UNIVERSITY OF CAMBRIDGE
Department of Computer Science and Technology

Online Workload Allocation and Energy Optimization in Large Language Model Inference Systems

Grant Wilkins

Churchill College

3 June 2024

Submitted in partial fulfillment of the requirements for the
Master of Philosophy in Advanced Computer Science

Total page count: 59

Main chapters (excluding front-matter, references and appendix): 41 pages (pp 12–52)

Main chapters word count: 11945

Methodology used to generate that word count: The Overleaf word count feature of the entire document.

Declaration

I, Grant Wilkins of Churchill College, being a candidate for the Master of Philosophy in Advanced Computer Science, hereby declare that this report and the work described in it are my own work, unaided except as may be specified below, and that the project report does not contain material that has already been used to any substantial extent for a comparable purpose. In preparation of this project report I did not use text from AI-assisted platforms generating natural language answers to user queries, including but not limited to ChatGPT. I am content for my project report to be made available to the students and staff of the University.

We note that similar results to Section 4.4 and 5.2 appeared at the 2024 ACM e-Energy conference in the paper "Hybrid Heterogeneous Clusters Can Lower the Energy Consumption of LLM Inference Workloads" by Grant Wilkins, Richard Mortier, and Srinivasan Keshav at `https://dl.acm.org/doi/10.1145/3632775.3662830`.

Signed: Grant Franklin Wilkins

Date: 3 June 2024

Work expands so as to fill
the time available for completion.

C. NORTHCOTE PARKINSON

Abstract

The rapid adoption of Large Language Models (LLMs) has furthered natural language processing and helped text generation, question answering, and sentiment analysis. However, deploying these models in production environments poses significant challenges, particularly for energy efficiency. This thesis addresses these challenges by investigating how dynamic workload allocation can improve energy management for LLM inference within heterogeneous data centers.

We first profile various LLMs' energy consumption and runtime on different hardware configurations, including NVIDIA GPUs, Apple Silicon, Intel CPUs, and AMD CPUs. Our experimental results highlight the trade-offs between model complexity, the number of input/output tokens, and system performance. We develop predictive models for energy consumption and runtime, which are the foundation for our optimization strategies.

Our primary contributions include the development of a workload-aware heterogeneous data center model that balances energy consumption and runtime based on operational demands. Next, we propose an offline routing mechanism that partitions and routes queries to minimize energy consumption and runtime while maintaining high accuracy. Additionally, we outline an online routing approach that dynamically adjusts to incoming queries in real-time, incorporating queue awareness to enhance resource utilization and reduce wait times. Through extensive simulations, we demonstrate that our algorithms can optimize energy consumption through trade-offs with accuracy. We allow operational tuning in all simulations, depending on current system requirements.

This thesis contributes to the broader goal of sustainable computing by providing actionable insights and practical solutions for optimizing LLM inference. Our findings underscore the importance of energy-efficient AI deployment. We open-source the datasets and benchmarks we developed during this research, enabling further exploration and innovation in energy-efficient LLM inference.

Acknowledgements

This project would not have been possible without the wonderful support of Professor Richard Mortier and Professor Srinivisan Keshav. Their guidance, experience, and advice were invaluable during this project. I acknowledge Clemson University's Palmetto Computer Cluster and Argonne National Laboratory's Leadership Computing Resource Center for their generous provision of compute. I would also like to thank Adam Wierman for preliminary discussions about sustainable datacenters and environmental equity that helped shape our research focus.

I also would like to thank the Churchill Scholarship Foundation for their generous financial and professional support to pursue my degree and have a wonderful year here at Cambridge. Also, thank you to the many friends and colleagues I have met this year. All of this would not have been as enriching, fun, or enjoyable without you all.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Large Language Models (LLMs) such as OpenAI's GPT-4 [38] and Google's Gemini [52] are revolutionizing natural language processing (NLP), text generation, and translation. The ability of LLMs to understand and generate human-like language has opened up new possibilities for AI-enabled applications, impacting many industries [12].

However, deploying LLMs for inference requires large pools of heterogeneous accelerators to perform expensive tensor calculations, requiring systems designers to rethink traditional cloud service techniques. One of the main factors in determining resource allocation for LLMs is their on-device memory requirements [3, 22, 26, 60]. As models increase in size and complexity, their memory requirements grow. These models then require more accelerators and, in general, consume more energy [8].

Because of this trend, the energy consumption of AI systems has become a significant concern. Many studies have highlighted the large energy budgets that training and serving large-scale models require [8, 28, 29]. Therefore, it is necessary to consider strategies for optimizing the energy efficiency of these systems without compromising performance. This focus is particularly relevant for LLM inference, where, for example, over a year of serving LLM inference can consume over 25× more energy than training a model [8]. The environmental implications of these energy-intensive AI systems stretch beyond energy usage into carbon emissions and water consumption associated with cooling data centers [28, 29]. However, this study focuses primarily on energy consumption as a proxy for environmental and sustainable efforts. We acknowledge that more dimensions of this problem affect the environment and contribute to climate change.

We hypothesize that by evaluating LLM performance on various computational resources (e.g., different generations, makes, and models of GPUs and CPUs) and varying aspects of LLM inference (e.g., number of input and output tokens, number of parameters), it is possible to minimize energy consumption while maintaining high throughput and accuracy. However, finding an optimal hardware and workload-aware strategy is complex, requiring consideration of a wide range of factors, including the characteristics of the LLMs themselves, the energy usage of a cluster's hardware, and the impact of different workloads on inference tasks.

Objectives and Contributions

This thesis addresses the challenge of optimizing energy consumption and runtime for LLM inference by leveraging data from various systems with different hardware. We aim to present optimal system and routing configurations, investigate the impact of LLM complexity parameters on energy efficiency, and determine if the trade-off between model complexity and inference speed aligns with energy consumption patterns across different hardware architectures.

We begin in Chapter 2 with a survey of literature on serving LLM inference, the energy consumption of LLMs, and ways to perform energy-efficient scheduling, detailing also how our study improves upon the known solutions and extends the field. Next, in Chapter 3, we formalize our scheduling and energy efficiency goals using predictive models for energy, runtime, and accuracy.

To further achieve our objectives, we first analyze several LLMs' energy consumption and runtime with different parameter sizes across various hardware configurations and inference workloads. The models we develop in Chapter 3 and the results detailed in Chapter 4, form the foundation for developing highly explainable, predictive models for the runtime and energy consumption based on the number of input and output tokens for different LLMs and hardware systems. These models are crucial for understanding how LLM complexity parameters, such as the number of model parameters and token input/output size, affect energy efficiency in specific system configurations, as discussed in Chapter 4 and further explored in Chapter 5.

Additionally, we compare the energy efficiency and performance characteristics of LLM inference across different hardware platforms. This comparison reveals whether the relationship between model complexity and inference speed is consistent across various architectures. Our findings in Chapter 5 highlight the implications for deploying LLM inference services on hardware, considering cost, scalability, and energy efficiency. We propose solutions such as a workload-aware scheduler for heterogeneous clusters, an offline mechanism for routing inference workload batches to different LLMs, and a tunable online routing approach that trades off energy consumption and accuracy.

One of the key contributions of this thesis is the open sourcing of comprehensive datasets and a benchmark suite for evaluating the energy efficiency of LLM inference [1]. Our results, framework, and simulations provide researchers and practitioners with valuable tools to assess the impact of their design choices and optimize LLM deployment in production environments.

[1]See `https://github.com/grantwilkins/energy-inference.git`.

Chapter 2

Related Work

2.1 LLM Inference as a Service

Determining how to efficiently offer LLM inference as a service is an open question. Due to their large memory footprint and often expensive runtimes, there have been many attempts to optimize their deployment.

Several systems have been developed to enhance the performance and efficiency of LLM inference services. AlpaServe [30] implements model parallelism and devises model-parallel placements that align with request arrival patterns, aiming to optimize the quality of service. FastServe [57] introduces an innovative skip-join Multi-Level Feedback Queue (MLFQ) scheduler and employs iteration-level preemption to enhance job completion efficiency. This approach allows for dynamic resource allocation adjustment based on job priorities and current system load, thereby reducing latency and improving throughput. SpotServe [36] focuses on deploying LLM serving systems on preemptive instances, aiming for a balance between costs and performance. By utilizing spot instances, which are typically more cost-effective than reserved instances, SpotServe can reduce operational expenses while maintaining acceptable performance levels. Addressing service quality, Wang et al. [55] study the efficiency and reliability of LLM serving, highlighting the challenges of maintaining high-quality service while managing computational loads effectively. This perspective is an essential first step in defining the quality of service for LLM systems.

While these systems highlight the ongoing efforts to optimize LLM inference services for offline and online scenarios, ours is the first to take an energy-optimal scheduling approach. As the demand for real-time AI services grows, sustainable inference serving will be crucial in meeting quality of service requirements while balancing environmental effects.

2.2 LLM Cluster Resource Management

As discussed, LLMs require substantial computational resources and heterogeneous accelerators to perform tensor computations efficiently. As these models grow, their memory footprint increases correspondingly, exacerbated by the precision of the floating-point representation used for parameters.

Several services have been developed to address these challenges and optimize LLM allocation and runtime across various resource scenarios, each with unique software constraints. Alpa [61] automates both inter- and intra-operator parallelism, aligning model parallelism with the computational capabilities of the available hardware. DeepSpeed [3] offers a multi-GPU inference solution that minimizes latency and maximizes throughput for dense and sparse transformers, utilizing heterogeneous memory across CPU, NVMe, and GPU to support models with more memory requirements than the aggregate GPU memory.

Orca [58] introduces dynamic batch scheduling for LLM inference, aiming to balance the load across available accelerators and minimize inference latency. Similarly, FlexGen [48] targets high-throughput LLM inference using limited resources, such as a single commodity GPU. FlexGen combines memory and computation from GPU, CPU, and disk, employing linear programming to optimize tensor storage and access patterns.

LLM-PQ [60] focuses on phase-aware partitioning and adaptive quantization for serving LLMs on heterogeneous clusters. This approach dynamically adjusts model partitioning and quantization levels based on the current phase of the inference task, improving efficiency and performance. PagedAttention [26] addresses managing the KV cache memory for each request. Inspired by virtual memory and paging techniques, PagedAttention reduces memory waste and allows flexible sharing of KV cache, significantly improving throughput for popular LLMs.

While these existing systems excel in optimizing specific aspects of LLM serving, our approach provides a comprehensive framework that balances energy efficiency, runtime, and accuracy through both offline and online routing algorithms. Our approach's unique contribution lies in its holistic optimization strategy, which dynamically adjusts to operational demands, balancing multiple objectives rather than focusing on a single performance metric. Our work will integrate seamlessly depending on the optimization strategy chosen from the above service. A data center operator only needs to create new energy consumption and runtime models, and then our system meshes with the outlined research. By integrating energy consumption models, runtime predictions, and workload-aware scheduling, we offer a versatile and adaptive solution that enhances the overall efficiency and sustainability of LLM inference in production environments.

2.3 LLM Inference Energy Studies

Characterizing the energy consumption of LLM inference has been a recent topic of interest. Desislavov et al. [11] provide a timely examination of trends in AI inference energy consumption, arguing that while performance has increased dramatically, energy consumption has not escalated at the same pace, thanks to hardware optimizations and algorithmic innovations. This perspective is crucial as it suggests the potential for further optimizations in LLM inference tasks, which are typically energy-intensive. Chien et al. [8] discuss more significant trends in LLM inference energy consumption and do not focus on device-level energy modeling benefits. Samsi et al. [47] explore the energy consumption of Meta's Llama LLMs for different batch sizes and numbers of GPUs, showing the potential energy benefits of these tunable parameters. Stojcovik et al. [49] discuss the impacts of GPU frequency scaling on the energy efficiency of serving LLMs; however, at this point, this work is only a characterization and not an applied analysis. Anderson et al. [4] propose carbon-aware data center software that could complement physical hardware adjustments by making energy and carbon metrics visible to application developers, encouraging more energy-efficient coding practices.

Our study is the most extensive comparison of energy consumption and LLM inference runtime across various systems and models.

2.4 Energy-Aware Data Center Scheduling

There is a large body of work that focuses on energy-aware scheduling [13, 21, 31, 35, 45, 50], but none focus on the specific challenge of developing workload-aware models for LLM inference. Focusing on energy consumption, Hu et al. [20] analyze deep learning workloads in GPU data centers, offering insights into energy conservation strategies through workload scheduling. This research aligns with our objectives by confirming the important role of scheduling in reducing energy footprints. Li et al. [27] introduce Clover, which promises to minimize carbon emissions for serving AI inference. Unlike our study, this work does not explicitly consider LLMs or a per-model function to capture energy and runtime, instead focusing on carbon-emission patterns for a data center. Gu et al. [17] presents PowerFlow, a tool that uses clock-frequency data from GPUs to minimize energy consumption as a scheduling decision. However, their study does not consider LLMs and is not necessarily workload-aware. Patel et al. introduce POLCA [39], which can provide a way to automatically power-cap based on existing workload traces. Li et al. [29] focuses on delivering a geographic load balancing perspective for AI inference, optimizing environmental equity. However, their model considers large-scale workload traces, not device-level energy and runtime data.

Chapter 3

Problem Formulations

The deployment of LLMs for inference in production cloud environments requires careful consideration of the energy efficiency, runtime, and model accuracy. This chapter presents a series of problem formulations that allow us to optimize these factors within data centers hosting multiple LLMs. In this chapter and further analysis, we adopt the following basic notation.

In our setup, a query is a textual input to an LLM with a known number of output tokens.[1] Q represents a batch of queries that have been "tokenized," such that each item in this multiset is a tuple of an input and output token count, $(\tau_{in}, \tau_{out}) \in \mathbb{N}^2$, and $Q = \{(\tau_{in}, \tau_{out})_1, \ldots, (\tau_{in}, \tau_{out})_i\}$. Depending on the problem's focus, we aim to divide this workload among diverse hardware or LLMs. In cases where we consider time-varied arrivals, we define a time-evolving function, $Q(t) : [0, \infty) \to (\mathbb{N}^2)^m$, to represent the m-tuples of the number of input and output tokens for each query that have arrived in our system by time t.

We denote available hardware systems as a set $\mathcal{S} = \{1, \ldots, S\}$. We denote available LLMs as a set $\mathcal{K} = \{1, \ldots, K\}$.

Functions $e_{K,S}(\tau_{in}, \tau_{out})$, $r_{K,S}(\tau_{in}, \tau_{out})$, $a_{K,S}(\tau_{in}, \tau_{out}) : \mathbb{N}^2 \to [0, \infty)$ are used to denote models for the energy consumption, runtime, and accuracy of a given LLM K on hardware S. At points where there is only one type of hardware or LLM, it is necessary to drop the K or S, at which point we will clarify the context. Each of these functions has a normalized counterpart $\widehat{e_{K,S}}, \widehat{r_{K,S}}, \widehat{a_{K,S}} : \mathbb{N}^2 \to [0, 1]$ that maps the estimated values to [0,1] to make these different metrics comparable to aid in optimization. We normalize by dividing by the largest known value of each quantity during the optimization or online routing, meaning that the normalization adjusts with arriving values.

$\zeta \in [0, 1]$ is an operational parameter for trade-offs in our various cost functions.

[1]Realistically, the number of output tokens is not known *a priori* but can be estimated through analyzing past input-output pairs like in Zheng et al. [62].

