TU WIEN Informatics

A Dynamic Routing Approach for Sustainable Language Model Inference

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Data Science

eingereicht von

Thomas Markus Ziller, BSc
Matrikelnummer 11739403

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Shashikant Shankar Ilager, M.Tech. PhD
Mitwirkung: Univ.Prof.in Mag.a rer.soc.oec. Dr.in rer.soc.oec. Ivona Brandić

Wien, 5. Mai 2025

Thomas Markus Ziller	Shashikant Shankar Ilager

TU WIEN Informatics

A Dynamic Routing Approach for Sustainable Language Model Inference

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Data Science

by

Thomas Markus Ziller, BSc
Registration Number 11739403

to the Faculty of Informatics

at the TU Wien

Advisor: Shashikant Shankar Ilager, M.Tech. PhD
Assistance: Univ.Prof.in Mag.a rer.soc.oec. Dr.in rer.soc.oec. Ivona Brandić

Vienna, May 5, 2025

Thomas Markus Ziller	Shashikant Shankar Ilager

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Erklärung zur Verfassung der Arbeit

Thomas Markus Ziller, BSc

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.
Ich erkläre weiters, dass ich mich generativer KI-Tools lediglich als Hilfsmittel bedient habe und in der vorliegenden Arbeit mein gestalterischer Einfluss überwiegt. Im Anhang „Übersicht verwendeter Hilfsmittel“ habe ich alle generativen KI-Tools gelistet, die verwendet wurden, und angegeben, wo und wie sie verwendet wurden. Für Textpassagen, die ohne substantielle Änderungen übernommen wurden, haben ich jeweils die von mir formulierten Eingaben (Prompts) und die verwendete IT- Anwendung mit ihrem Produktnamen und Versionsnummer/Datum angegeben.

Wien, 5. Mai 2025

Thomas Markus Ziller

Danksagung

Ich möchte mich von Herzen bei meinem Betreuer, Shashikant Ilager, bedanken, der mich während meiner gesamten Masterarbeit eng begleitet hat. Seine Geduld und sein stetiges Engagement haben es mir ermöglicht, dieses spannende Thema zu verfolgen und abzuschließen. Mein Dank gilt auch Prof. Ivona Brandic, die mir zum Ende meines Studiums die Möglichkeit gegeben hat, in ihrer Research Group mitzuwirken. Die dabei gewonnenen Einblicke haben mich fachlich und persönlich geprägt. Ein besonderer Dank geht außerdem an Alessandro Tundo, Prof. Ezio Bartocci und Prof. Leonardo Mariani für die wertvollen Diskussionen und den fachlichen Input, der mir wesentlich bei der Ausarbeitung geholfen hat. Auch bei der gesamten HPC Research Group möchte ich mich für die gemeinsame Zeit bedanken – sie war geprägt von selbstloser Unterstützung und einer unerwarteten Vertrautheit.

Mein tiefster Dank gilt außerdem meiner Familie, insbesondere meinen Eltern, die mir dieses Studium ermöglicht haben. Ebenso möchte ich meiner Freundin Alex danken, die mich über all die Jahre hinweg unterstützt und den von Amplituden gezeichneten Studentenalltag mitgetragen und mitertragen hat. Abschließend möchte ich mich bei meinem guten Freund DI Thomas Niedermayer bedanken, der mir mit seiner Erfahrung über mein gesamtes Studium an der TU Wien hinweg bei fachlichen und organisatorischen Fragen zur Seite stand.

Acknowledgements

I wish to express my heartfelt gratitude to my advisor, Shashikant Ilager, who closely guided me throughout my master's thesis. His patience and continuous commitment allowed me to pursue and complete this fascinating topic. I also thank Prof. Ivona Brandic for giving me the opportunity to contribute to her Research Group in the final stages of my studies. The insights gained there have shaped me both professionally and personally. Special thanks go to Alessandro Tundo, Prof. Ezio Bartocci, and Prof. Leonardo Mariani for their valuable discussions and expert input that significantly helped me develop my work. I'm also grateful to the entire HPC Research Group for our time together – characterized by selfless support and an unexpected sense of familiarity.

My deepest thanks go to my family, especially my parents, who made this course of study possible. I'd also like to thank my girlfriend Alex, who supported me throughout these years and both shared and endured the student life with its characteristic ups and downs. Finally, I'm grateful to my good friend DI Thomas Niedermayer, who stood by me throughout my entire time at TU Wien, offering guidance on both academic and administrative matters.

Kurzfassung

Große Sprachmodelle (LLMs) bieten beispiellose Fähigkeiten, doch ihre breite Anwendung wird durch den erheblichen Bedarf an Rechenressourcen während der Inferenz, insbesondere den Energieverbrauch, eingeschränkt. Statische *One-Model-Fits-All*-Inferenzstrategien sind oft ineffizient, da sie das vielfältige Spektrum verfügbarer Modelle nicht nutzen und variierende Anforderungen von Anfragen nicht berücksichtigen. Diese Arbeit adressiert diese Herausforderung durch den Vorschlag und die Evaluierung eines dynamischen, kontextsensitiven Routing-Frameworks, das darauf ausgelegt ist, den Kompromiss zwischen Inferenzgenauigkeit und Energieeffizienz zu optimieren.

