for deployment across multiple objectives (throughput and energy consumption). Thus, MOBO can be a valuable tool for automating

the search for the best-suited configuration.

### 4.3 Configuration Analysis (Profiling)

Table 4: Top CUDA kernels by execution time for TP=1 vs TP=4.

Kernel Type	TP=1 (%)	TP=4 (%)
NCCL AllReduce	—	<b>36.2%</b>
GEMM (cutlass)	43.2%	15.6%
Flash Attention	19.0%	14.1%
GEMM (ampere_s1688)	18.5%	—
GEMM (ampere_ldg8)	8.2%	14.5%
<i>Total Model Computation</i>	88.9%	44.2%
<i>Total Collective Ops</i>	0%	36.2%

To further understand why MOBO preferred  $tp = 4$  when optimising for throughput and  $tp$  and  $pp = 1$  when conserving energy, we also conducted profiling to visualise the system-level computation. In Figure 3, we extract the important kernel and memory visualisations from Nsys profiling of the CUDA API and observe that the NCCL operation is present at  $TP = 4$  but absent at  $TP = 1$ . When  $TP = 4$  and tensor parallelism is activated, it distributes matrix operations across four GPU cards, thus increasing the available memory capacity for either a larger model or a larger batch size. Our MOBO, trained on a discrete search space, effectively captures this knowledge: as it increases *tensor\_parallelism\_size* to 4, it also increases *max\_num\_seq* to improve throughput. In tensor parallelism, we also use the NCCL Allreduce operation, which combines the partial results from each GPU to form the complete synchronised tensor.

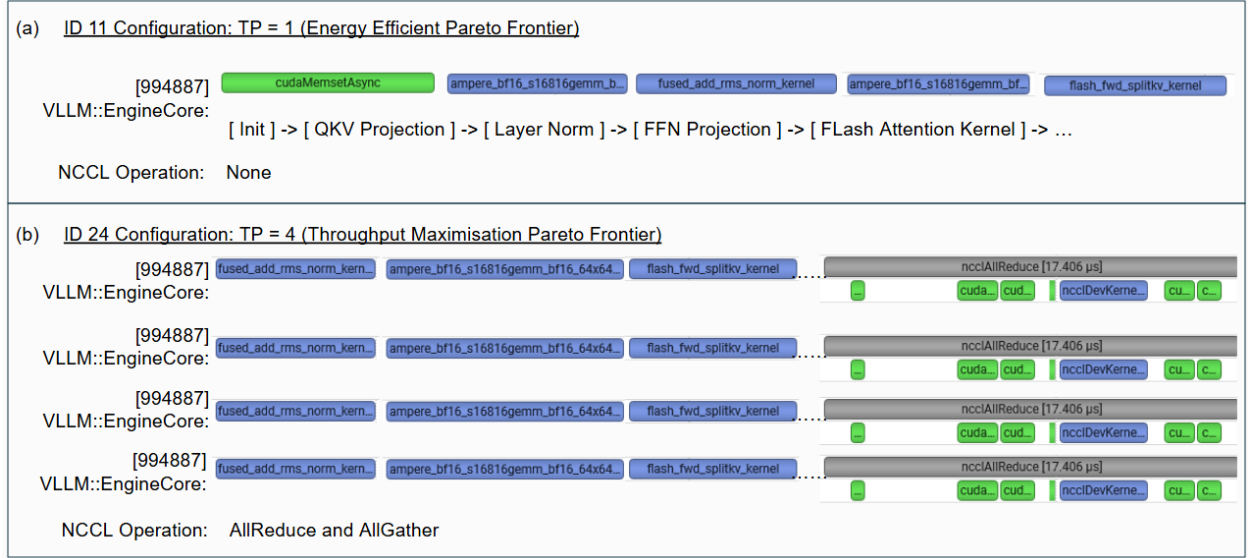


Figure 3: Selective summarisation on Nsys kernel timeline for TP=1 (energy efficiency) vs TP=4 (Throughput maximisation). (a) Single-GPU execution with no communication. (b) Four-GPU parallel execution with ncclAllReduce synchronization barriers (grey), illustrating the throughput-energy trade-off identified by MOBO.

Although NCCL Allreduce introduces additional overhead, this is amortised across batched requests, thereby maximising throughput. For  $TP = 1$ , we don't require all 4 GPUs to remain active and synchronised throughout the inference. Although AllReduce improves throughput, it also increases energy consumption across all participating GPUs. Conversely, when *tensor\_parallelism\_size* is 1, the model layers are executed on a single GPU with no communication overhead. This results in lower total energy consumption per token, which helps explain why MOBO prefers  $TP = 1$  when optimising for energy consumption.

The top CUDA kernels utilised during vLLM inference are also summarised in Table 4 to show the changes in dynamics when we increase the *tensor\_parallelism\_size*.

## 5 Discussion and Conclusions

### 5.1 Practical Recommendations

Our project demonstrates the feasibility of using multi-objective optimisation to identify the optimal configuration for vLLM deployment. Most default parameters used by vLLM focus on throughput. Even

so, the optimal parameter for throughput optimisation might change when the type of workload changes, whether it's coding, summarisation, or conversation, hence static deployment for vLLM is not sufficient for responsible inference. Furthermore, multi-objective optimisation enables the deployer to select whether to optimise for throughput or energy consumption. When the inference workload is low, e.g., at midnight, the deployer can automate redeployment by selecting an updated configuration with an energy-efficient objective.

On top of that, our project explores continuous v.s. discrete search space trade-off. Empirical results show that a discrete search space significantly reduces search complexity, even though it violates the Gaussian Process smoothness assumption, it still accelerates convergence. The MOBO with a discrete search space also achieved comparable Pareto frontier quality within fewer iterations, thus making it more practical for a production environment.