3.1 Workload Routing in a Heterogeneous Data Center

Our first objective is to construct a simple model of operational demands of a heterogeneous data center with multiple hardware types hosting a single LLM. We drop the K-index mentioned above because there is only one LLM. We define this problem's cost as a trade-off between energy and runtime by assigning queries to different hardware systems, $\mathcal{S} = \{1, \ldots, S\}$. To optimally route to other hardware, we present the following optimization problem.

$$\min_{\{Q_S\}_{S \in \mathcal{S}}} \sum_{S \in \mathcal{S}} \sum_{(\tau_{in}, \tau_{out}) \in Q_S} \zeta \widehat{e_S}(\tau_{in}, \tau_{out}) + (1-\zeta) \widehat{r_S}(\tau_{in}, \tau_{out}) \tag{3.1}$$

$$\text{s.t.} \quad \bigcup_{S \in \mathcal{S}} Q_S = Q \tag{3.2}$$

$$Q_I \cap Q_J = \emptyset \text{ for } I \neq J, \forall I, J \in \mathcal{S}. \tag{3.3}$$

$\widehat{e_S}(\tau_{in}, \tau_{out})$ is the normalized energy consumed by system S during inference. $\widehat{r_S}(\tau_{in}, \tau_{out})$ is the normalized inference runtime. Q_S is the portion of Q that is assigned to run on system S.

We formulate this general assignment problem (GAP) that ensures each query is processed exactly once, optimizing for either energy efficiency or quick response times, depending on the operational demands reflected in ζ. Certain systems may perform better on specific tasks based on the workload characteristics, such as the need for rapid response times. Adjustments in ζ allow the data center to change the focus between minimizing energy consumption and reducing runtime as operational priorities change.

3.2 Workload Routing in a Data Center Serving Multiple LLMs

Building upon the previously outlined problem, we consider another scenario of hosting several LLMs in a data center with a single type of CPU+GPU resource. Adopting a similar optimization framework, we will use our energy performance data for different LLMs $\mathcal{K} = \{1, \ldots, K\}$ to optimize energy and accuracy by optimizing hosting multiple LLMs in a single system.

3.2.1 Modeling a Data Center

Here, we outline our assumptions about the resources available in our data center. Assume we have $n_K \in \mathbb{N}$ GPUs assigned to each LLM, K, with $N_{system} \in \mathbb{N}$ total GPUs. Deciding how many instances of each model our system should have ready is outside the scope of this work. Practical considerations for this would be customer usage history, the relationship between n_K and N_{system}, performance data, and the reliability of the service. Therefore, we assume that there is a proportionality γ_K of the accelerators in our system assigned to each K. We have that $\gamma_K \in [0,1], \forall K$, and $\sum_{K \in \mathcal{K}} \gamma_K = 1$. We can see that each LLM K has $\gamma_K N_{system}$ GPUs assigned to them and $\left\lfloor \frac{\gamma_K N_{system}}{n_K} \right\rfloor$ instances within our system.

3.2.2 Offline Routing Queries to Different LLMs

In practice, most datacenters serving LLM inference will be a variation of an online and interactive service. However, one can imagine potential applications of an offline framework such as in overnight data analysis, routing batch arrivals, or for training performance models to automate decision-making. Therefore, we proceed with developing a framework to design an offline routing framework for serving multiple LLMs.

With a model of the resources in our data center, we need to design a framework that will route our workload to minimize the energy and maximize the accuracy of our system. Keeping our notation for a workload of queries is a set, Q, then we can adopt the following notation for a disjoint, multiset partition of our query workload to each LLM, $Q = \langle Q_K \rangle_{K \in \mathcal{K}}$.

Accuracy is not a straightforward metric, so we attempt to quantify it here. Many benchmarks attempt to capture the "accuracy" of an LLM. Examples of these are the MMLU [18], HellaSwag [59], etc., and each have their criticisms. To sidestep the problems introduced by each specific test for accuracy [34], we will use the HuggingFace Leaderboard's [5] *average accuracy* that takes an average of every single recorded accuracy available from their repository of datasets and tests. Therefore, moving forward, when we refer to the accuracy of an LLM, we refer to the score, $A_K \in [0,1]$, which we take from this leaderboard.

For optimization purposes, we must define a function based on the constant A_K. We propose $a_K : \mathbb{N}^2 \to [0, \infty)$, a monotonically increasing function based on the number of input and output tokens that a model K ingests and produces. Therefore, for a model K processing tokens (τ_{in}, τ_{out}) we have

$$a_K(\tau_{in}, \tau_{out}) = A_K \tau_{in} + A_K \tau_{out}. \tag{3.4}$$

Let ζ denote a tuning parameter that lets a data center operator trade off energy for accuracy. Let $|Q|$ represent the total number of queries in our workload, and $|Q_K|$ represent the total number of queries each model K processes. We can then formulate our workload assignment problem as

$$\min_{Q_K \in Q} \sum_{K \in \mathcal{K}} \sum_{(\tau_{in}, \tau_{out}) \in Q_K} \zeta \widehat{e_K}(\tau_{in}, \tau_{out}) - (1-\zeta)\widehat{a_K}(\tau_{in}, \tau_{out}) \tag{3.5}$$

$$\text{s.t.}, 0 < \frac{|Q_K|}{|Q|} < 1 \tag{3.6}$$

$$Q = \bigcup_{K \in \mathcal{K}} Q_K \tag{3.7}$$

$$Q_I \cap Q_J = \emptyset, I \neq J, \forall I, J \in \mathcal{K}, \tag{3.8}$$

where Equations 3.7 and 3.8 define the partition coverage of the workload, and Equation 3.6 ensures we give each LLM some queries. In our implementation, we dynamically normalize our energy and accuracy measures across all the queries to allow us to adjust the scale of costs across different models and query combinations.

3.2.3 Online Routing Incoming Queries to Different LLMs

Offline problems can be helpful for a description of a problem with complete information; however, in real-world inference-serving scenarios, there is limited information about the future. This section details the transition from an offline problem formulation to an online dynamic routing approach, which involves rethinking the system architecture to handle real-time data flow and decision-making. In the offline setup, a scheduler sends queries to different LLMs with a static snapshot of the system's state. However, in an online scenario, these decisions must be responsive to the continuously changing state of the system.

The first step in the transition involves defining the system state that encapsulates all necessary information to make routing decisions. The system state includes:

- $Q_K(t) : [0, \infty) \to (\mathbb{N}^2)^{m_K}$ - the total multiset of tuples for the number of input and output tokens, with cardinality m_K, assigned to each model K by time t.
- q_K - the number of items in the queue awaiting processing on a model K.
- $cost_K$ - the current cost associated with the model K that considers the queries currently processed and in the queue.

In Figure 3.1, we show a basic diagram of the flow of queries through a data center performing real-time scheduling of LLM inference for a set of $\mathcal{K}$ models. Based on the cost structure, the goal is to minimize energy consumption with accuracy in mind using the models for accuracy and energy.

The dynamic routing algorithm is at the core of the online model. It is responsible for assessing incoming queries and assigning them to the appropriate model queue based on the current system state and predictive analytics. Upon the arrival of a new query, the

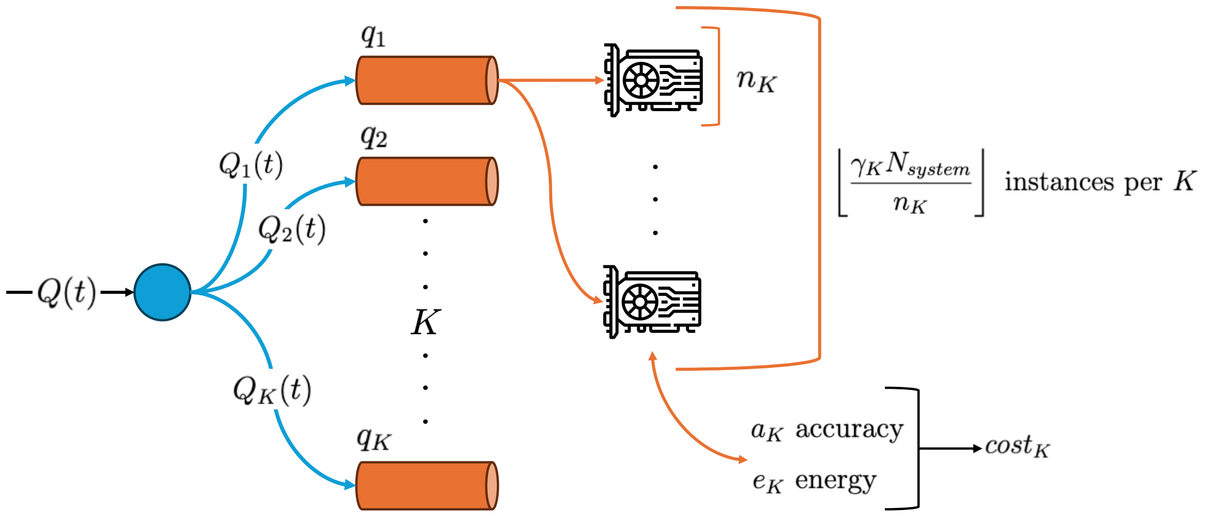


Figure 3.1: System Diagram of Workload-Aware Data Center Hosting K LLMs

system evaluates which model's queue to route the query to by minimizing a cost function that balances the trade-offs between processing time, energy consumption, and accuracy. We summarize this process in Algorithm 1.

Algorithm 1 Handle Incoming Query with Dynamic Routing

```
 1: procedure HandleQuery($query$, $\zeta$, $t$)
 2:     if $t == 0$ then
 3:         $cost = \{K : 0$ for each $K$ in $\mathcal{K}\}$
 4:         $q = \{K :$ deque() for each $K$ in $\mathcal{K}\}$
 5:     end if
 6:     $min\_cost \leftarrow \infty$                    ▷ Temporary variable to find minimum
 7:     $K^* \leftarrow$ None                          ▷ Temporary variable to assign model
 8:     for each $K$ in $\mathcal{K}$ do
 9:         $cost\_with\_query \leftarrow cost[K] +$ CalculateCost$(K, query, \zeta)$
10:         if $cost\_with\_query < min\_cost$ then
11:             $min\_cost \leftarrow cost\_with\_query$
12:             $K^* \leftarrow K$
13:         end if
14:     end for
15:     $q[K^*]$.append($query$)
16:     $cost[K^*] \mathrel{+}= min\_cost$
17: end procedure
18: function CalculateCost($K$, $query$, $\zeta$)
19:     $(\tau_{in}, \tau_{out}) \leftarrow query$
20:     return $\zeta \times \widehat{e_K}(\tau_{in}, \tau_{out}) - (1 - \zeta) \times \widehat{a_K}(\tau_{in}, \tau_{out})$
21: end function
```

At the time, $t = 0$, we initialize the cost, $cost[K]$ for each model, K, to zero, and allow the queue, $q[K]$, to also be empty. Then, as queries are sent to each model K, we update the queue, $q[K]$, and the associated cost, $cost[K]$. We iterate over each model for an incoming query to calculate the potential cost of assigning the query to K. We then update the minimum cost and select the model based on the estimated cost. After considering all models, we put the query into the K^*'s queue and update our cost for the K^* with the added cost.

This process allows us to route queries to different LLMs at the rate at which we can calculate the cost, allowing rapid decision-making that attempts to increase the utilization of resources across the data center while minimizing the cost associated with energy with accuracy in mind.

3.2.4 Queue-Awareness for Improved Quality of Service

Due to the formulation of Algorithm 1, with shifts in ζ, specific models can receive poor utilization rates, and model runtime time and queue wait times can be lengthy. For example, if there were three models with increasing accuracy and parameter size, if $\zeta = 0$ the smallest model would receive no queries, due to the system only optimizing for accuracy. Similarly, if $\zeta = 1.0$, the largest models would experience drought due to minimizing energy. In both cases, wait times can vary depending on the arrival rate of queries to the system. Therefore, to ensure reasonable usage of all models, it is essential to introduce a notion of queue-awareness through a penalty. Also, in an online and interactive service, users expect minimal latency, and for an LLM, this can be gauged via time-to-first-token (TTFT) [1, 26]. Therefore, if we reduce the amount of time spent in the queue, we can decrease the TTFT. This optimization requires polling the lengths of the queues and then using this to penalize over-assignment to a single model. We formulate our queue awareness in Algorithm 2.

Algorithm 2 Queue-Awareness Calculation

```
1: function CalculateCostWithQueueLength($K$, $query$, $q$)
2:     $base\_cost, penalty \leftarrow 0$
3:     $(\tau_{in}, \tau_{out}) \leftarrow query$
4:     $base\_cost \leftarrow \zeta \times \widehat{e_K}(\tau_{in}, \tau_{out}) - (1 - \zeta) \times \widehat{a_K}(\tau_{in}, \tau_{out})$
5:     $penalty \leftarrow 0.1 \times len(q[K])$
6:     return $base\_cost + penalty$
7: end function
```

This penalty assures that the queue for a model K is not overloaded, mitigating growing mean wait times for queries. We note that 0.1 is a scale factor chosen from a sensitivity analysis that allows for ζ still to control the level of accuracy and energy trade-off and does not force query allocation to be equal across all models. This strategy is critical to ensure that the edge cases of ζ do not underutilize resources in our data center and do not result in poor service for users.

Chapter 4

Results

4.1 Measuring Energy Usage

We monitor the energy usage of inference on each system using the CPU/GPU power usage, runtime, and VRAM memory usage to provide a comprehensive view of each model's energy footprint on each hardware system. We consider NVIDIA GPUs, Apple Silicon CPU/GPU, Intel CPUs, and AMD CPUs. We note that in all cases, regardless of system, power, P, energy, E, and time Δt, are all positive real numbers. Note also that the relationship between power and energy is $E = P\Delta t$,

4.1.1 NVIDIA GPUs

We use PyJoules [42], a Python-based energy measurement library, to quantify the energy consumption associated with inference on NVIDIA GPUs. PyJoules provides an interface to `NVML` [37], the NVIDIA Management Library, providing a granular and accurate energy usage assessment for targeted devices. This tool offers real-time energy consumption of GPUs for a given tracked process, which is a key component of our analysis given the GPU-heavy computation involved in LLM inference.

4.1.2 Apple Silicon CPU/GPU

No standard energy measurement tools are available for profiling energy and power usage for Apple Silicon through an API like PyJoules or RAPL. Therefore, we employ a subprocess-based approach to poll macOS' `powermetrics` utility, providing a detailed view of the energy usage during model inference. To capture the energy consumption of the M1 GPU, we execute the `powermetrics` command through a Python subprocess. We record the power draw of each process, including detailed metrics for CPU and GPU power consumption at 200ms intervals. We chose this interval after testing to find the finest granularity measurement without incurring a large overhead ($<$ 1% of CPU from `top`) for the I/O of piping the large `powermetrics` output into a file. The energy monitoring is conducted concurrently with the model inference. A separate thread is dedicated to running the `powermetrics` command, ensuring uninterrupted and real-time data collection. Post-operation, the collected data is processed to extract the recorded power data and then integrated by time to find the energy consumption. We confirm we are the only process on the GPU during inference; therefore, the energy, $E_{Total,GPU}$, is straightforward to calculate for each recorded power value, $P_{GPU,i}$, for a corresponding time step size Δt_i.

$$E_{Total,GPU} = \sum_i P_{GPU,i} \Delta t_i \tag{4.1}$$

The CPU power draw data is less direct as other processes reside on the CPU. However, an "energy impact factor", $\alpha_i \in [0, 1]$, through `powermetrics` allows us to infer how much of the power is due to our Python inference process. Therefore, we calculate the CPU energy, $E_{Total,CPU}$, by multiplying $P_{CPU,i}$ by the "energy impact factor," α_i, at each timestep:

$$E_{Total,CPU} = \sum_i (\alpha_i P_{CPU,i}) \Delta t_i. \tag{4.2}$$

4.1.3 Intel CPUs

We also use PyJoules for Intel CPUs, as in our approach for NVIDIA GPUs. This tool supports RAPL (Running Average Power Limit) interfaces, enabling us to obtain fine-grained energy consumption data. We focus on two primary RAPL domains: Package 0 and Package 1, which correspond to the entire CPU package's energy consumption, including cores, non-core components, and DRAM on supported platforms. We denote Packages 0 and 1 with subscripts 0 and 1, respectively.