Das vorgeschlagene Middleware-Framework verwendet kontextuelle Multi-Armed-Bandit (MAB) Algorithmen, um eine adaptive Routing-Policy online zu lernen. Es verarbeitet eingehende Anfragen durch die Extraktion einfacher Merkmale und leitet jede Anfrage an das am besten geeignete Modell aus einem heterogenen Pool weiter, basierend auf der gelernten Leistung hinsichtlich Genauigkeit und gemessenem Energieverbrauch. Dieser Online-Lernansatz operiert unter partiellem Feedback, vermeidet die Notwendigkeit umfangreicher Offline-Kalibrierung und vereinfacht die Integration neuer Modelle.

Wir haben das Framework implementiert und evaluiert, indem wir etablierte kontextuelle MAB-Strategien auf einem vielfältigen Datensatz von fünf Benchmark-Aufgaben und einem Pool von 16 aktuellen, frei zugänglichen LLMs angewendet haben. Experimentelle Ergebnisse zeigen, dass die dynamischen Routing-Strategien statische (Einzelmodell-) und zufällige Baselines (Referenzstrategien) durchweg übertreffen, indem sie überlegene Genauigkeits-Energie-Betriebspunkte erreichen, die oft über die statische Pareto-Front hinausgehen. Im Vergleich zum zufälligen Routing erzielten die kontextuellen Banditen Genauigkeitssteigerungen von 22% bei gleichzeitiger Reduzierung des kumulativen Energieverbrauchs um 31%. Das Framework ermöglichte ein effektives Navigieren des Genauigkeits-Energie-Kompromisses basierend auf einem konfigurierbaren Präferenzparameter (λ) und zeigte Adaptivität durch erfolgreiche Integration eines neuen Modells zur Laufzeit ohne Policy-Retraining.

Die Ergebnisse unterstützen die Umsetzbarkeit und Effektivität des Einsatzes kontextueller Bandit-Algorithmen für echtzeitfähiges, energieeffizientes LLM-Inferenz-Routing. Dieser Ansatz bietet eine praktikable und skalierbare Lösung zur Optimierung der Ressourcennutzung in dynamischen LLM-Bereitstellungsszenarien.

Abstract

Large language models (LLMs) offer unprecedented capabilities but their widespread deployment is constrained by significant computational resource demands during inference, especially energy consumption. Static, *one-model-fits-all* inference strategies are often inefficient as they fail to leverage the diverse spectrum of available models and do not account for varying query requirements. This thesis addresses this challenge by proposing and evaluating a dynamic, context-aware routing framework designed to optimize the trade-off between inference accuracy and energy efficiency.

The proposed middleware framework utilizes contextual multi-armed bandit (MAB) algorithms to learn an adaptive routing policy online. It processes incoming queries by extracting lightweight contextual features (task type, semantic cluster, text complexity) and routes each query to the most suitable model from a heterogeneous pool based on observed performance with respect to accuracy and measured energy consumption. This online learning approach operates under partial feedback, eliminates the need for extensive offline calibration, and simplifies the integration of new models.

We implemented and evaluated our framework using established MAB strategies on a diverse set of 5 benchmark tasks and a pool of 16 contemporary open-access LLMs. Experimental results demonstrate that the dynamic routing strategies consistently outperform static (single-model) and random baselines by achieving superior accuracy-energy operating points often beyond the static Pareto front. Compared to random routing, the contextual bandits achieved accuracy gains of 22% while reducing cumulative energy consumption by 31%. The framework allows effective maneuvering of the accuracy-energy trade-off based on a configurable preference parameter (λ) and demonstrated adaptability by successful incorporation of a new model added at runtime without policy retraining.

The findings support the feasibility and effectiveness of using contextual bandit algorithms for real-time, energy-efficient LLM inference routing. This approach offers a practical and scalable solution for optimizing resource utilization in dynamic LLM deployment scenarios.

Contents

CHAPTER 1

Introduction

1.1 Motivation

Large language models (LLMs) such as GPT-4 and Claude 3.7 Sonnet represent major advancements in artificial intelligence (AI) capabilities [AAA+23a]. These models show high proficiency in generating human-like text and are able to handle a wide spectrum of natural language tasks. Consequently, organizations across industries are adopting them in various applications and workflows, which accelerates automation in areas such as customer service and market research [Com25]. However, the deployment of these powerful models requires substantial computational resources, mainly attributed to their vast scale, leading to substantial energy consumption and operational costs [AAA+23a].

While the resources required for training LLMs receive much attention, large-scale *inference* deserves equal focus. Cumulative energy consumed by millions of inference calls in operational settings can surpass the energy used for initial training [WRG+22]. A single ChatGPT query is estimated to consume approximately 2.9 Wh [dV23], which, extrapolated to millions or even billions of queries, translates into a remarkable resource footprint. Furthermore, data centers that host both training and inference already draw an appreciable share of global electricity, and their demand continues to grow [Int25]. Thus, developing strategies to lower inference resource consumption without disproportionately harming model performance arises as a critical area of current AI research.

Even in well-resourced data centers with flexible infrastructures designed for high-capacity operations, cost and scalability remain concerns directly tied to the resource demands of models [HYW+24]. A common practice is the reliance on a static *one-model-fits-all* inference strategy. Every query is routed to the same large model, regardless of complexity or required quality [CZZ23]. This uniformity wastes resources because many queries, such as non-critical translation tasks, could be sufficiently handled by smaller, cheaper models. Studies suggest that accepting minor reductions in quality can yield significant

energy savings, yet, the static single-model paradigm dominates and neglects exploitable trade-offs [FAHA23, SRR+22].