### 5.2 Future work

In future work, this foundational work could be integrated into an online serving system by enabling real-time, lightweight rescheduling. For example,

when the workload shifts from conversation to long-context processing, the optimal TP and PP configuration may change, therefore, a lightweight rescheduling using MOBO could improve energy efficiency and meet the Service-Level Objective.

Additionally, the scheduler design for inference requests is an important area that provides significant optimisation opportunities. The current vLLM scheduler uses heuristics for batching decisions, preemption strategies, and memory allocation. The idea of MOBO can also be extended to jointly optimise scheduler parameters and parallelism configurations, thereby optimising performance.

### 5.3 Conclusions

This work presents a multi-objective Bayesian optimisation framework for configuring the vLLM inference system, showing that MLSys practitioners can also optimise for energy consumption alongside throughput, for responsible inference serving.

### References

- [1] Amey Agrawal, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav S Gulavani, and Ramachandran Ramjee. Sarathi: Efficient llm inference by piggybacking decodes with chunked prefills. *arXiv preprint arXiv:2308.16369*, 2023.
- [2] Jeff Barr. Amazon ec2 update – inf1 instances with aws inferentia chips for high performance cost-effective inferencing. AWS News Blog, December 2019.
- [3] Ke Cheng, Zhi Wang, Wen Hu, Tiannuo Yang, Jianguo Li, and Sheng Zhang. Scoot: Slo-oriented performance tuning for llm inference engines. In *Proceedings of the ACM on Web Conference 2025*, pages 829–839, 2025.
- [4] Tinkle Chugh. Mono-surrogate vs multi-surrogate in multi-objective bayesian optimisation. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 2143–2151, 2022.
- [5] Jae-Won Chung, Jeff J Ma, Ruofan Wu, Jiachen Liu, Oh Jun Kweon, Yuxuan Xia, Zhiyu Wu, and Mosharaf Chowdhury. The ml. energy benchmark: Toward automated inference energy measurement and optimization. *arXiv preprint arXiv:2505.06371*, 2025.
- [6] Samuel Daulton, Maximilian Balandat, and Eytan Bakshy. Parallel bayesian optimization of multiple noisy objectives with expected hypervolume improvement. *Advances in neural information processing systems*, 34:2187–2200, 2021.
- [7] Connor Holmes, Masahiro Tanaka, Michael Wyatt, Ammar Ahmad Awan, Jeff Rasley, Samyam Rajbhandari, Reza Yazdani Aminabadi, Heyang Qin, Arash Bakhtiari, Lev Kurilenko, et al. Deepspeed-fastgen: High-throughput text generation for llms via mii and deepspeed-inference. *arXiv preprint arXiv:2401.08671*, 2024.
- [8] HPCwire Staff. Aws to offer nvidia’s t4 gpus for ai inferencing. *HPCwire*, March 2019.
- [9] Youhe Jiang, Fangcheng Fu, Xiaozhe Yao, Taiyi Wang, Bin Cui, Ana Klimovic, and Eiko Yoneki. Thunderserve: High-performance and cost-efficient llm serving in cloud environments. *arXiv preprint arXiv:2502.09334*, 2025.
- [10] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th symposium on operating systems principles*, pages 611–626, 2023.
- [11] Zhuohan Li, Lianmin Zheng, Yinmin Zhong, Vincent Liu, Ying Sheng, Xin Jin, Yanping Huang, Zhifeng Chen, Hao Zhang, Joseph E Gonzalez, et al. {AlpaServe}: Statistical multiplexing with model parallelism for deep learning serving. In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*, pages 663–679, 2023.
- [12] Pratyush Patel, Esha Choukse, Chaojie Zhang, Íñigo Goiri, Brijesh Warriar, Nithish

- Mahalingam, and Ricardo Bianchini. Characterizing power management opportunities for llms in the cloud. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, pages 207–222, 2024.
- [13] David Patterson, Joseph Gonzalez, Quoc Le, Chen Liang, Lluís-Miquel Munguia, Daniel Rothchild, David So, Maud Texier, and Jeff Dean. Carbon emissions and large neural network training. *arXiv preprint arXiv:2104.10350*, 2021.
- [14] Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, Jonathan Heek, Kefan Xiao, Shivani Agrawal, and Jeff Dean. Efficiently scaling transformer inference. *Proceedings of machine learning and systems*, 5:606–624, 2023.
- [15] Samyam Rajbhandari, Conglong Li, Zhewei Yao, Minjia Zhang, Reza Yazdani Aminabadi, Ammar Ahmad Awan, Jeff Rasley, and Yuxiong He. Deepspeed-moe: Advancing mixture-of-experts inference and training to power next-generation ai scale. In *International conference on machine learning*, pages 18332–18346. PMLR, 2022.
- [16] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.
- [17] vLLM Team. Asyncllm (v0.11.0) – vllm api reference, 2024. Accessed: 2024-12-30.
- [18] vLLM Team. Llmengine (v0.11.0) – vllm api reference, 2024. Accessed: 2024-12-30.
- [19] Bingyang Wu, Yinmin Zhong, Zili Zhang, Shengyu Liu, Fangyue Liu, Yuanhang Sun, Gang Huang, Xuanzhe Liu, and Xin Jin. Fast distributed inference serving for large language models. *arXiv preprint arXiv:2305.05920*, 2023.
- [20] Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soojeong Kim, and Byung-Gon Chun. Orca: A distributed serving system for {Transformer-Based} generative models. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pages 521–538, 2022.
- [21] Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. {DistServe}: Disaggregating prefill and decoding for goodput-optimized large language model serving. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pages 193–210, 2024.