PyJoules allows us to capture the energy usage of these domains in real time, enabling us to profile the energy consumption specifically during model inference tasks. To account for base energy consumption unrelated to our inference process, we conduct a pre-analysis phase to measure the CPU's average idle power draw. This idle measurement is then subtracted from the total energy consumption during inference to accurately determine the net energy expenditure attributable to the inference process.

We instrument our code to query the RAPL readings at the start and end of the inference task, calculating the energy consumption as follows:

$$E_{Total,CPU} = \sum_i \Big((P_{0,i} - P_{0,Idle}) + (P_{1,i} - P_{1,Idle}) \Big) \Delta t_i, \tag{4.3}$$

where $P_{0,i}$ and $P_{1,i}$ represent the power draw from Package 0 and Package 1, respectively, and $P_{0,Idle}$ and $P_{1,Idle}$ represent the average idle power draw of the CPU packages, respectively.

4.1.4 AMD CPUs

AMD CPUs do not have a Python API, so instead, we utilize AMDμProf's `timechart` feature, which provides detailed power draw metrics for every core on the chip at fine-grained intervals. By polling AMDμProf at 100ms intervals, we can capture the power draw of each physical core throughout the model inference process.

To ensure we accurately attribute the energy consumption to our inference task, we monitor the CPU core residency through `psutil` and power draw in tandem. This tool allows us to identify and record the specific cores actively engaged in the inference process at each time step. The total energy consumption for the inference task is then calculated by summing the power usage across all active cores and integrating over the time of inference, as follows:

$$E_{Total,CPU} = \sum_i \left(\sum_{core} P_{core,i} \Delta t_i \right) \tag{4.4}$$

where $P_{core,i}$ represents the power draw of an individual core at each time step, i.

4.2 Model Choice

4.2.1 LLMs Profiled

Our study employs several open-source LLMs, summarized in Table 4.1. We vary parameter counts to highlight trade-offs between runtime, energy consumption, and accuracy. These models also represent diverse architectures and training corpora. We subject each model to a series of standardized NLP tasks to evaluate energy consumption during inference.

Table 4.1: LLM Energy Consumption and Runtime

LLM (#Params)	**Disk Storage Size (GB)**	**Min # A100s**	$\boldsymbol{A_K}$**(%)** [5]
Falcon (7B)	14.48	1	44.17
Falcon (40B)	83.66	3	58.07
Llama-2 (7B)	13.48	1	50.97
Llama-2 (13B)	26.03	1	55.69
Llama-2 (70B)	137.98	4	64.52
Mistral (7B)	15.0	1	60.97
Mixtral (8x7B)	93.37	3	68.47

Below, we summarize key aspects of each model and its family/architecture.

Falcon 7B was developed by the Technical Institute of the United Arab Emirates and is a decoder-only model with 7 billion parameters trained on a diverse dataset of 1,500 billion tokens [2]. Its applications span multiple use cases due to its broad training on the RefinedWeb dataset [40]. *Falcon 40B* is the mid-tier version of the Falcon series. It was trained on a similarly curated dataset of nearly five trillion tokens, ensuring diverse linguistic and cultural representation. It is notable for its efficiency, using significantly less resources than comparable models like GPT-3 [6] and Chinchilla AI [19].

Llama-2 (7B) [53] was developed by Meta AI and features 7 billion parameters, making it suitable for a broad range of NLP tasks while being relatively resource-efficient. Optimized for general-purpose applications, it provides solid performance in tasks such as text generation, summarization, and translation. *Llama-2 (13B)* offers enhanced accuracy and a deeper understanding of more complex queries and datasets. This model balances its memory demands, as aforementioned, with performance, making it a good choice for applications requiring more detailed text understanding and generation capabilities. *Llama-2 (70B)* is an advanced model designed for complex problem-solving. This model performs better than smaller models in specialized tasks, such as medical diagnosis from textual data, legal document analysis, and emulating conversations. Due to its huge storage size and network density, it requires significant computational resources, including at least 4×NVIDIA A100 40 GB GPUs.

Mistral 7B [23] was developed by Mistral AI, a company developing small, efficient, open-source models. It was engineered for both high performance and efficiency. It utilizes innovative attention mechanisms like grouped-query attention and sliding window attention to handle long sequences efficiently and reduce memory requirements during inference. This small model outperforms larger models across various NLP benchmarks, particularly in reasoning, mathematics, and code generation tasks. *Mixtral (8x7B)* [24] is a Sparse Mixture of Experts (SMoE) model [14] that optimizes computational efficiency by using a subset of parameters per token. It performs well in many languages and complex tasks like mathematics and code generation.

4.2.2 Impact of Key-Value Dictionary Caching on Inference Runtime

Caching key-value (KV) dictionaries is an optimization strategy aimed at reducing inference runtime for similar prompts in Large Language Models (LLMs) [41]. A KV dictionary encapsulates the collection of input and output tokens processed by the model, facilitating quicker decoding in subsequent inferences by reusing previously computed results [54].

LLMs cache a KV pair in the dictionary for each new token during the inference process. By storing this mapping, the model bypasses the need to regenerate them for each token that appears in the subsequent request, accelerating the encoding and decoding process. This is particularly beneficial when queries share similar tokens, as the model can leverage pre-computed KV pairs to expedite output generation.

To establish a baseline for LLM performance devoid of any enhancements, we turn off KV caching. This approach ensures that each inference is handled independently, with the model reconstructing the KV dictionary from scratch for every prompt. Although this method results in longer inference times, it consistently measures the model's raw computational requirements and latency. Such a baseline is important for establishing the performance of a model without the influence of optimizations. By understanding the baseline capabilities of LLMs, we can better identify an upper limit on energy consumption and runtime for our models of choice.

4.3 LLM Inference Performance on Diverse Clusters

4.3.1 Hardware Details of Test Systems

We show our systems we profile in Table 4.2. We consider these systems as they demonstrate three prominent CPU manufacturers and different generations of GPUs.

System Name	CPU	GPU(s) per Node	DRAM per Node	VRAM per GPU
MacBook Pro	10-core M1 Pro	14-core M1 Pro	32 GB	-
Swing AMD+A100	2×64-core AMD EPYC 7742	8×NVIDIA A100	1 TB	40 GB
Palmetto Intel+V100	40-Core Intel Xeon 6148G	2×NVIDIA V100	376 GB	16 GB

Table 4.2: Our System Configurations

We note that for the M1-Pro, we do not show Falcon (7B) results as its inference was nearly 10× slower than Llama-2 (7B) and Mistral (7B) on the same system.

Across all of our systems, we use a standard Conda environment with Python 3.12.0. We utilize PyTorch v2.2.1, Transformers v4.39.1, Tokenizers v0.15.2, Numpy v1.26.4, HuggingFace v0.22.1, and Accelerate v0.28.0.

For the Swing AMD+A100 system, we specifically use CUDA v11.4.0, AMD-μProf v4.1.124, PyJoules v0.5.1.

4.4 LLM Inference Performance

4.4.1 Experimental Strategy

We conduct a series of experiments to evaluate the performance of different system configurations across various models, systematically varying the number of input and output tokens to measure the effect on runtime and energy consumption under two main experimental conditions.

For the first experimental condition, we execute inference requests with increasing input token counts, ranging from 8 to 2048 tokens, while maintaining a fixed output token count of 32. This setup allows us to isolate the impact of the number of input tokens on the system's performance and energy efficiency.

In the second set of experiments, we vary the output token limit from 8 to 4096 tokens, keeping the input token count constant at 32. This approach helps us understand how increasing the number of output tokens affects the runtime and energy consumption of the systems tested.

We conduct each experiment in a randomized order to mitigate any potential bias the testing sequence introduces. To ensure the reliability of our results, we repeat each configuration until either (1) the measured runtime is within 0.5 seconds of the actual mean runtime with 95% confidence or (2) we conduct 25 trials for each setting.

4.4.2 Input Token Analysis

Here, we present the impact on runtime, energy consumption per token, and throughput for LLMs across different hardware configurations while varying the number of input tokens. We perform these experiments using the suite of systems outlined in Table 4.2 with the models outlined in Section 4.2. In our experiments on the Palmetto Intel+V100 system, the V100 GPU had an out-of-memory error beyond 1024 output tokens for Falcon (7B).

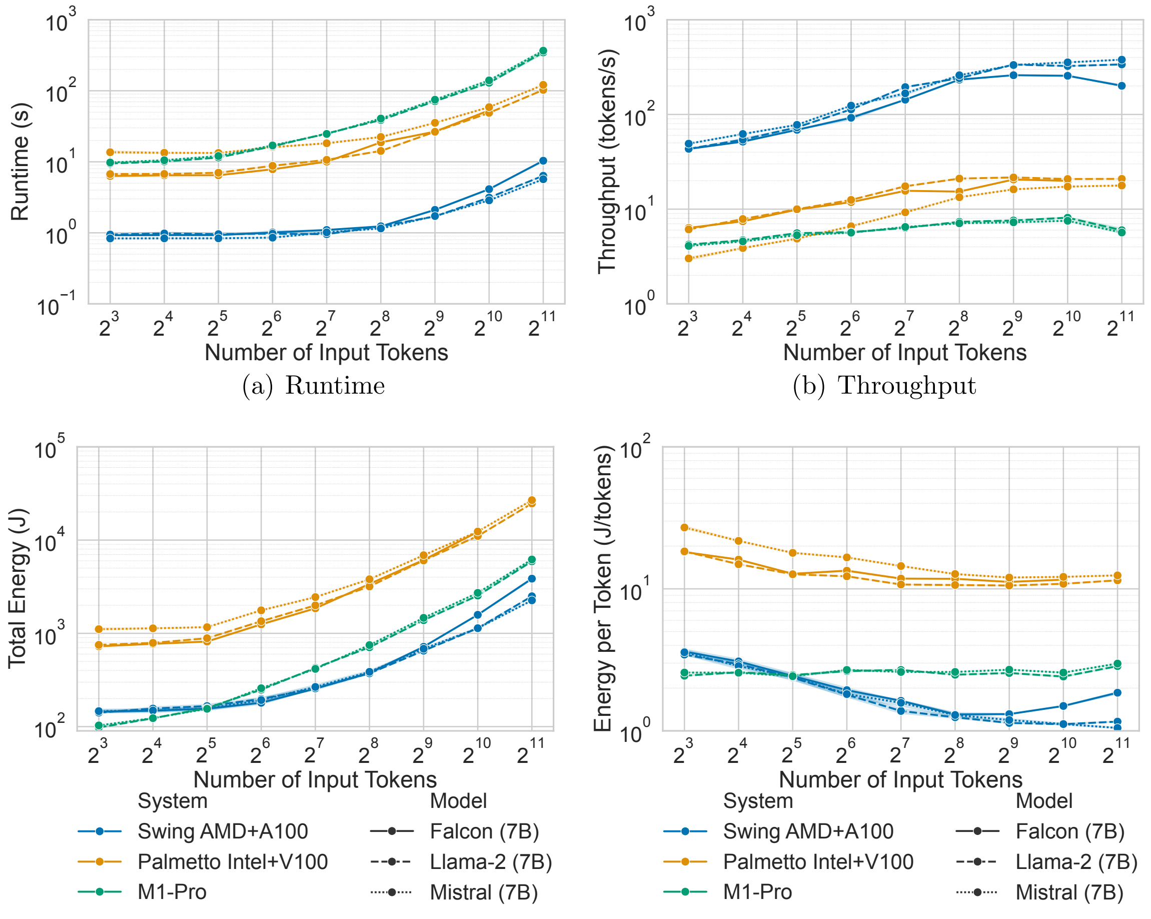
Our runtime measurements show a significant increase as input tokens grow. As depicted in Figure 4.1(a), all systems exhibit a nonlinear escalation in runtime with increasing token counts, with the M1-Pro system showing the most significant magnitude. This trend highlights the computational burden imposed by larger input token counts, particularly on smaller systems that are not as well designed to handle extensive workloads.

For all systems, we notice that throughput follows a "roofline model" with increasing input tokens [56]. Figure 4.1(b) illustrates these dynamics, indicating an increase in throughput for all systems until a certain point where inference becomes bound by compute and not by the overhead of the software, as described by roofline performance models [56].

Energy efficiency varies markedly across different systems. The M1-Pro demonstrates consistently low energy consumption per token, particularly for smaller input token counts, as shown in Figure 4.1(d). This efficiency reflects the M1-Pro's design optimization for low-power operations. In contrast, the Swing AMD+A100, while capable of handling more significant token inputs more efficiently, consumed more energy per token for small workloads yet became more energy efficient at larger input token counts, underscoring a trade-off between workload size and energy efficiency.

4.4.3 Output Token Analysis

Here, we examine the performance trends associated with increasing the number of output tokens for our LLMs and systems of interest, specifically focusing on runtime, energy consumption per token, and throughput. In our experiments, the M1-Pro could not generate more than 512 output tokens without taking more than 6 hours per prompt. For the Palmetto Intel+V100 system, the V100 GPU had an OOM error beyond 1024 output tokens for Falcon (7B) and all models beyond 2048 tokens.



(a) Runtime (b) Throughput (c) Total Energy (d) Energy per Token

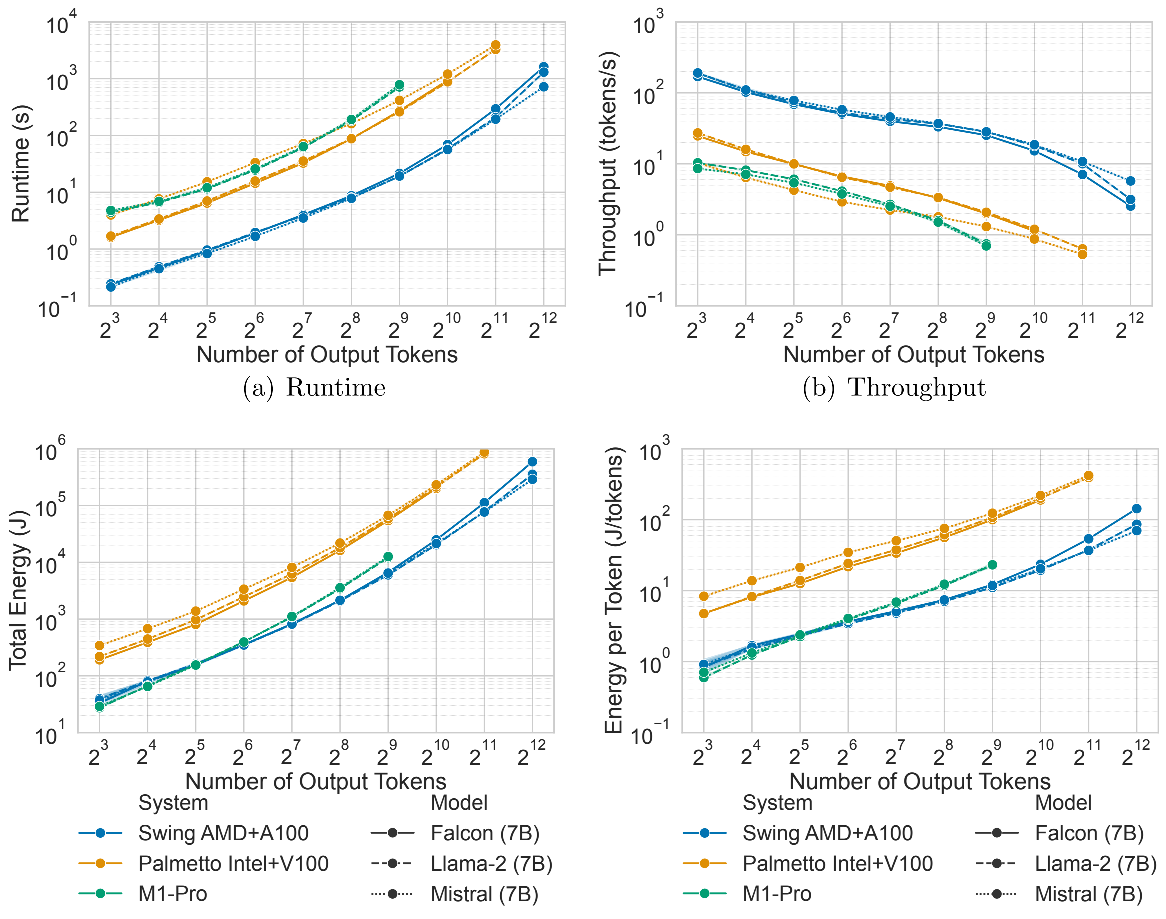
Figure 4.1: Performance of Various Systems and Models for Processing Variable Input Tokens. Error bars are too small to be visible due to the low variance in the data.