Efforts to mitigate these inefficiencies have motivated the exploration of routing strategies for inference optimization. System designs such as RouteLLM [OAW+25] and MixLLM [WLC+25] optimize model selection, while cascading approaches like FrugalGPT [CZZ23] iteratively escalate model capabilities based on input characteristics. While promising, these approaches often show limitations, as they typically target only a small, predefined set of models and rely on extensive offline calibration to establish routing rules or configurations. This reliance critically limits their ability to react quickly to newly available models or shifting workloads, posing a significant barrier in the rapidly evolving LLM landscape. Consequently, the broad variety of models available on platforms such as HuggingFace remains underutilized, preventing systems from fully leveraging potential efficiency gains.

In response to these open challenges, moving beyond static or rigidly calibrated systems is crucial for realizing truly efficient LLM deployment. Static, context-agnostic approaches are inherently inefficient because LLM performance varies significantly with query characteristics (see Section 2.3.4). Moreover, the practical limitations inherent in existing dynamic routing approaches, specifically their calibration overhead and slow adaptation, limit their effectiveness in balancing the conflicting objectives of *accuracy* and *resource efficiency* under uncertainty online. Addressing this requires a system capable of operating in real-time under uncertainty while adapting its policy *online* to incorporate newly available models and react to performance feedback, all without demanding extensive offline retraining. Collectively, these requirements point towards an online learning approach capable of handling the accuracy-efficiency *trade-off* based on partial feedback.

To address these needs, this work frames the *dynamic, context-aware routing* problem as a *multi-objective optimization problem (MOOP)*. We propose approximating its solution via *multi-armed bandit (MAB)* algorithms. The proposed framework acts as middleware that learns an adaptive policy online to assign queries to the most suitable model from a diverse pool based on measured accuracy and efficiency (e.g., energy consumption) in specific contexts. Our approach inherently handles partial feedback and allows for continuous adaptation.

1.2 Research Problem & Objectives

1.2.1 Problem Statement

The computational cost of LLM inference remains a critical challenge for large-scale AI deployment. Static, one-model-fits-all pipelines waste resources because many queries do not require frontier-model capabilities. The rapid commoditization of LLMs and the frequent release of new models further complicate static approaches which demand

systems that can effectively learn the capabilities of previously unseen models. The key limitations of existing inference routing strategies can be summarized as:

- **Limited Adaptability:** Current dynamic approaches often rely on a fixed set of models and require extensive offline calibration, and thus, are lacking flexibility to adapt to new models or shifting workloads.
- **Underutilization of Model Diversity:** Repositories of models with a diverse range of capabilities remain underexplored due to the calibration overhead associated with incorporating them.
- **Lack of Real-Time Optimization:** Many routing strategies optimize model selection based on precomputed rules rather than facing the challenges of uncertainty in real-time optimization.

These issues motivate an *adaptive, scalable, energy-efficient* routing framework driven by real-time requirements. Consequently, this thesis aims to develop a **middleware framework** that routes queries while balancing contextual accuracy and energy efficiency. We articulate this goal into the following three sub-objectives:

O1: **Identify** factors affecting the accuracy-efficiency trade-off in LLM inference.

O2: **Develop** a routing mechanism that assigns queries to models dynamically based on context and learned performance.

O3: **Evaluate** trade-offs between accuracy and energy consumption across diverse workloads using the developed mechanism.

To achieve these objectives, we investigate the following research questions:

RQ$_1$ What key factors influence the accuracy-efficiency trade-off in large language model inference?
This research question investigates factors that affect the balance between contextual accuracy and resource efficiency such as model characteristics and query properties. We explore how these factors can be represented quantitatively and identify the underlying multi-objective optimization problem (MOOP).

RQ$_2$ How can a dynamic routing system effectively manage the accuracy-efficiency trade-off online?
Addressing the MOOP and its properties like partial feedback and online learning identified in RQ1, this question explores how a dynamic routing system can learn and adapt its model selection policy accordingly. The goal is to approximate Pareto-optimal solutions for the accuracy-energy trade-off by leveraging contextual information during model selection.

RQ$_3$ How can the effectiveness of the proposed dynamic routing system be validated under simulated operational conditions?
In this final research question, we explore how our online, adaptive routing system performs against baselines under simulated operational conditions and what insights can be gained regarding the factors influencing its optimization achievements and adaptability.

1.3 Methodological Approach

The thesis follows a multi-stage experimental design that integrates a literature review, prototype development, controlled experimentation, and extensive evaluation to assess the feasibility and efficacy of dynamic routing strategies for LLM inference.

1.3.1 Literature Review

The study conducts a comprehensive literature review to examine existing research on LLM inference optimization, dynamic routing, and multi-objective optimization. Relevant publications are primarily sourced from *IEEE Xplore*, *Google Scholar*, and the *ACM Digital Library* to identify state-of-the-art methods and their limitations. The review focuses on multi-model routing strategies and adaptive online learning techniques for multi-objective optimization. Insights gained from this survey shapes the framework design and evaluation setup.

1.3.2 Proposed Solution: A Middleware Framework

The key contribution of this thesis is the development and evaluation of a middleware prototype that is designed for dynamic, context-aware, and energy-efficient LLM inference routing. This framework implements MAB approaches outlined earlier, which enable the system to refine its routing decision-making from observed responses in real-time. The specific architecture, algorithms, and implementation details are presented in Chapters 5 and 6.