Runtime significantly increases with the number of output tokens across all systems. As illustrated in Figure 4.2(a), there is a large rise in runtime. This increase indicates the substantial computational x LLMs require to generate successive output tokens.

In Figure 4.2(b), we observe a decrease in throughput across all systems as the number of output tokens increases. This trend highlights the inherent computational complexity in generating larger sequences of tokens in LLM tasks. As the output token count grows, the system must reprocess each additional token, recalculating the context and updating internal model states [54]. This effect increases the total computation per query and leads to greater processing time per token, consequently lowering the overall throughput.

As expected, we notice that in Figure 4.2(c), there is an increasing amount of energy consumed for the number of output tokens. This trend is roughly the same as that expressed in Figure 4.2(a), as also expected from the relationship between energy and power.

Energy consumption per token also shows an increasing trend as output tokens grow. We notice this trend in Figure 4.2(d), underscoring the energy-intensive nature of producing larger outputs. While generally more energy-efficient, systems such as the M1-Pro begin to consume more energy per token as output demands increase, reflecting the intensive processing involved in output generation.



(a) Runtime (b) Throughput (c) Total Energy (d) Energy per Token

Figure 4.2: Performance of Various Systems and Models for Processing Variable Output Token. Missing data points in M1-Pro and Palmetto Intel+V100 are due to CUDA out-of-memory errors. Due to the low variance in the data, error bars are too small to be visible.

4.4.4 Comparing the Input and Output Analyses

When comparing Figure 4.1(a) and Figure 4.2(a), we observe that increases in the number of output tokens result in a more considerable increase in runtime than increases in input token counts. Similarly, the total energy consumed, as shown in Figure 4.1(c) and Figure 4.2(c), follows the same trend for the number of input versus output tokens. The computational complexity of processing input tokens primarily involves encoding the input context, which occurs once per input sequence and is semi-linear. In contrast, generating output tokens is inherently iterative. Each new output token requires the model to run through all its layers to predict the next token based on an ever-expanding context, which includes both the initial input and all previously generated output tokens [54]. This ongoing computation involves recalculating attention across an increasing number of tokens, updating hidden states, and generating a probability distribution over the vocabulary for each new token. Consequently, as the number of output tokens grows, the computational load increases significantly, leading to more significant runtime increases than processing input tokens.

The impacts on runtime also translate to throughput, depicted in Figure 4.1(b) and Figure 4.2(b). There is a noticeable decline in throughput as output tokens increase, more so than input tokens. The decrease in throughput for output tokens is primarily due to the heightened computational requirements for generating subsequent tokens, where each token's generation slows down as the sequence lengthens. Furthermore, our analysis shows that the energy per token increases as output tokens grow. The energy required to generate each output token becomes significant due to longer passes through the transformer network. We contrast this with the energy consumption when processing input tokens, which increases more slowly.

4.5 Impacts of Model Size

A key question is how runtime and energy consumption scales with increasing parameter counts. Larger models[1] are often more accurate and have higher quality answers to given queries, as shown by their performance on widely-cited tests like the MMLU [18] and HellaSwag [59].

We conduct a series of experiments to evaluate the performance of differing workloads across models of increasing parameter counts. We perform the same test as mentioned in Section 4.4.1 to vary the number of input and output tokens to measure their effects on runtime and energy consumption. Also, we fix the batch size at 32. Due to larger models' memory constraints, we only test on the Argonne Swing AMD+A100 system.

4.5.1 Input Token Analysis

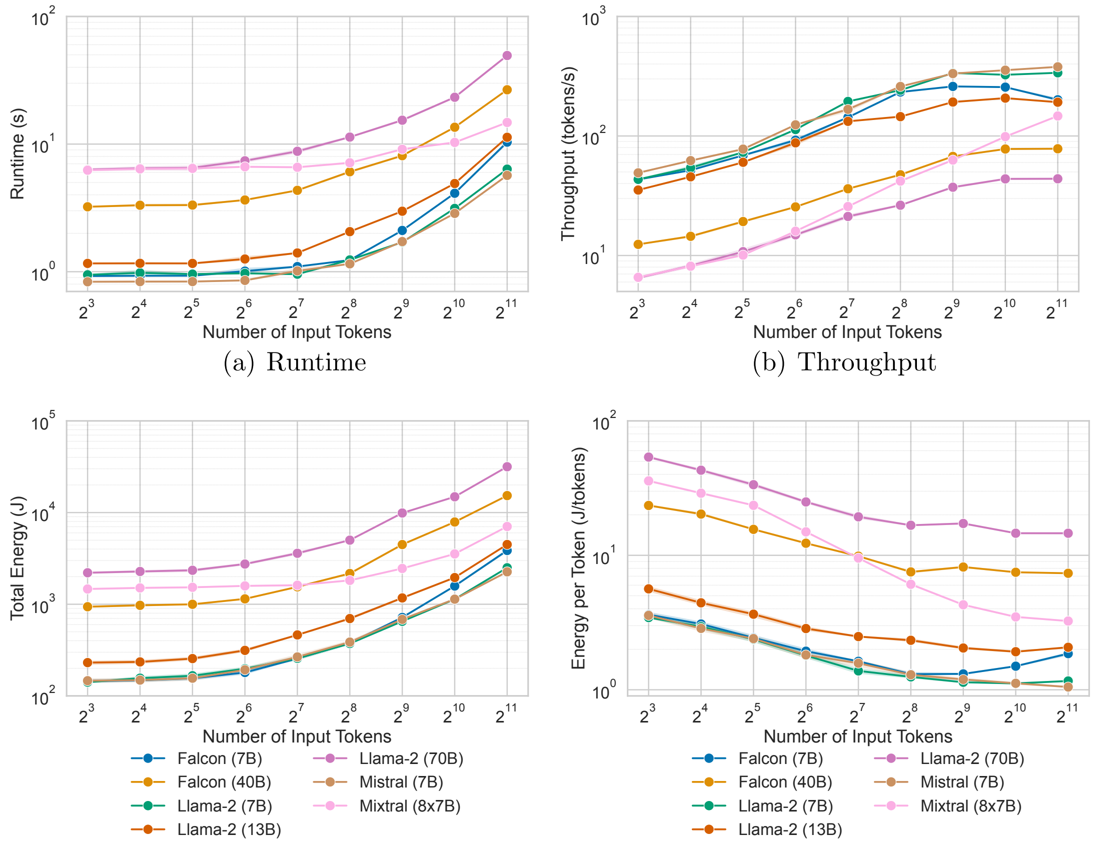
Figure 4.1 presents the impact of varying numbers of input tokens on the runtime, throughput, and energy per token for various LLMs. The results depict a clear trend: as the number of input tokens increases, the runtime tends to increase, while the throughput in Figure 4.3(b) plateaus, following a "roofline" model [56], like for the smaller models. Specifically, the runtime increase is most dramatic for larger models like Llama-2 (70B) and Falcon (40B), likely due to the higher computational burden these models sustain as they process more extensive input sequences. The energy consumption per token demonstrates similar trends, with smaller models exhibiting lower energy per token than larger models.

Figure 4.3(c) shows that a model will consume more total energy for a higher parameter count. This effect worsens energy efficiency in Figure 4.3(d), as smaller models consume less energy for the same amount of work. This finding is unsurprising, as smaller and less

[1]By larger, we refer to models that are greater than 7B parameters, as these were the smallest models we consider in our testing.

dense models should perform much faster but with lower accuracy, as shown in Table 4.1. Therefore, system operators and users must consider this trade-off when choosing models and a relationship we will explore when solving Equation 3.5.

An outlier to all of these cases is Mixtral (8x7B), which has a higher throughput and energy efficiency than other large models at larger input tokens. This LLM's sparse mixture-of-experts architecture (SMoE) [24, 44] allows it to activate only 12B parameters on average by selecting two expert sub-models. This selection phase has a runtime and energy overhead for classifying the prompt, apparent in Figures 4.3(a) and 4.3(c), which the model overcomes for larger workloads. Therefore, for SMoE, one gets the accuracy advantages of a large model with less energy and lower runtime than the denser counterparts.



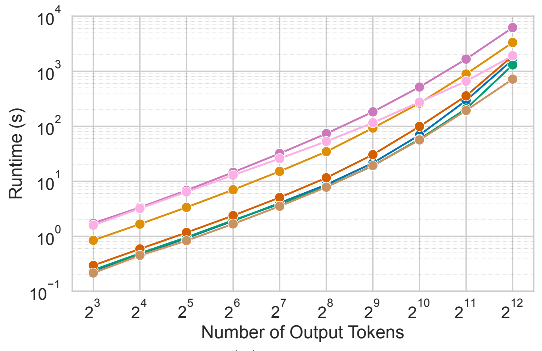
(a) Runtime (b) Throughput (c) Total Energy (d) Energy per Token

Figure 4.3: Performance of Different LLM Parameter Counts for Processing Variable Input Tokens

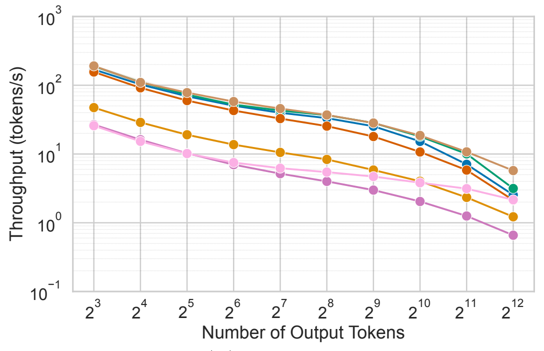
4.5.2 Output Token Analysis

Figure 4.4 illustrates how changes in the number of output tokens affect runtime, throughput, and energy consumption per token across different LLMs. Notably, the runtime exhibits a steep increase with larger output token counts, consistent across all models but especially significant for high-parameter models such as Falcon (40B) and Llama-2 (70B). Throughput decreases as the number of output tokens increases. This inverse relationship highlights the time required to generate each additional token, which involves more extensive interaction between model layers and successive passes through the LLM to generate each token [54]. Energy per token also increases with the number of output tokens and parameters. This increase is particularly sharp in larger models like Falcon (40B).

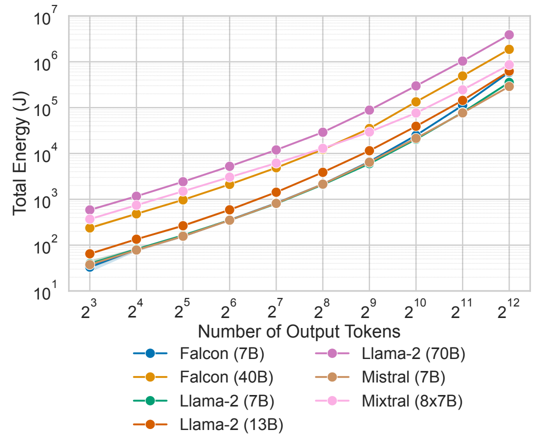
Again, Mixtral (8x7B) demonstrates greater energy efficiency than its large parameter counterparts. Therefore, even in high output token generation cases, an SMoE architecture can yield improvements in energy efficiency.



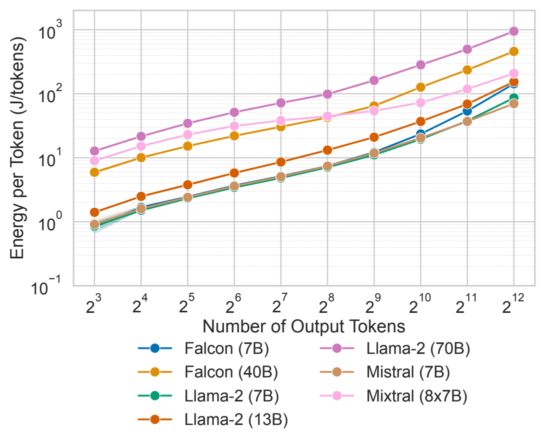
(a) Runtime



(b) Throughput



(c) Total Energy



(d) Energy per Token

Figure 4.4: Performance of Different LLM Parameter Counts for Processing Variable Output Tokens

4.5.3 Comparing the Input and Output Token Analysis

Comparing Figures 4.3(a) and 4.4(a), we observe that increasing the number of output tokens results in a more significant increase in runtime than increasing the number of input tokens. Also, larger parameter LLMs require far more runtime than smaller models. The reason is because generating each new output token requires the model to reprocess the entire sequence, which involves recalculating attention weights and updating the context with each additional token. The computations involved in this process must pass through a larger and denser network of nodes as the parameter count increases, leading to a higher runtime and energy consumption.

Figures 4.3(b) and 4.4(b) illustrate that throughput decreases more rapidly as output tokens increase compared to input tokens. This decline is due to the increased time required to generate each successive output token, which reduces the overall token generation rate. However, the throughput for input tokens plateaus after a certain point, aligning where the system becomes compute-bound rather than constrained by input token processing. In general, smaller models have much higher throughput than the larger models.

Figures 4.3(c) and 4.4(c) show that total energy consumption rises significantly with the number of parameters and output tokens, more so than with input tokens. Moreover, Figure 4.4(d) indicates that the energy consumption per token also increases sharply with the number of output tokens and parameters. This trend underscores the higher energy demand associated with generating longer sequences in larger models, as each additional output token requires more computational effort and power, along with the model size, exacerbating this phenomenon.

The analysis highlights a key consideration for system designers and operators. While processing input tokens involves a single pass through the model to encode the context, generating output tokens is more resource-intensive. The number of parameters in a model also can lead to a large increase in energy consumption and runtime.

Chapter 5

Energy-Optimal LLM Serving

5.1 Modeling Energy and Runtime for LLMs

5.1.1 Interdependence of Input and Output Tokens

From observing our results, a logical question is: are the number of input and output tokens independent in their effect on energy consumption and runtime? The following table presents the ANOVA results for assessing the effects of the number of input tokens, the number of output tokens, and their interaction on the total energy consumption and runtime for LLM inference. To collect this data we perform a grid search from 8 to 2048, in increments of powers of two, for the space of input and output tokens to eliminate the bias of holding the input or output size constant. This analysis includes data aggregated across all models in Table 4.1.

Table 5.1: ANOVA Results for LLM Energy Consumption and Runtime.

Metric	Variable	Sum of Squares	*df*	F-value	p-value
Energy (J)	Number of Input Tokens	5.17×10^{10}	8	15.86	3.79×10^{-17}
	Number of Output Tokens	4.13×10^{11}	8	126.63	1.22×10^{-65}
	Interaction	1.18×10^{11}	64	4.53	4.67×10^{-15}
Runtime (s)	Number of Input Tokens	3.43×10^{5}	8	12.97	2.34×10^{-14}
	Number of Output Tokens	2.78×10^{6}	8	104.98	4.56×10^{-60}
	Interaction	8.21×10^{5}	64	3.88	1.92×10^{-12}

The significance of the main effects and the interaction term highlights the combined influence of the input and output tokens on the energy consumption and runtime of LLM inference. The *number of input tokens* and *number of output tokens* substantially impact energy consumption and runtime, with output tokens having a more significant effect size as indicated by the higher F value. Also, the *interaction* term shows that the number of input and output tokens have a coupled effect on energy consumption and runtime. Finally, if we take a significance value of $\alpha = 0.01$, then the high F values and extremely low p-values for these effects confirm their statistical significance and the robustness of these findings across different LLMs.

5.1.2 Formulating Models for Energy and Runtime

Therefore, we can use the results in Table 5.1 to guide the creation of models to predict the energy consumption and runtime of LLMs for use in optimization problems like in Equations 3.1 and 3.5. We know that for accurate models based on the number of input and output tokens, an interaction term needs to be used to combine them.

We propose a model to describe the total energy consumption for a model K on a system S as a function of input and output token counts, τ_{in} and τ_{out}, respectively:

$$e_{K,S}(\tau_{in}, \tau_{out}) = \alpha_{K,S,0}\tau_{in} + \alpha_{K,S,1}\tau_{out} + \alpha_{K,S,2}\tau_{in}\tau_{out}, \tag{5.1}$$

where $\alpha_{K,S,0}, \alpha_{K,S,1}, \alpha_{K,S,2} \in \mathbb{R}$ are parameters unique to each model and system, determined through ordinary least squares regression.

Similarly, we propose the following model to describe the total runtime for a model K on system S as a function of input and output tokens, τ_{in} and τ_{out}, respectively:

$$r_{K,S}(\tau_{in}, \tau_{out}) = \beta_{K,S,0}\tau_{in} + \beta_{K,S,1}\tau_{out} + \beta_{K,S,2}\tau_{in}\tau_{out}, \tag{5.2}$$

where $\beta_{K,S,0}, \beta_{K,S,1}, \beta_{K,S,2} \in \mathbb{R}$ are also unique to each model K and system S.

5.1.3 Fitting Models with Ordinary Least Squares

Using the statsmodel v0.14.2 Python package and its Ordinary Least Squares (OLS) API, we can determine the values of $\alpha_{K,S,i}$ and $\beta_{K,S,j}$ that best fit Equations 5.1 and 5.2 for each LLM, K on a system, S.

Due to the limitations in testing models on systems that were not the Argonne Swing AMD+A100, we only have S for the AMD+A100 system. A summary of the quality of these fits is included in Table 5.3. As we can see, this model has high explainability for the effect of input and output tokens on energy and runtime for inference of these different LLMs.

One of the significant limitations of the models in Table 5.2 is their specific fit for a system and model. Also, due to the compounding nature of the number of input and output tokens, these models do a poor job of extrapolating beyond their dataset. Therefore, models must be fit for a system, model, within the expected workload range.

Table 5.2: OLS Parameters Across Models for S=Swing AMD+A100

LLM (#Params)	**Energy Model** (e_K)			**Runtime Model** (r_K)		
	$\boldsymbol{\alpha_{K,0}}$	$\boldsymbol{\alpha_{K,1}}$	$\boldsymbol{\alpha_{K,2}}$	$\boldsymbol{\beta_{K,0}}$	$\boldsymbol{\beta_{K,1}}$	$\boldsymbol{\beta_{K,2}}$
Falcon (7B)	-8.81	41.28	8.91e-02	-2.45e-02	1.07e-01	2.42e-04
Falcon (40B)	-20.76	197.37	2.35e-01	-3.61e-02	3.62e-01	4.02e-04
Llama-2 (7B)	-3.89	31.52	4.27e-02	-1.00e-02	8.35e-02	1.07e-04
Llama-2 (13B)	-6.79	56.01	7.29e-02	-1.74e-02	1.43e-01	1.85e-04
Llama-2 (70B)	-12.03	414.82	3.15e-01	-3.12e-02	7.03e-01	5.33e-04
Mistral (7B)	-3.12	31.69	3.53e-02	-7.80e-03	8.10e-02	8.10e-02
Mixtral (8x7B)	-8.48	105.54	9.86e-02	-1.34e-02	2.96e-01	1.50e-04

Table 5.3: Summary of OLS Regression Results Across Models

LLM (#Params)	**Energy Model** (e_K)			**Runtime Model** (r_K)		
	$\boldsymbol{R^2}$	**F-statistic**	**P-value**	$\boldsymbol{R^2}$	**F-statistic**	**P-value**
Falcon (7B)	0.964	681.2	2.53e-55	0.962	651.1	1.35e-54
Falcon (40B)	0.972	904.5	1.78e-60	0.976	1073	2.74e-63
Llama-2 (7B)	0.973	942.3	3.76e-61	0.972	1032	1.19e-62
Llama-2 (13B)	0.972	887.8	3.60e-60	0.972	907.0	1.60e-60
Llama-2 (70B)	0.976	1022	6.66e-62	0.980	1230	6.23e-65
Mistral (7B)	0.975	997.0	1.70e-61	0.976	1039	3.62e-62
Mixtral (8x7B)	0.980	1238	4.97e-65	0.992	3139	2.23e-80

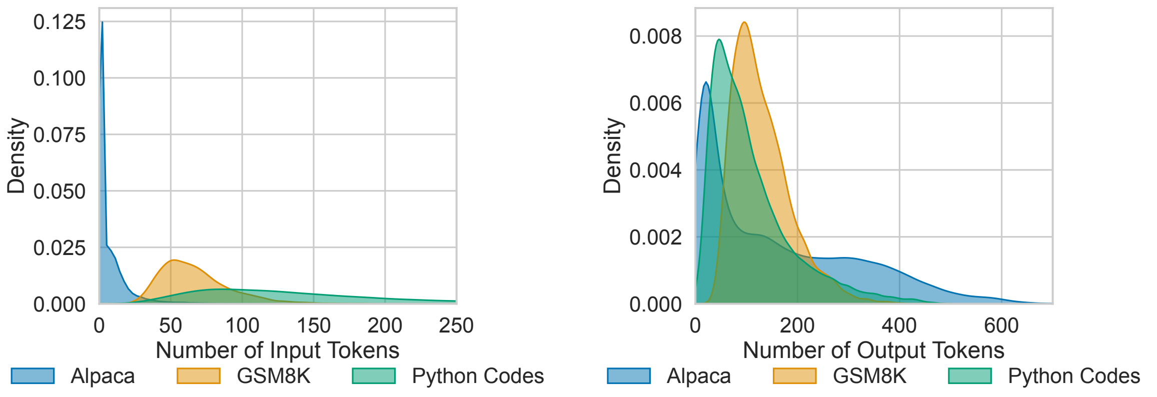
5.2 Energy-Optimal Hybrid Data Center for Serving a Single LLM

Considering the performance results we collect from multiple systems, we notice that there is an energy-optimal way to serve a single LLM with a hybrid data center with a combination of M1 Pros and AMD+A100s. The intuition behind this is that the energy expended per token for the M1 Pro is lower than that of the AMD+A100 up to a certain point in the number of input and output tokens as seen in Figures 4.1(d) and 4.2(d). However, the energy efficiency characteristics are different when varying the number of input and output tokens, and therefore, we will proceed with separate analyses.

For the following section, we consider the specific case of only hosting one LLM, Llama-2 (7B), in a sufficiently large hybrid data center with M1-Pros and A100s. Then, we have some workload, Q, for an LLM, which is a set of input tokens with known output tokens.

5.2.1 Our Workload and Datasets

To capture realistic LLM workloads, we utilize fine-tuning datasets from HuggingFace, including a large set of input queries with their associated answers. Fine-tuning datasets utilize synthetic questions and answers from large models like GPT-4 to refine the weights and parameters of smaller models [7]. We analyze the token distribution in prompts from the Alpaca [51], GSM8K [9], and Python Codes 25K [15] dataset, a benchmark dataset frequently used in model fine-tuning. Alpaca comprises 52002 prompts, offering a diverse range of lengths akin to a typical workload in systems like GPT-4 [38]. GSM8k is a collection of 8792 prompts of grade-school math problems and their answers from GPT-4. Python Codes 25K comprises 24813 programming task prompts with examples of Python solutions. The distribution of input and output tokens is a proxy for understanding the variegated nature of LLM workloads.



(a) Input Tokens (b) Output Tokens

Figure 5.1: Kernel Density Estimation of Token Count Distribution for Alpaca [51], GSM8K [9], and Python Codes [15]

5.2.2 A Threshold-Based Solution

Section 3.1 presents a formulation for dividing a workload among multiple systems to minimize energy consumption and runtime. This problem is a non-trivial GAP to solve. However, due to the high runtime and poor energy efficiency performance of the Palmetto Intel+V100 system, we only consider a data center with M1-Pro and AMD+A100, such that $\mathcal{S} = \{M1, A100\}$. With only two systems, we can consider a workload assignment solution strictly based on a threshold for the number of input tokens, output tokens, or the product of these. Since we are only using one LLM and two systems, we drop the K index and proceed with just e_{M1} and e_{A100}.

Formulating Our Threshold Approach

In this solution, we take a cutoff threshold, τ_{in}^*, for input token count. Our solution dictates that queries with $\tau_{in} \leq \tau_{in}^*$ tokens are processed on M1-Pro accelerators, which are energy efficient while handling smaller token workloads. Conversely, queries with $\tau_{in} > \tau_{in}^*$ tokens leverage the greater computational ability of A100 GPUs, which offer greater energy efficiency for larger tasks. Below, we use a unit-step function, such that $u(\tau) = 1$ if $\tau \geq 0$ and 0 if $\tau < 0$.

The energy component of our cost function, split over the input token threshold, is as follows:

$$E_{Total,in}(Q, \tau_{in}^*) = \sum_{(\tau_{in},\tau_{out})\in Q} u(\tau_{in}^* - \tau_{in})e_{M1}(\tau_{in}, \tau_{out}) + (1 - u(\tau_{in}^* - \tau_{in}))e_{A100}(\tau_{in}, \tau_{out}), \tag{5.3}$$

where $E_{Total,in}$ represents the total energy consumption for a given dataset of input lengths τ_{in}^* and $e_{M1}(\tau_{in}, \tau_{out})$ and $e_{A100}(\tau_{in}, \tau_{out})$ denote the energy consumption for varying the input token size for the M1-Pro and A100 systems, respectively.

Similarly, we can formulate a solution that depends only on a threshold τ_{out}^* for the number of output tokens. This method is the same concept as for the input tokens, except this time, we have different values in the unit-step function for the τ_{out} output tokens. We revise our performance model as follows:

$$E_{Total,out}(Q, \tau_{out}^*) = \sum_{(\tau_{in},\tau_{out})\in Q} u(\tau_{out}^* - \tau_{out})e_{M1}(\tau_{in}, \tau_{out}) + (1 - u(\tau_{out}^* - \tau_{out}))e_{A100}(\tau_{in}, \tau_{out}), \tag{5.4}$$

Considering that the formulation of our energy consumption models in Equation 5.1 includes a product interaction effect, we can also perform a product threshold, where we consider a threshold for $\tau_{in}\tau_{out}$. Here, we update our step function to include a product of $\tau_{in}^*\tau_{out}^*$. Then, our total energy consumption model becomes

$$\begin{aligned} E_{Total,prod}(Q, \tau_{in}^*\tau_{out}^*) = \sum_{(\tau_{in},\tau_{out})\in Q} & u(\tau_{in}^*\tau_{out}^* - \tau_{in}\tau_{out})e_{M1}(\tau_{in}, \tau_{out}) \\ & + (1 - u(\tau_{in}^*\tau_{out}^* - \tau_{in}\tau_{out}))e_{A100}(\tau_{in}, \tau_{out}), \end{aligned} \tag{5.5}$$

The models for total runtime follow the same formulation as Equations 5.3, 5.4, and 5.5 exchanging r_{M1} and r_{A100} for e_{M1} and e_{A100}.

5.2.3 Simulation Results

Using these models, we include plots that show the total energy consumption and runtime for our three different datasets in Figures 5.2 and 5.3, respectively. The method color refers to whether we are changing the input, output, or product threshold or using just one chip type. To keep all the methods on the same plot, we take the square root of the product threshold to align it on the same scale as the other methods. We use the line style for "System Type" to denote whether we are considering "Hybrid" for a method using both kinds of hardware or "Full" for just one.

Due to the M1-Pro's inability to generate more than 512 output tokens, we have only considered input and output thresholds until now.

Figure 5.2 illustrates the energy consumption of a hybrid data center as we shift the thresholds for input tokens (τ_{in}^*), output tokens (τ_{out}^*), and their product ($\sqrt{\tau_{in}^*\tau_{out}^*}$). The results demonstrate that different threshold methods have distinct impacts on energy

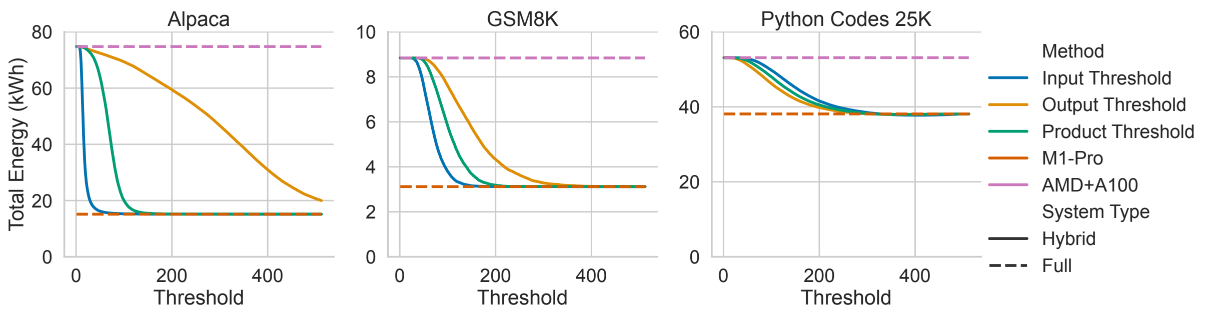


Figure 5.2: Energy Consumption for Shifting the Threshold of τ_{in}^*, τ_{out}^*, and $\sqrt{\tau_{in}^*\tau_{out}^*}$.

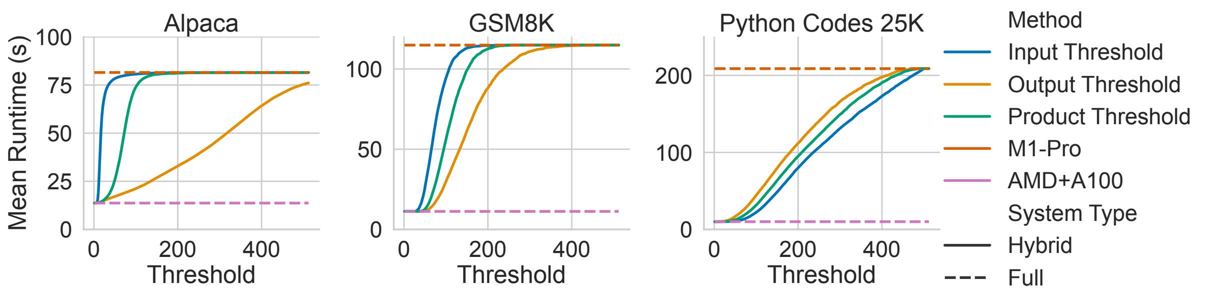


Figure 5.3: Mean Runtime per Query for Shifting the Threshold of τ_{in}^*, τ_{out}^*, and $\sqrt{\tau_{in}^*\tau_{out}^*}$.

consumption across the datasets.