1.3.3 Experimental Setup

The empirical evaluation is conducted in a controlled environment on an NVIDIA A100 Tensor Core GPU (80GB VRAM), provisioned by the HPC research group at TU Wien. The model pool consists of publicly available open-weight models of varying capabilities and parameter sizes to create a representative sample. For simulating diverse workloads, we source queries from five widely established benchmark datasets. Measurements used at runtime are used to evaluate and analyze system effectiveness and the efficiency–accuracy trade-off.

1.3.4 Evaluation

The framework's effectiveness is determined via comparison with random routing and a static single-model baselines. We vary query mix, model availability, and energy constraints, and apply statistical significance tests including repeated simulated runs to account for stochastic MAB behaviour.

1.4 Outline

The work is structured to progress from foundational concepts to the design, implementation, and evaluation of the proposed dynamic inference routing framework. Each chapter builds on its predecessor and contributes to a coherent narrative that addresses the research questions through theoretical grounding, problem identification, methodological development, and empirical validation.

Chapter 2 establishes the theoretical groundwork. It first explains the core Transformer architecture, the basis of LLMs. Next, the chapter explores scaling laws which connect model size, training data, computing power, and performance. We discuss how these laws changed over time, examine inference-aware scaling considerations, and detail the practical development of model families. Lastly, the chapter treats the large sustainability problems and ethical issues that large-scale LLM deployment creates. The discussion covers environmental effects like energy use and carbon output.

Chapter 3 reviews prior work on inference routing strategies. After providing a concise overview of existing inference optimization research, we categorize relevant systems across seven shared dimensions that constitute key characteristics of inference routing systems. These dimensions facilitate a more meaningful comparison and analysis than pure qualitative discussion. The chapter concludes with elaborating on the limitations of existing methods to address the challenges outlined in Section 1.1.

Chapter 4 formalizes the problem of energy-efficient LLM inference by defining key concepts, and frames it as multi-objective optimization between accuracy and efficiency to establish the theoretical basis that underpins the proposed solution. Chapter 5 presents how the middleware aims to solve the dynamic inference routing problem. It provides details on the system architecture, discusses how the middleware interacts with models, and explains how MAB-based decision-making is integrated.

Chapter 6 covers the actual implementation of the routing framework including details on the hardware setup and the measurement tools used for efficiency evaluation. Finally, it discusses key engineering challenges. Chapter 7 presents the evaluation plan and results. It outlines experiments, shows the experimental findings and discusses their implications and limitations.

CHAPTER 2

Background: LLM Fundamentals and Sustainability Challenges

This chapter provides the essential foundation for understanding the proposed dynamic routing framework. We begin by discussing the fundamental Transformer architecture that LLMs are based on. Building on this, the next part of the chapter gives insights into the evolution from early Transformer models to the current large-scale models and the decisive role of scaling laws in driving model development. Finally, we examine sustainability challenges connected to the adoption of LLMs such as environmental impact and resource accessibility, which motivate the need for efficiency optimizations like the one explored in this thesis.

2.1 The Transformer Architecture

Over the past several years, Transformer models have fundamentally changed the field of natural language processing and largely replaced the previously dominant RNN and LSTM architectures. In their foundational publication *Attention Is All You Need*, Vaswani et al. [VSP+17] introduced the Transformer architecture. A key innovation was the model's ability to process all tokens of an input sequence in parallel, unlike sequential models. This parallel processing capability significantly improved computational efficiency while enhancing the model's ability to capture complex semantic relationships between words. The following section provides an overview of the motivation and functionality of Transformer components, and leads towards how we arrived at today's LLMs.

2.1.1 Core Components

The fundamental Transformer architecture combines several core components. Although individual components were not entirely novel, their effective combination, especially

surrounding the self-attention mechanism, enabled substantial advances in processing complex language structures and achieving unprecedented scalability that led to major performance breakthroughs. The following is a detailed breakdown of these core components:

Self-attention
Self-attention is a central mechanism within the Transformer architecture. It allows the model to identify and quantitatively encode how words are associated with each other, and, unlike RNN approaches, to gain a more holistic understanding of the sequence. Transformer models proved particularly effective for tasks that involve long sequences or complex linguistic structures[VSP+17, DCLT19]. To briefly summarize the shortcomings of previous architectures:

- **Sequential processing:** RNN-based models compute the hidden state at position t based on the input at t and the hidden state from position $t-1$, therefore processing tokens sequentially. Due to this dependency, this procedure has to be repeated for each token, and complicates effective parallelization, slowing down both training and inference. Transformers, in contrast, mitigate this limitation because the self-attention mechanism computes scores between all token pairs simultaneously.
- **Long-range dependencies:** RNN-based models struggle with long sequence lengths caused by the sequential passing of information, often running into gradients to diminish or explode [BSF94]. However, various language tasks, such as summarization and question-answering, require the capability to handle long sequences. Again, Transformer models address this limitation as self-attention enables direct connections between any two tokens, which facilitates capturing long-range dependencies.

In practice, self-attention is calculated based on three distinct vector representations derived from each input token's embedding: the query (Q), the key (K), and the value (V). The system computes the attention score between two tokens via a scaled dot product of their respective Q and K vectors. It then scales the result by $1/\sqrt{d_k}$ (where d_k is the key vector's dimension) to prevent the dot product from exploding, before a softmax function generates the final attention weights [VSP+17].

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

The resulting attention weights quantify the relevance of tokens when the model computes the representation for a specific token. The model then uses these weights to compute a weighted sum of the V (value) vectors from all tokens in the sequence. This weighted sum forms the token's output representation and encodes sequence information weighted

by relevance. By utilizing not just one but multiple attention heads, multi-head attention (MHA) extends this procedure and uses multiple different vector representations for each Q, K, and V of a token via independently learned projection weights. MHA enhances this process by performing the self-attention calculation multiple times in parallel, where each head uses different learned linear projections for the Q, K, and V vectors. Eventually, the system concatenates the outputs from all heads and then projects this combined output back to the original embedding dimension through a final linear layer [VSP+17].

Positional Embeddings
Since the self-attention mechanism itself does not inherently process tokens in order, Transformers require a way to encode information about the position of tokens within the sequence [VSP+17, SUV18]. To allow this kind of parallel processing of inherently sequential data like natural language, a positional element is added to the initial token embedding. The architecture adds positional embeddings to the initial token embeddings, visualized at the bottom of Figure 2.1. The combined vector serves as the input embedding and contains information about both the token's meaning and its absolute or relative position.

Feedforward Network
After the attention mechanism, a feedforward neural network (FFN) independently processes each position in the sequence. This network applies further non-linear transformations to the attention output which allows the model to capture more complex relationships within the data.

Residual Connections and Layer Normalization
Besides MHA and FFN, Transformer models implement residual connections, a type of skip connection, and layer normalization. Residual connections add the input of the layer directly to the output to preserve more original information and reduce the risk of vanishing or exploding gradients [HZRS16]. Layer normalization scales and centers the layer's activation values which avoids one part of the model overpowering another [BKH16]. Together, they are essential regularization mechanisms that keep weight updates stable and propagation well-scaled.

2.1.2 Architectural Variants

The foundational Transformer architecture described above serves as a blueprint for modern LLMs. Since its introduction [VSP+17], researchers have developed many variants by adapting or focusing on specific parts of the original encoder-decoder structure. These adaptations often intend to optimize performance for particular categories of natural language tasks and are commonly grouped into three main architectural families:

Encoder-Only
Encoder-only models process the entire input sequence bidirectionally, a method allowing

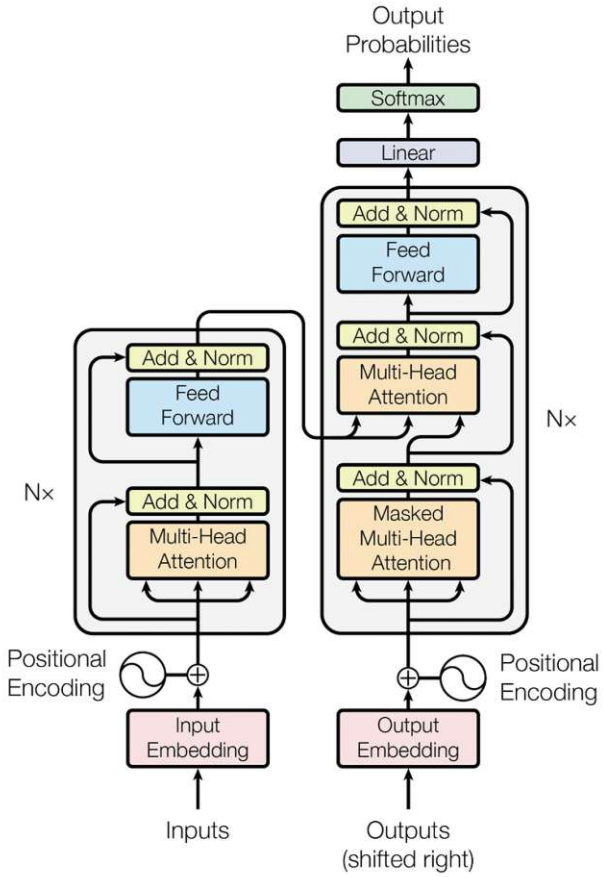

Figure 2.1: **The original Transformer architecture as proposed by Vaswani et al. [VSP+17].** The diagram shows the encoder (left) and decoder (right) stacks with the core components multi-head attention, feed-forward networks, and residual connections with layer normalization.

them to capture a word's context from both preceding and succeeding words with the primary goal being text comprehension. Training typically involves randomly masking (hiding) a portion of the input tokens. The model must then predict these missing tokens based on the surrounding context. In practice, developers often extend the resulting base model with additional linear layers or fine-tune it to perform specific language tasks such as sentiment analysis or Named Entity Recognition (NER). However, a key limitation of the standard encoder-only design is its non-generative nature, which prevents its application to a wide range of tasks unless modified. Well-known examples of this architecture include BERT [DCLT19] and its derivatives, ALBERT [LCG+20] and RoBERTa [LOG+19].

Decoder-Only
In contrast, decoder-only models primarily target sequence generation. They operate unidirectionally, predicting one token at a time strictly from left to right by attending only to the current and preceding tokens via masked self-attention. This iterative prediction process mimics the natural flow of language and makes it suitable for open-ended tasks such as creative writing and code generation. When trained on massive datasets at sufficient scale, decoder-only models have demonstrated strong generalization capabilities [BMR+20, RWC+19]. Nevertheless, this unidirectional processing introduces drawbacks. Compared to encoder-only models of similar parameter sizes, decoder-only models often struggle with tasks that require deep text understanding, such as classification or extractive question-answering (QA) [BSMC24, FLY+23].