Using the number of input tokens as a threshold results in a varied pattern of energy consumption reduction for all datasets depending on the distribution of the tokens in the dataset. For instance, with the Alpaca dataset, energy consumption decreases rapidly as the threshold increases due to most of the queries having few input tokens, as shown in Figure 5.1. The output threshold works oppositely to the input threshold. For example, in the Alpaca case, since output tokens have a more even distribution, shifting the output token threshold results in a more gradual change in energy consumption. The product method combines input and output token sizes, providing a more balanced approach. The energy consumption trends follow a middle path between the input and output threshold methods, offering a compromise that leverages the strengths of both M1-Pro and A100 systems.

Figure 5.3 shows the runtime implications of shifting the thresholds for input tokens, output tokens, and their product. As the input threshold increases, the mean runtime increases across the board due to the M1-Pro being nearly 6× slower than the A100. As with energy consumption, the question is how quickly this threshold moves tasks to the other chip type. This result hinges entirely on the distribution of input and output tokens. The runtime trend for the output threshold method follows a similar pattern to the input threshold but with smoother increases. The product threshold method results in a more stable runtime pattern. Considering both input and output token sizes, this method effectively balances the workload distribution, falling between the input and output threshold curves. We find that out of the tested solutions, the product threshold method offers a balanced approach for managing runtime, combining the efficiency of M1-Pro with the computational capabilities of A100.

5.2.4 Balancing Energy Efficiency and Runtime Performance

Our analysis of both input and output token processing within a hybrid, heterogeneous data center framework has led to the identification that depending on the distribution and number of input and output tokens, we can strategically allocate tasks to M1 Pro systems or A100 GPUs to optimize energy efficiency.

Shifting the token distribution leverages the M1 Pro's superior energy efficiency for input and output tasks up to the threshold, beyond which we utilize the A100's computational power. This policy saves energy as the more energy-efficient M1-Pro handles tasks with fewer tokens for outputs up to the threshold. However, this energy optimization comes at the expense of increased runtime, which is particularly noticeable in output token generation where the M1 Pro, despite its efficiency, can not match the A100's speed.

The energy-runtime trade-off presents a favorable scenario for applications with low runtime sensitivity. For instance, batch processing of LLM tasks, such as overnight data analyses or non-time-critical computations, can benefit significantly from this energy-efficient configuration.

5.3 Offline Query Routing to Multiple LLMs

If we consider a data center with multiple LLMs and only one kind of CPU+GPU (e.g., AMD+A100), we can consider routing workloads to different LLMs. The motivation for this technique is that there is a trade-off in attempting to maximize the average accuracy of responses from an ensemble of LLMs and minimize the energy and runtime expended. We hypothesize that we can find an optimal workload division of queries across different LLMs offline by using models that deterministically estimate energy and runtime.

5.3.1 Representing Workload Routing as an LP Problem

Using our runtime and energy consumption models from Table 5.2, we can represent the problem outlined in Section 3.2.2 for workload-aware routing problems in an LP solver. Using PuLP (v.2.8.0), a Python package designed for solving optimization problems like that we formulate in Equation 3.5, we encode our query workload of input and output tokens with a set of binary variables that indicate which model will process that pair of tokens. Then, we convert the given constraints in Equation 3.6–3.8 using this format to route our workload to different models effectively.

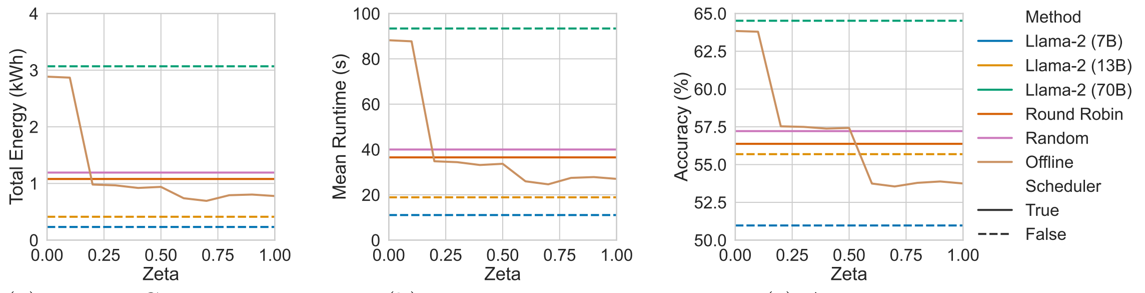
For this example, we consider a data center serving the three Llama-2 models of 7B, 13B, and 70B parameters. Assume that our set $\mathcal{K} = \{1, 2, 3\}$ enumerates those models, respectively. A parameter that affects our optimization problem significantly is the data center partition γ_i. In our evaluation, we choose $\gamma_1 = 0.05, \gamma_2 = 0.2$, and $\gamma_3 = 0.75$. We assume that each model has enough GPUs and instances so that we can assign each query to an LLM without delay.

As we show in Table 4.1 and Figures 4.1 and 4.2, an LLM with a larger parameter count has greater accuracy but also greater runtime and energy consumption for each input and output token. It is reasonable to host differently sized models to allow us to serve inference requests more runtime and energy efficiently with a trade-off of slightly lower accuracy.

With this, we can use the models for energy and runtime in Equations 5.1 and 5.2 that we find for each LLM from OLS regression and our function to capture accuracy from Equation 3.4 to calculate the costs associated with Equation 3.5. For our sample workload, we use a subset of 200 queries from the Alpaca dataset [51] due to the NP-hard complexity of solving this problem, as described in Section 5.5.

5.3.2 Results of Offline Routing

Figure 5.4 shows the trade-offs in energy consumption, mean runtime, and accuracy by varying the operational parameter ζ while routing queries to different models. We compare this hybrid approach to the runtime, accuracy, and energy consumption of three alternatives: using only one model to process all the queries, round-robin routing, and random routing. We use the line style, "Scheduler" to denote whether we are routing using some kind of scheduling algorithm or just sending all queries to one kind of LLM.



(a) Energy Consumption (b) Mean Runtime (c) Accuracy

Figure 5.4: Varying Operational Parameter ζ for Offline Simulation. Due to very similar allocations of queries, the round-robin and random routing lines are indistinguishable

In Figure 5.4(a), as ζ increases from 0 to 1.0, there is a noticeable trend in energy consumption across different simulation scenarios. When ζ is low, energy consumption is high because the system prioritizes accuracy over energy efficiency. This trend is evident in the figures showing energy consumption, where energy usage decreases as ζ increases, indicating that higher ζ values lead to more energy-efficient routing decisions, sacrificing accuracy for energy savings. Similarly, Figure 5.4(b) shows that the mean runtime per query decreases with increasing ζ. A low ζ value results in longer runtimes as the system routes queries to models that provide higher accuracy but are less efficient in time and energy. Conversely, higher ζ values result in shorter runtimes, as the system favors more energy and time-efficient models over the most accurate ones. Figure 5.4(c) demonstrates the accuracy-cost trade-off, with incrementally more accuracy requiring significant increases in runtime and energy consumption.

Our solution allows data center operators to navigate the trade-off space using the parameter ζ. For example, they could provide higher accuracy when there is an energy surplus or lower energy prices and deliver faster and lower energy responses during peak loads, though with slightly diminished accuracy. This flexibility is important for adapting to different operational scenarios. Overall, our proposed offline routing algorithm demonstrates its capability to optimize energy consumption and runtime while maintaining acceptable levels of accuracy, proving its effectiveness in managing workloads in a multi-LLM data center environment.

5.4 Online Query Routing to Multiple LLMs

As we have presented our results for the offline routing problem, we now consider the case of real-time data center routing queries to different LLMs as formulated in Algorithm 1.

Here, we present the energy, accuracy, waiting time, and runtime results for considering a spectrum of ζ values for our cost function in Algorithm 1. We also show the effects of incorporating a queue busyness penalty like in Algorithm 2. Finally, we demonstrate the impact of varying the arrival rate λ on our quantities of interest.

5.4.1 Simulation Model

In the following simulation, we consider a data center with $N_{system} = 16000$ A100 GPUs hosting Llama-2 7B, 13B, and 70B instances. We choose this amount to match the Meta system mentioned in this article [46]. The occupancy values for these models are $\gamma_1 = 0.05, \gamma_2 = 0.2, \gamma_3 = 0.75$, respectively. As a reminder, using n_K as the minimum GPUs required for each LLM from Table 4.1, there are $\left\lfloor \frac{\gamma_K N_{system}}{n_K} \right\rfloor$ instances of each LLM K within our system. In this case, there are 800 instances of Llama-2 (7B), 3200 instances of Llama-2 (13B), and 3000 instances of Llama-2 (70B).

For our workload, $Q(t)$, we again use the Alpaca dataset [51]. We assume all queries arrive in a time window $[0, T]$. We model these arrivals as a Poisson process with rate $\lambda = |Q(T)|/T$. After a query is routed to an LLM, it is placed in an unbounded first come, first served M/D/1 queue to wait until an instance is available. Once an instance of the LLM is available, the query is popped from the queue to that instance, where we deterministically estimate its runtime via our runtime model, r_K.

The M/D/1 queue model allows us to predict each query's average waiting time and system time. The average waiting time W_q in an M/D/1 queue is given by the formula [25]:

$$W_{q,K} = \frac{\lambda \sigma_K^2}{2(1 - \lambda \sigma_K)},$$

where σ_K is the mean model runtime for a query for a model K. We can approximate σ_K by taking a mean of r_K values for a sample set of queries Q' such that $\sigma_K \approx \frac{1}{|Q'|} \sum_{(\tau_{in},\tau_{out}) \in Q'} r_K(\tau_{in}, \tau_{out})$. Given that the runtime inference takes is deterministic, and the arrival process is Poisson, the *service time* for a generic query, $W_K(\tau_{in}, \tau_{out})$ (total time a query spends in the system, including waiting time) is:

$$W_K(\tau_{in}, \tau_{out}) = W_{q,K} + r_K(\tau_{in}, \tau_{out}).$$

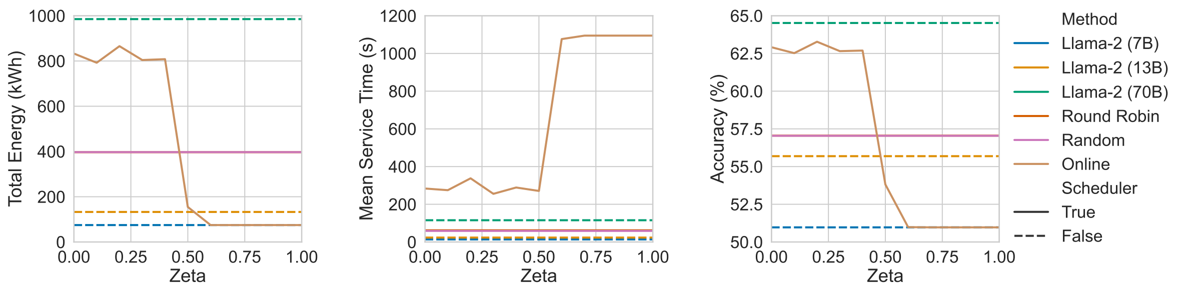
As we will show, we can lower the mean service time using a penalty associated with routing to busy queues, as shown in Algorithm 2. Also, the value of λ can affect the routing outcomes. Therefore, we will explore the impacts of this parameter on energy, runtime, accuracy, and mean query wait time.

5.4.2 Results for Varying Operational Parameter, ζ

As the operational parameter ζ increases from 0 to 1.0, there is a noticeable trend in energy consumption, mean service time, and accuracy. For these simulations, we assume all 52002 arrivals occur within 10000 seconds, meaning that $\lambda = 5.2$ queries/s. As in the offline case, we use the line style, "Scheduler" to denote whether we are routing using some kind of scheduling algorithm or just sending all queries to one kind of LLM. We note that we do not incorporate any wait time for results without a scheduler. Also, we assume the energy of a query residing in a queue is negligible.

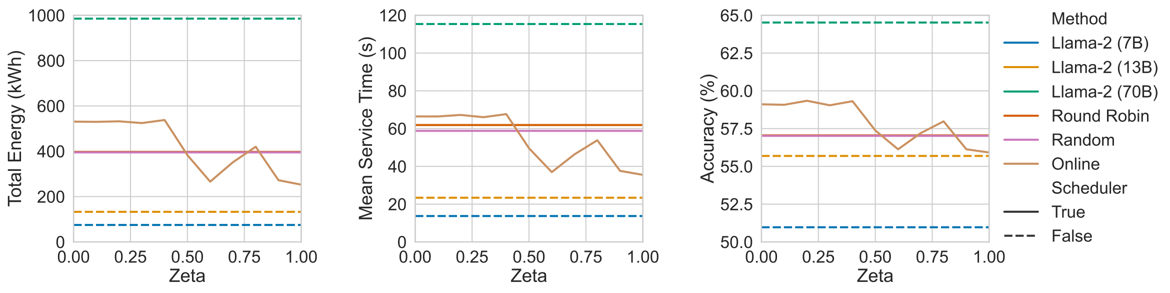
When ζ is low, energy consumption and accuracy are high, as the system prioritizes accuracy over energy efficiency. This effect is evident in the figures showing energy consumption (Figures 5.5(a) and 5.6(a)) where energy usage decreases as ζ increases, indicating that higher ζ values lead to more energy-efficient routing decisions, giving up some accuracy for energy savings. However, the mean service time increases with increasing ζ for no queue awareness, while decreasing when we introduce queue awareness (Figures 5.5(b) and 5.6(b)). This effect is due to the dominating behavior that waiting can have when we do not consider the current utilization of the models. Without considering the length of queues, services cannot be as responsive to requests, particularly when prioritizing energy efficiency. Accuracy shows an inverse relationship with ζ (Figures 5.5(c) and 5.6(c)). In general, the queue awareness maintains a lower range of energy, runtime, and accuracy values with changes in ζ.

As discussed with mean service time, the average wait time also varies with ζ. Without queue awareness, the wait time is significantly higher than the round-robin and random



(a) Energy Consumption (b) Mean Service Time (c) Accuracy

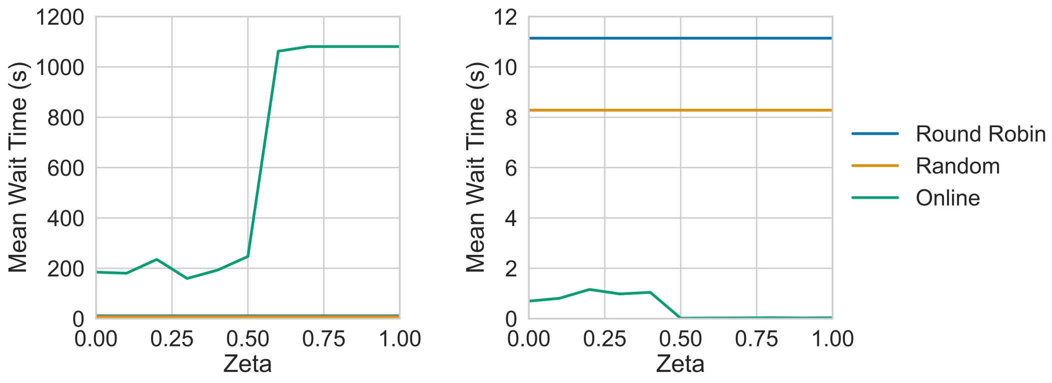
Figure 5.5: Varying Operational Parameter ζ for Online Simulation without Queue Awareness ($\lambda = 5.2$ queries/s). Due to very similar query allocations, the round-robin and random routing lines are indistinguishable.