Encoder-decoder
Encoder-decoder architectures, often called sequence-to-sequence (seq2seq) models, try to combine the strengths of the previous two types. Their designs are tailored for tasks where an input sequence needs transformation into an output sequence. The process involves an encoder component reading the input and encoding it into a dense representation. This representation is subsequently passed to a decoder component, which proceeds to generate the output autoregressively. This mixed approach has proven particularly effective in tasks like machine translation, paraphrasing, and text summarization. Notable encoder-decoder models include T5 (Text-to-Text Transfer Transformer) [RSR+20] and BART (Bidirectional and Auto-Regressive Transformer) [LLG+20]. However, a trade-off exists, as encoder-decoder models tend to be less efficient than encoder-only architectures, while not yet demonstrating the generative scalability achieved by decoder-only models [RSR+20, KMH+20]. Historically, this limitation has hindered their large-scale adoption for highly specialized tasks and, as a result, has led researchers to often prioritize the more focused encoder-only and decoder-only architectures.

The Transformer architecture provides the essential foundation for modern language models. Key features like parallel processing and inherent scalability, especially evident in decoder designs, made new training approaches feasible. These approaches moved away from training separate models for each task and instead focused on creating large, general-purpose models. This section details how innovative training methods and significant scaling built upon this foundation to produce today's powerful LLMs.

2.2 Evolution to Large Language Models

This section discusses the central developments and paradigm shifts that resulted in today's powerful language models by building upon this architecture. Progress accelerated not only through architectural adjustments, but substantially through innovative training methods and a dramatic increase in scale.

2.2.1 The Era of Pre-training and Transfer Learning

Enabled by the scalability of Transformer architectures (Section 2.1.2), a common methodology for language model training emerged. First, the model is **pre-trained** on vast text corpora to build an understanding of language and acquire general knowledge, followed by **fine-tuning** on smaller, task- or domain-specific data to adapt the model for its intended applications [RNSS18, DCLT19]. Once the foundation is built via pre-training, the model allows comparably efficient adaptation to specific downstream tasks as it only needs a relatively small amount of task-specific labeled data during fine-tuning to achieve strong performance.

Early influential models soon showed the effectiveness of the pre-training paradigm. Built on the encoder-only architecture's strength in text understanding, BERT [DCLT19] set new performance records on natural language understanding (NLU) benchmarks like text classification and open-book question answering. Its success validated large-scale pre-training for comprehension tasks. Meanwhile, the GPT series pushed forward the decoder-only path. GPT-2 [RWC+19], notably larger with 1.5 B parameters compared to BERT-Large's 340 M, particularly excelled at generating coherent text. It also showed surprising skill on a variety of tasks without specific fine-tuning (zero-shot learning). The success of these models confirmed the value of large-scale pre-training and gave insights into the different strengths of encoder versus decoder designs.

2.2.2 Investigating the Impact of Scale

The demonstrated success of the pre-training paradigm with models like BERT and GPT-2 raised questions about the actual limits of this approach for achieving more *general* capabilities. Researchers started exploring whether further pushing parameter sizes could unlock fundamentally different model behaviors rather than only refining performance on existing benchmarks. This led to hypotheses around *emergent abilities* which suggest that sufficiently scaled models might develop complex cognitive skills that are not present in smaller models [WTB+22]. Initial scaling experiments by Kaplan et al. [KMH+20] offered a promising outlook on performance improvements and provided the empirical framework for model scaling. This further motivated the pursuit of the ambitious yet speculative goal of discovering these emergent capabilities.

GPT-3 [BMR+20] serves as a pivotal example from this phase of investigations. With 175 B parameters and training on extensive web data corpora, it achieved a considerable increase in scale compared to its predecessors. A notable observation was its capabilities in few-shot learning scenarios, where a model has to demonstrate the ability to perform a novel task based on minimal examples provided within the input query. Furthermore, novel capabilities appeared to manifest. GPT-3 expressed competence in areas such as elementary mathematics, translation, and basic common-sense reasoning and surpassed the performance of smaller, task-specific models without explicit fine-tuning.

The development of GPT-3 occurred within the broader context of ongoing research. Projects such as Google's encoder-decoder family T5, which included models up to 11

B parameters, showed strong task generalization capabilities [RSR+20]. The research community was further driven by other large-scale, well-performing models at that time such as Megatron-Turing NLG model at 530 B parameters [SPN+22].

Table 2.1: **Illustrative Examples of Language Model Scaling (2018-2022).** Relevant language models released between 2018 and 2022, showing the trend towards larger parameter counts, especially for decoder-only architectures.