(a) Energy Consumption (b) Mean Service Time (c) Accuracy

Figure 5.6: Varying Operational Parameter ζ for Online Simulation with Queue Awareness ($\lambda = 5.2$ queries/s). Due to very similar query allocations, the round-robin and random routing lines are indistinguishable.

routing values, as shown in Figure 5.7(b). Incorporating queue awareness reduces the average wait time by nearly three orders of magnitude, especially for higher ζ values. By including a penalty based on how busy a queue is, we can distribute the workload more evenly across models and avoid overloading any single model. A drawback of this method is that we have to operate within a smaller range of accuracy and energy consumption, yet in cases where interactive support is necessary, then this is an important factor.



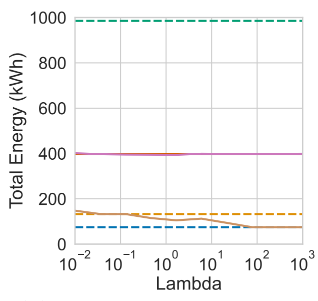
(a) No Queue Awareness (b) Queue Awareness

Figure 5.7: Impacts of Varying Operational Parameter ζ for Mean Wait Time for Queries ($\lambda = 5.2$ queries/s)

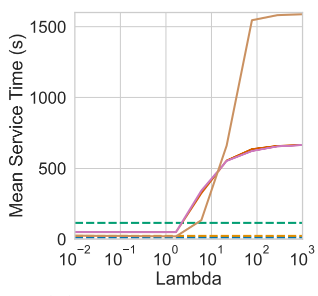
5.4.3 Results for Varying Arrival Rate, λ

A key aspect of testing how robust routing frameworks are is observing how they handle a spike in arrival traffic. In our formulation, λ represents the average number of query arrivals per second. Therefore, in these simulations, we show the effects of increasing the number of queries per second with and without queue awareness.

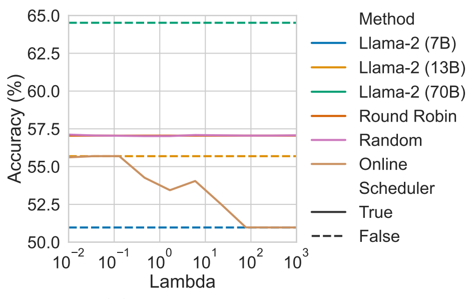
Without queue awareness, energy consumption decreases with the arrival rate λ. However, with queue awareness, there is an increase in energy consumption. Figures 5.8(a) and 5.9(a) illustrate that higher λ values lead to split behavior in energy consumption based on the algorithm chosen. This difference is significant because, without queue awareness, when our system is overloaded, it will route queries to the Llama-2 (7B) model as it can process queries quickly and often with the lowest energy cost. However, when we consider the lengths of the queues, our system distributes the workload more evenly across the instances. Mean service time follows a different pattern from energy consumption in Figures 5.8(b) and 5.9(b). As the arrival rate increases, the query's mean service time increases as the wait time does as well. Accuracy follows the same trend as energy consumption with increasing λ values (Figures 5.8(c) and 5.9(c)).



(a) Energy Consumption

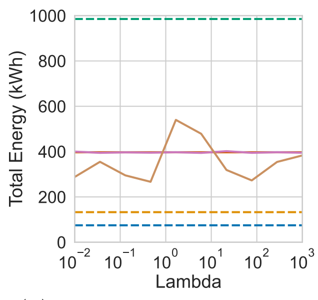


(b) Mean Service Time

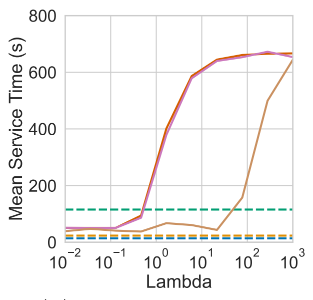


(c) Accuracy

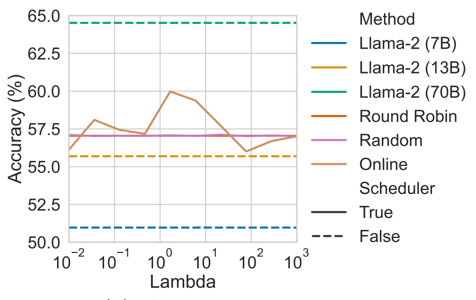
Figure 5.8: Varying Arrival Rate λ for Online Simulation without Queue Awareness ($\zeta = 0.5$). Due to very similar query allocations, the round-robin and random routing lines are indistinguishable.



(a) Energy Consumption



(b) Mean Service Time

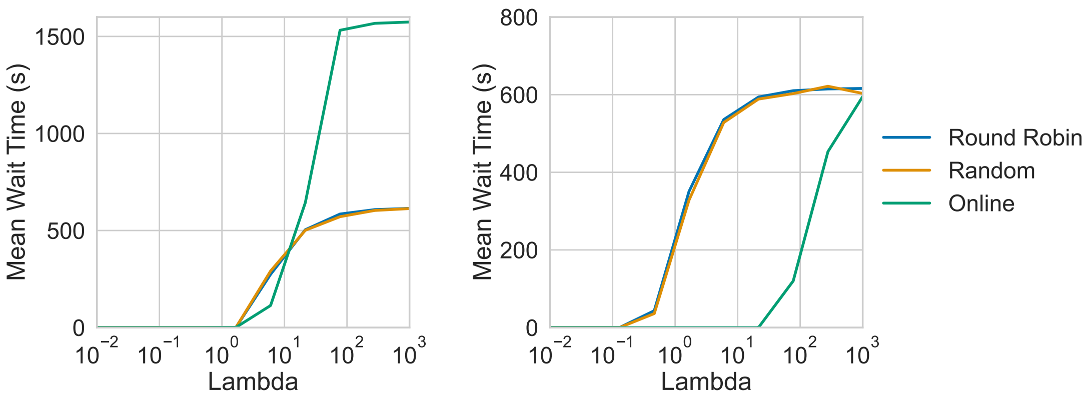


(c) Accuracy

Figure 5.9: Varying Arrival Rate λ for Online Simulation with Queue Awareness ($\zeta = 0.5$). Due to very similar query allocations, the round-robin and random routing lines are indistinguishable.

The average wait time increases significantly with λ, mainly when queue awareness is not considered (Figure 5.10(a)). Queue awareness helps mitigate this effect, as shown in

Figure 5.10(b), where the average wait time for our online method is substantially lower across different λ values. This indicates that incorporating queue length into the routing decision effectively balances the load and reduces wait times even under high query arrival rates. At a specific rate of arrivals, the load needs to be shifted to these models, or wait times will grow very large. Operators can implement elastic scaling for arrivals greater than ten queries/second to mitigate these wait times. However, solutions like this require further analysis due to the added runtime and energy costs of starting new instances.



(a) No Queue Awareness (b) Queue Awareness

Figure 5.10: Impacts of Varying Arrival Rate λ for Mean Query Wait Time ($\zeta = 0.5$)

5.4.4 Discussion of Online Routing Simulations

The results from varying the operational parameter ζ and the arrival rate λ shed light on the performance and efficiency of our proposed online query routing algorithm.

The varying ζ analysis demonstrates our algorithm's capability to balance energy consumption, mean service time, and accuracy according to the desired operational priorities. By adjusting ζ, data center operators can effectively manage trade-offs between energy efficiency and model accuracy. This flexibility is crucial for adapting to different operational scenarios, such as meeting sustainability goals, mitigating peak usage times, or during critical tasks where accuracy is paramount.

On the other hand, the results of varying λ highlight the scalability and robustness of the algorithm under different load conditions. As λ increases, we show that our system can handle higher query volumes without significant degradation in performance. The incorporation of queue awareness significantly enhances this capability by distributing the workload more evenly, thus preventing any single model from becoming a bottleneck. This ensures that the system can maintain lower average wait times and consistent accuracy levels, even as the arrival rate of queries increases.

As we can see, the cost calculation that includes queue awareness greatly enhances our algorithm. The largest improvement comes with improved service time and wait time. A trade-off, however, is queue awareness makes ζ less effective. The routing resides within a tighter range of accuracy and energy, meaning it cannot explore different kinds of routing. Considering the improvement in quality of service with lower wait times, this lack of control is likely beneficial [1, 55].

5.5 Complexity Analysis

Having presented both the online and offline results, we now compare these algorithms regarding their algorithmic complexity, scalability, and runtime efficiency.

5.5.1 Offline Algorithm Complexity

The offline algorithm utilizes an LP approach, with its complexity derived from multiple factors. The algorithm calculates energy and accuracy costs for each query across all models, resulting in a complexity of $O(K \times |Q|)$ for each type of cost calculation, where K is the number of models and $|Q|$ is the number of queries. Setting up the LP involves implementing the constraints from Equations 3.6–3.8. The formulation of these constraints has a complexity of $O(K \times |Q|)$ for assignment constraints and $O(K \times |Q|)$ for workload constraints, resulting in an overall formulation complexity of $O(K \times |Q|)$.

Solving the LP can be computationally intensive, mainly since the problem resembles a multidimensional multi-choice knapsack problem (MMKP), which is known to be NP-hard [43]. The Coin-Or Branch and Cut (CBC) [16] solver used by PuLP employs branch-and-bound and branch-and-cut algorithms to solve such problems. The worst-case complexity for these methods is exponential, typically $O(K^{|Q|})$ for a K-MMKP problem [33]. However, CBC's implementation incorporates several heuristics and optimization techniques, often leading to much better performance in practice.

5.5.2 Online Algorithm Complexity

The online algorithm processes each incoming query based on current and limited historical information, offering a more scalable approach. The cost of assigning each query to each model is calculated with a complexity of $O(K)$ per query, where K is the number of models. Determining the optimal model for each query involves comparing these computed costs and updating various system states (queues and historical data), all of which maintain a complexity of $O(K)$ per query.

A limiting factor to the performance of the online algorithm is the queuing required for queries. In Section 5.4.1, we discuss that a higher λ (more frequent arrival rate) or a higher σ_K (longer processing time per query) leads to longer waiting times in the queue. However, as we show, Algorithm 2 can effectively mitigate this effect by considering the length of queues before making a routing decision.

5.5.3 Comparison of Online and Offline Algorithms

The online algorithm scales linearly with the number of models per query, making it more manageable in real-time applications where queries arrive sequentially. This scalability and reduced complexity per query make the online algorithm more runtime competitive compared to the offline approach. While possibly achieving a more optimal allocation through a global view of all queries, the offline algorithm takes a substantial amount of runtime and is less adaptable to bursty arrivals.

Chapter 6

Discussion and Conclusion

6.1 Discussion of Limitations

While this thesis lays a solid foundation for energy-efficient LLM inference, several known limitations and areas for improvement exist. Critical areas for exploration include the following.

While we profile state-of-the-art, efficient, and publicly available LLMs, we do not expand our test range to include LLMs larger than 70B parameters. We could not do this mainly due to compute constraints (e.g., the amount of VRAM per node we had access to), but there are also few publicly available models of this size. Profiling larger models like Falcon 180B or the model in Dash et al. [10] would help address this limitation in scale. Similarly, we only consider raw text-based LLM tasks and do not consider multi-modal features like processing PDFs, audio, and images. Exploring how these factors affect the energy consumption of these extensive services is essential for optimizing modern use cases. Finally, we only test our solutions in a simulation. We would benefit from implementing and testing our proposed algorithms in real-world cloud platforms like AWS, Azure, and Google Cloud to validate effectiveness at scale.

A possible critique of our work is that we do not use optimization tactics like AlpaServe [30] or FastServe [57]. However, we see this as a potential strength of our work. First, our work and framework are independent of optimizations. As long as a model for energy and runtime captures the performance of an LLM using an optimization tactic, then our energy-aware serving methodology works the same. Second, our demonstrated energy consumption and runtime values are an upper bound on what is needed for the outlined inference tasks. Therefore, examining the impacts of optimization strategies on energy consumption is out of the scope of this work, yet we hypothesize that our methodology would work with other methods.

Another point for improvement is in the quantification of accuracy. Using the mean accuracy from the HuggingFace Leaderboard [5] is a good start to quantifying a model's capability, however, performing some level of human evaluation or performing a meta-analysis to determine the achieved accuracy from routing decisions would strengthen this work.

In future studies beyond the data and methods included, we plan to incorporate broader sustainability metrics, such as carbon footprint and water usage [28, 29], into the optimization framework, providing a more holistic approach to sustainable computing. Developing adaptive algorithms that dynamically adjust parameters like ζ in real-time based on current system conditions and workloads could further improve energy efficiency and performance. We also want to explore adapting our solutions to multiple data center cases to increase their effectiveness further and significantly improve energy efficiency and performance. This extension would align our methods with existing work in geographic load balancing [32], a crucial aspect of modern data center provisioning and operation.

6.2 Conclusion

As we have shown, the deployment of LLMs has high computational and energy demands that can present substantial challenges. This thesis addressed these challenges by exploring dynamic workload allocation and energy management in heterogeneous data centers serving LLM inference.

We conducted comprehensive profiling of several LLM families and architectures, including Falcon, Llama-2, and Mistral, across diverse hardware configurations. Our experiments highlighted the intricate trade-offs between model complexity, input/output token sizes, energy consumption, and runtime. These insights informed the development of predictive models for energy and runtime, which are crucial for optimizing data center operations.

Key contributions of this work include the formulation of a workload-aware hybrid data center model that dynamically balances energy efficiency and runtime based on operational priorities. We devised an offline routing algorithm that partitions and directs inference workloads to minimize energy use and runtime while maintaining high accuracy. This algorithm features a tunable parameter, ζ, allowing operators to adjust the balance between energy efficiency and accuracy to suit different scenarios. Our online routing approach also incorporates queue-awareness, ensuring efficient resource utilization and reduced wait times even under variable workloads.

Bibliography

[1] Megha Agarwal, Asfandyar Qureshi, Linden Li Nikhil Sardana, Julian Quevedo, and Daya Khudia. 2023. LLM Inference Performance Engineering: Best Practices. https://www.databricks.com/blog/llm-inference-performance-engineering-best-practices

[2] Ebtesam Almazrouei, Hamza Alobeidli, Abdulaziz Alshamsi, Alessandro Cappelli, Ruxandra Cojocaru, Mérouane Debbah, Étienne Goffinet, Daniel Hesslow, Julien Launay, Quentin Malartic, Daniele Mazzotta, Badreddine Noune, Baptiste Pannier, and Guilherme Penedo. 2023. The Falcon Series of Open Language Models. arXiv:2311.16867 [cs.CL]

[3] Reza Yazdani Aminabadi, Samyam Rajbhandari, Ammar Ahmad Awan, Cheng Li, Du Li, Elton Zheng, Olatunji Ruwase, Shaden Smith, Minjia Zhang, Jeff Rasley, and Yuxiong He. 2022. DeepSpeed-inference: enabling efficient inference of transformer models at unprecedented scale. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis* (Dallas, Texas) *(SC '22)*. IEEE Press, Article 46, 15 pages.