Model	Year	Approximate Parameters	Architecture
BERT-Base [DCLT19]	2018	110 M	Encoder-only
BERT-Large [DCLT19]	2018	340 M	Encoder-only
GPT-2 (Large) [RWC+19]	2019	774 M	Decoder-only
RoBERTa-Large [LOG+19]	2019	355 M	Encoder-only
GPT-2 (XL) [RWC+19]	2019	1.5 B	Decoder-only
T5-Large [RSR+20]	2020	770 M	Encoder-Decoder
T5-3B [RSR+20]	2020	3 B	Encoder-Decoder
T5-11B [RSR+20]	2020	11 B	Encoder-Decoder
GPT-3 [BMR+20]	2020	175 B	Decoder-only
Gopher [RBC+21]	2021	280 B	Decoder-only
Megatron-Turing NLG [SPN+22]	2022	530 B	Decoder-only
PaLM [CND+23]	2022	540 B	Decoder-only

2.3 Scaling Laws for Parameter Count, Data, and Inference Cost

The following section provides an overview of the scaling laws of LLMs – a set of empirical observations that describe the relationship between model size, training data, and performance.

2.3.1 Parameter Scaling Laws

The remarkable capabilities of modern LLMs mainly stem from their sheer size, a phenomenon first systematically documented by Kaplan et al. [KMH+20]. Their research identified two principal scaling relationships: one concerning model parameters and another related to computational resources. The parameter scaling law revealed a power-law relationship: performance consistently improves as the parameter count increases. This improvement follows a predictable pattern of loss reduction with model growth. Concurrently, they showed that performance also scales predictably with the computational effort invested during training. The foreseeable link between compute, parameters, and performance across different architectures offered crucial insights into

the learning dynamics of Transformers. It provided a framework to quantify the benefits of scaling and estimate a model's potential capability for a given compute budget.

The finding shifted research away from new architecture innovations and towards brute-force inspired scaling as the primary driver of performance improvements [BMR+20]. Major industry labs reacted by dedicating substantial resources to constructing increasingly larger models, with parameter counts now routinely reaching hundreds of billions and training datasets incorporating trillions of tokens from varied sources like web text, books, and code repositories [BMR+20, CND+23, TDFH+22]. Later, Hoffmann et al. [HBM+22] refined these initial findings with what became known as the *Chinchilla scaling laws*. Their work demonstrated that achieving optimal performance under a fixed compute budget requires balancing model size with the volume of training data. This insight led to a practical guideline suggesting models should train on approximately 20 tokens per parameter for maximum efficiency. Therefore, relying solely on these large models for all queries could be inefficient, which provides a strong motivation for adaptive strategies such as routing less demanding queries to smaller, more compute-efficient models while reserving the capacity of larger models for complex tasks.

2.3.2 Inference-aware Scaling Considerations

While the scaling laws described focused primarily on optimizing training loss within a fixed compute budget, recent research has broadened their scope to incorporate inference costs into the optimization calculus. For instance, Sardana et al. [SPDF24] explored scaling laws specifically concerning inference efficiency. Their findings suggest that when considering the total lifecycle costs including both training and deployment, optimal strategies often involve training smaller models for significantly longer durations than the Chinchilla guidelines recommend. This finding rationalizes the industry practice of *overtraining* – continuing training well beyond the point where standard metrics like perplexity stop showing substantial improvement.

Further supporting this idea, Gadre et al. [GSS+24] provided evidence that LLMs continue to gain capability on downstream tasks even when trained far past perplexity convergence. Their work confirms that such extensive training can yield practical benefits, even if it appears suboptimal from a purely training-centric perspective. One hypothesis is that this prolonged training allows models to internalize rarer data patterns or develop more nuanced capabilities that perplexity alone does not fully capture, which ultimately leads to better generalization in specific real-world applications. Models such as Llama 3 8B exemplify this approach, having been trained on approximately 2,000 tokens per parameter [GDJ+24] – surpassing the Chinchilla-optimal ratio by almost two orders of magnitude. This strong divergence between theoretical training optimality and practical deployment strategies hints at challenges in model development for real-world applications.

Adding yet another layer to scaling considerations, Kumar et al. [KAS+25] investigated the role of parameter precision alongside training data volume and parameter count. Through a comprehensive analysis of pre-training 465 language models across various

precision settings (ranging from 3 to 16 bits), they evaluated how both training-time precision and post-training quantization impact model quality. The key discovery was that lower-precision training acts as a predictable way to effectively reduce a model's parameter count. Their findings suggest that a precision of 7–8 bits might represent an optimal range across different model scales and compute budgets. Interestingly, they noted that overtrained models exhibit increased sensitivity to the performance degradation potentially caused by post-training quantization.

2.3.3 Model Families and Parameter Size Differentiation

Practical considerations derived from scaling laws insights have led to the development of *model families* that cover different parameter scales, usually built on top of the same architecture design and training methodology. For instance, the Falcon model family includes models with parameter counts ranging from 1.3B to 180B [AAA+23b]. Similarly, the BLOOM [LSFA+22] and OPT [ZRG+22] model families offer models ranging from 560M to 176B and 125M to 66B, respectively, based on consistent training methodologies.

Quantization efforts further extend this ecosystem. Methods like OPTQ [FAHA23] and AWQ [LTT+24] create even more efficiency-optimized alternatives by reducing precision post-training. The resulting *quantized* models often dramatically reduce memory and computational requirements by sacrificing performance only marginally. Altogether, this growth of model variants creates a *spectrum* of options with different efficiency and performance characteristics. The availability of this wide spectrum of models, ranging from large, high-capability versions to smaller, more efficient alternatives, provides the opportunity for systems to optimize inference through dynamic routing to select the most appropriate model for a given query and its requirements.