[4] Thomas Anderson, Adam Belay, Mosharaf Chowdhury, Asaf Cidon, and Irene Zhang. 2023. Treehouse: A Case For Carbon-Aware Datacenter Software. *SIGENERGY Energy Inform. Rev.* 3, 3 (oct 2023), 64–70. https://doi.org/10.1145/3630614.3630626

[5] Edward Beeching, Clémentine Fourrier, Nathan Habib, Sheon Han, Nathan Lambert, Nazneen Rajani, Omar Sanseviero, Lewis Tunstall, and Thomas Wolf. 2023. Open LLM Leaderboard. https://huggingface.co/spaces/HuggingFaceH4/open_llm_leaderboard.

[6] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. In *Ad-*

vances in Neural Information Processing Systems, H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (Eds.), Vol. 33. Curran Associates, Inc., 1877–1901. `https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf`

[7] Jiuhai Chen and Jonas Mueller. 2024. Automated Data Curation for Robust Language Model Fine-Tuning. arXiv:2403.12776 [cs.CL]

[8] Andrew A Chien, Liuzixuan Lin, Hai Nguyen, Varsha Rao, Tristan Sharma, and Rajini Wijayawardana. 2023. Reducing the Carbon Impact of Generative AI Inference (Today and in 2035). In *Proceedings of the 2nd Workshop on Sustainable Computer Systems* (Boston, MA, USA) *(HotCarbon '23)*. Association for Computing Machinery, New York, NY, USA, Article 11, 7 pages. `https://doi.org/10.1145/3604930.3605705`

[9] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training Verifiers to Solve Math Word Problems. arXiv:2110.14168 [cs.LG]

[10] Sajal Dash, Isaac Lyngaas, Junqi Yin, Xiao Wang, Romain Egele, Guojing Cong, Feiyi Wang, and Prasanna Balaprakash. 2023. Optimizing Distributed Training on Frontier for Large Language Models. arXiv:2312.12705 [cs.DC]

[11] Radosvet Desislavov, Fernando Martínez-Plumed, and José Hernández-Orallo. 2023. Trends in AI inference energy consumption: Beyond the performance-vs-parameter laws of deep learning. *Sustainable Computing: Informatics and Systems* 38 (2023), 100857. `https://doi.org/10.1016/j.suscom.2023.100857`

[12] Tyna Eloundou, Sam Manning, Pamela Mishkin, and Daniel Rock. 2023. Gpts are gpts: An early look at the labor market impact potential of large language models. *arXiv preprint arXiv:2303.10130* (2023).

[13] Kaijie Fan, Marco D'Antonio, Lorenzo Carpentieri, Biagio Cosenza, Federico Ficarelli, and Daniele Cesarini. 2023. SYnergy: Fine-grained Energy-Efficient Heterogeneous Computing for Scalable Energy Saving. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–13.

[14] William Fedus, Jeff Dean, and Barret Zoph. 2022. A Review of Sparse Expert Models in Deep Learning. arXiv:2209.01667 [cs.LG]

[15] Fly. 2024. Python Codes 25k. `https://huggingface.co/datasets/flytech/python-codes-25k` Accessed: 2024-05-12.

[16] John Forrest, Ted Ralphs, Stefan Vigerske, Haroldo Gambini Santos, John Forrest,

Lou Hafer, Bjarni Kristjansson, Edwin Straver, Miles Lubin, Jan-Willem, Samuel Brito, Matthew Saltzman, Bruno Pitrus, and Fumiaki Matsushima. 2023. *coin-or/Cbc: Release releases/2.10.11.* `https://doi.org/10.5281/zenodo.10041724`

[17] Diandian Gu, Xintong Xie, Gang Huang, Xin Jin, and Xuanzhe Liu. 2023. Energy-Efficient GPU Clusters Scheduling for Deep Learning. arXiv:2304.06381 [cs.DC]

[18] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2021. Measuring Massive Multitask Language Understanding. arXiv:2009.03300 [cs.CY]

[19] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Jack W. Rae, Oriol Vinyals, and Laurent Sifre. 2022. Training Compute-Optimal Large Language Models. arXiv:2203.15556 [cs.CL]

[20] Qinghao Hu, Peng Sun, Shengen Yan, Yonggang Wen, and Tianwei Zhang. 2021. Characterization and prediction of deep learning workloads in large-scale GPU datacenters. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '21)*. Association for Computing Machinery, New York, NY, USA, Article 104, 15 pages. `https://doi.org/10.1145/3458817.3476223`

[21] Hongpeng Huo, Chongchong Sheng, Xinming Hu, and Baifeng Wu. 2012. An energy efficient task scheduling scheme for heterogeneous GPU-enhanced clusters. In *2012 International Conference on Systems and Informatics (ICSAI2012)*. IEEE, 623–627.

[22] Paras Jain, Ajay Jain, Aniruddha Nrusimha, Amir Gholami, Pieter Abbeel, Joseph Gonzalez, Kurt Keutzer, and Ion Stoica. 2020. Checkmate: Breaking the Memory Wall with Optimal Tensor Rematerialization. In *Proceedings of Machine Learning and Systems 2020*. 497–511.

[23] Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, and et al. 2023. Mistral 7B. arXiv:2310.06825 [cs.CL]

[24] Albert Q. Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, Gianna Lengyel, Guillaume Bour, Guillaume Lample, Lélio Renard Lavaud, Lucile Saulnier, Marie-Anne Lachaux, Pierre Stock, Sandeep Subramanian, Sophia Yang, Szymon Antoniak, Teven Le Scao, Théophile Gervet, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2024. Mixtral of Experts. arXiv:2401.04088 [cs.LG]

[25] S. Keshav. 2012. *Mathematical Foundations of Computer Networking.* Pearson Education. `https://books.google.co.uk/books?id=KI9KDgzH52cC`

[26] 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 (SOSP '23).* Association for Computing Machinery, New York, NY, USA, 611–626. `https://doi.org/10.1145/3600006.3613165`

[27] Baolin Li, Siddharth Samsi, Vijay Gadepally, and Devesh Tiwari. 2023. Clover: Toward Sustainable AI with Carbon-Aware Machine Learning Inference Service. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '23).* Association for Computing Machinery, New York, NY, USA, Article 20, 15 pages. `https://doi.org/10.1145/3581784.3607034`

[28] Pengfei Li, Jianyi Yang, Mohammad A. Islam, and Shaolei Ren. 2023. Making AI Less "Thirsty": Uncovering and Addressing the Secret Water Footprint of AI Models. arXiv:2304.03271 [cs.LG]

[29] Pengfei Li, Jianyi Yang, Adam Wierman, and Shaolei Ren. 2024. Towards Environmentally Equitable AI via Geographical Load Balancing. arXiv:2307.05494 [cs.AI]

[30] Zhuohan Li, Lianmin Zheng, Yinmin Zhong, Vincent Liu, Ying Sheng, Xin Jin, Yanping Huang, Zhifeng Chen, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. 2023. AlpaServe: Statistical Multiplexing with Model Parallelism for Deep Learning Serving. In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23).* USENIX Association, Boston, MA, 663–679. `https://www.usenix.org/conference/osdi23/presentation/li-zhouhan`

[31] Qianlin Liang, Walid A Hanafy, Ahmed Ali-Eldin, and Prashant Shenoy. 2023. Model-driven cluster resource management for ai workloads in edge clouds. *ACM Transactions on Autonomous and Adaptive Systems* 18, 1 (2023), 1–26.

[32] Zhenhua Liu, Minghong Lin, Adam Wierman, Steven H. Low, and Lachlan L.H. Andrew. 2011. Greening geographical load balancing. *SIGMETRICS Perform. Eval. Rev.* 39, 1 (jun 2011), 193–204. `https://doi.org/10.1145/2007116.2007139`

[33] S. Martello and P. Toth. 1990. *Knapsack Problems: Algorithms and Computer Implementations.* Wiley. `https://books.google.co.uk/books?id=0dhQAAAAMAAJ`

[34] Timothy R. McIntosh, Teo Susnjak, Tong Liu, Paul Watters, and Malka N. Halgamuge. 2024. Inadequacies of Large Language Model Benchmarks in the Era of Generative Artificial Intelligence. arXiv:2402.09880 [cs.AI]

[35] Xinxin Mei, Xiaowen Chu, Hai Liu, Yiu-Wing Leung, and Zongpeng Li. 2017. Energy

efficient real-time task scheduling on CPU-GPU hybrid clusters. In *IEEE INFOCOM 2017-IEEE Conference on Computer Communications*. IEEE, 1–9.

[36] Xupeng Miao, Chunan Shi, Jiangfei Duan, Xiaoli Xi, Dahua Lin, Bin Cui, and Zhihao Jia. 2024. SpotServe: Serving Generative Large Language Models on Preemptible Instances. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS '24)*. Association for Computing Machinery, New York, NY, USA, 1112–1127. https://doi.org/10.1145/3620665.3640411

[37] NVIDIA. Accessed 2024. NVIDIA-NVML. https://docs.nvidia.com/deploy/nvml-api/index.html. Available online.

[38] OpenAI, :, Josh Achiam, Steven Adler, and et al. 2023. GPT-4 Technical Report. arXiv:2303.08774 [cs.CL]

[39] Pratyush Patel, Esha Choukse, Chaojie Zhang, Íñigo Goiri, Brijesh Warrier, Nithish Mahalingam, and Ricardo Bianchini. 2024. 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 (ASPLOS '24)*. Association for Computing Machinery, New York, NY, USA, 207–222. https://doi.org/10.1145/3620666.3651329

[40] Guilherme Penedo, Quentin Malartic, Daniel Hesslow, Ruxandra Cojocaru, Alessandro Cappelli, Hamza Alobeidli, Baptiste Pannier, Ebtesam Almazrouei, and Julien Launay. 2023. The RefinedWeb Dataset for Falcon LLM: Outperforming Curated Corpora with Web Data, and Web Data Only. arXiv:2306.01116 [cs.CL]

[41] Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, Anselm Levskaya, Jonathan Heek, Kefan Xiao, Shivani Agrawal, and Jeff Dean. 2022. Efficiently Scaling Transformer Inference. arXiv:2211.05102 [cs.LG]

[42] PowerAPI. 2024. PyJoules: Python-based energy measurement library for various domains including NVIDIA GPUs. https://github.com/powerapi-ng/pyJoules. Accessed: 2024-05-31.

[43] Jakob Puchinger, Günther R Raidl, and Ulrich Pferschy. 2006. The core concept for the multidimensional knapsack problem. In *Evolutionary Computation in Combinatorial Optimization: 6th European Conference, EvoCOP 2006, Budapest, Hungary, April 10-12, 2006. Proceedings 6*. Springer, 195–208.

[44] Samyam Rajbhandari, Conglong Li, Zhewei Yao, Minjia Zhang, Reza Yazdani Aminabadi, Ammar Ahmad Awan, Jeff Rasley, and Yuxiong He. 2022. DeepSpeed-MoE: Advancing Mixture-of-Experts Inference and Training to Power Next-Generation AI Scale. In *Proceedings of the 39th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 162)*, Kamalika Chaudhuri, Stefanie

Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato (Eds.). PMLR, 18332–18346. `https://proceedings.mlr.press/v162/rajbhandari22a.html`

[45] Lavanya Ramapantulu, Bogdan Marius Tudor, Dumitrel Loghin, Trang Vu, and Yong Meng Teo. 2014. Modeling the energy efficiency of heterogeneous clusters. In *2014 43rd International Conference on Parallel Processing*. IEEE, 321–330.

[46] Meta Research. [n. d.]. Introducing the AI Research SuperCluster — Meta's cutting-edge AI supercomputer for AI research. `https://ai.meta.com/blog/ai-rsc/`. Accessed: 2024-05-09.

[47] Siddharth Samsi, Dan Zhao, Joseph McDonald, Baolin Li, Adam Michaleas, Michael Jones, William Bergeron, Jeremy Kepner, Devesh Tiwari, and Vijay Gadepally. 2023. From Words to Watts: Benchmarking the Energy Costs of Large Language Model Inference. arXiv:2310.03003 [cs.CL]

[48] Ying Sheng, Lianmin Zheng, Binhang Yuan, Zhuohan Li, Max Ryabinin, Beidi Chen, Percy Liang, Christopher Ré, Ion Stoica, and Ce Zhang. 2023. FlexGen: high-throughput generative inference of large language models with a single GPU. In *Proceedings of the 40th International Conference on Machine Learning (ICML'23)*. JMLR.org, Article 1288, 23 pages.

[49] Jovan Stojkovic, Esha Choukse, Chaojie Zhang, Inigo Goiri, and Josep Torrellas. 2024. Towards Greener LLMs: Bringing Energy-Efficiency to the Forefront of LLM Inference. arXiv:2403.20306 [cs.AI]

[50] Xiaoyong Tang and Zhuojun Fu. 2020. CPU–GPU utilization aware energy-efficient scheduling algorithm on heterogeneous computing systems. *IEEE Access* 8 (2020), 58948–58958.

[51] R. Taori, I. Gulrajani, T. Zhang, and et al. 2024. Stanford alpaca: An instruction following llama model. `https://github.com/tatsu-lab/stanford_alpaca`. Accessed: 2024-01-15.

[52] Gemini Team, Machel Reid, Nikolay Savinov, and et al. 2024. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. arXiv:2403.05530 [cs.CL]

[53] Hugo Touvron, Louis Martin, Kevin Stone, and et al. 2023. Llama 2: Open Foundation and Fine-Tuned Chat Models. arXiv:2307.09288 [cs.CL]

[54] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems* (Long Beach, California, USA) *(NIPS'17)*. Curran Associates Inc., Red Hook, NY, USA, 6000–6010.

[55] Yuxin Wang, Yuhan Chen, Zeyu Li, Zhenheng Tang, Rui Guo, Xin Wang, Qiang Wang, Amelie Chi Zhou, and Xiaowen Chu. 2024. Towards Efficient and Reliable LLM Serving: A Real-World Workload Study. arXiv:2401.17644 [cs.DC]

[56] Samuel Williams, Andrew Waterman, and David Patterson. 2009. Roofline: an insightful visual performance model for multicore architectures. *Commun. ACM* 52, 4 (apr 2009), 65–76. `https://doi.org/10.1145/1498765.1498785`

[57] Bingyang Wu, Yinmin Zhong, Zili Zhang, Gang Huang, Xuanzhe Liu, and Xin Jin. 2023. Fast Distributed Inference Serving for Large Language Models. arXiv:2305.05920 [cs.LG]

[58] Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soojeong Kim, and Byung-Gon Chun. 2022. Orca: A Distributed Serving System for Transformer-Based Generative Models. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*. USENIX Association, Carlsbad, CA, 521–538. `https://www.usenix.org/conference/osdi22/presentation/yu`

[59] Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. HellaSwag: Can a Machine Really Finish Your Sentence? arXiv:1905.07830 [cs.CL]

[60] Juntao Zhao, Borui Wan, Yanghua Peng, Haibin Lin, and Chuan Wu. 2024. LLM-PQ: Serving LLM on Heterogeneous Clusters with Phase-Aware Partition and Adaptive Quantization. arXiv:2403.01136 [cs.LG]

[61] Lianmin Zheng, Zhuohan Li, Hao Zhang, Yonghao Zhuang, Zhifeng Chen, Yanping Huang, Yida Wang, Yuanzhong Xu, Danyang Zhuo, Eric P. Xing, Joseph E. Gonzalez, and Ion Stoica. 2022. Alpa: Automating Inter- and Intra-Operator Parallelism for Distributed Deep Learning. arXiv:2201.12023 [cs.LG]

[62] Zangwei Zheng, Xiaozhe Ren, Fuzhao Xue, Yang Luo, Xin Jiang, and Yang You. 2023. Response Length Perception and Sequence Scheduling: An LLM-Empowered LLM Inference Pipeline. *arXiv preprint arXiv:2305.13144* (2023).