The variation in parameters serves practical purposes beyond academic interest. Organizations deploying and using LLMs must navigate resource and latency constraints while managing accuracy expectations, both heavily depending on the use case. A model suitable for cloud-based customer service might be entirely impractical for mobile devices or edge deployments with stringent response time requirements. Real-world inference systems balance multiple efficiency metrics against accuracy requirements:

- **Latency & Throughput:** User-facing applications often face tight latency expectations, and internal batch processing pipelines process high-volume text data such as company-wide knowledge bases or historic collections of documents.
- **Memory footprint:** VRAM limitations often dictate model deployment options.
- **Energy consumption:** Efficiency requirements create operational costs in local or cloud deployments, which also affect environmental impact of AI-based products throughout their deployment lifetime.

2.3.4 Query-Dependent Performance Variations

While model scaling laws describe average model capabilities across benchmarks, performance variations across different query types reveal opportunities for deployment optimization. Research has demonstrated that performance degradation from model size reduction or quantization is not uniform across tasks or queries. Certain benchmark categories show minimal performance degradation on smaller or quantized models while others require larger model capacities to maintain acceptable accuracy.

Researchers at OpenAI [AAA+23a] observed that GPT-4 significantly outperforms smaller models on complex reasoning tasks, but the performance gap decreases considerably for simple classification or factual retrieval. Touvron et al. [TLI+23] demonstrated similar patterns and find that well-trained smaller models can match larger ones on narrower domains when optimized appropriately. The benchmarking analysis by Srivastava et al. [SRR+22] across diverse LLM tasks consistently showed task-dependent performance patterns. Reasoning-intensive problems, for example, displayed steeper performance drops when model capacity decreased compared to simpler tasks. Such variations across query types indicate significant inefficiency inherent in static, one-model-fits-all deployment strategies. This observed heterogeneity, combined with the availability of a wide range of models resulting from scaling efforts (Section 2.3.3), presents a clear motivation for adaptive approaches, like the dynamic routing framework proposed in this thesis, to optimize resource utilization by sensibly matching query characteristics to the most suitable model capabilities.

2.4 Sustainability Challenges

2.4.1 Environmental Impact of Large Language Models

The rapid rise of LLMs has ignited serious concerns regarding their resource consumption and its climate impact. Although countless applications already integrate these powerful models, clear, standardized methods for comprehensively measuring their ecological footprint are still emerging. While carbon dioxide equivalents (CO_2e) often serve as the default metric, a full accounting must consider broader implications, including water usage, complex supply-chain emissions, and the generation of electronic waste [WARH24].

To better structure this analysis, some researchers propose distinguishing between operational and embodied carbon footprints. The operational carbon footprint (OCF) encompasses emissions generated during the active phases of a model's life: training, inference execution, and the associated data-center overhead such as power delivery infrastructure and cooling systems. Calculating this typically involves adjusting energy consumption figures by local grid carbon intensity and a facility's Power Usage Effectiveness (PUE) score. The embodied carbon footprint (ECF), conversely, aims to capture the environmental costs embedded in the hardware lifecycle, including upstream factors like GPU manufacturing and transportation, as well as downstream impacts related to disposal and recycling. Methodologies like Life Cycle Analysis (LCA) strive for an even

broader perspective by segmenting total emissions across four key stages: manufacturing, transport, operational usage, and end-of-life processing [WARH24, WRG+22].

Recent research compellingly indicates that generative AI models, particularly those optimized for high-dimensional tasks, consume significantly more electricity compared to traditional data-processing applications [Int25], placing escalating demands on global power grids already strained by increasing digitalization. The International Energy Agency (IEA) provides drastic figures: data centers consumed an estimated 415 TWh globally in 2024, representing approximately 1.5% of worldwide electricity usage. Alarmingly, this figure is projected to more than double by 2030, possibly reaching 945 TWh or nearly 3% of anticipated global electricity demand [Int25]. The IEA further highlights that the accelerated servers predominantly used for AI workloads are a main driver of this growth. Their energy consumption alone is expected to surge by 30% annually, which would account for almost half the net increase in total data center energy demand by 2030 [Int25]. In technologically advanced regions like the United States, data center consumption could surpass 1,200 kWh per capita by 2030 — an amount equivalent to roughly 10% of an average American household's yearly electricity use [Int25].

Both the initial training phase and the ongoing inference stage contribute extensively to these escalating energy trends. Some researchers estimate that training GPT-3 (175 B parameters) alone resulted in approximately 550 tCO_2e, an amount comparable to the annual electricity-related emissions of about 75 average U.S. households [UE25, PGL+21]. More recent flagship models, like the LLaMA-3.1-405B parameter variant, are estimated to have generated nearly 8,930 tCO_2e during their training processes [Met24]. Crucially, beyond these substantial one-time training emissions, the cumulative operational cost of performing inference over a model's lifetime represents a massive, and potentially even larger, long-term energy burden [PGL+21]. This potential dominance of cumulative inference costs reinforces the critical importance of optimizing the inference phase for sustainable AI deployment. Enhancing the efficiency of this stage is essential to mitigate the growing energy demands associated with widely deployed LLMs. Strategies such as developing model families with diverse sizes and employing techniques like quantization represent vital approaches for reducing this ongoing inference cost and its corresponding environmental impact.

Summary

In this chapter, we reviewed LLM fundamentals, scaling laws, and sustainability challenges. The key insights include the observation of query-dependent performance variations across the growing spectrum of models and the serious sustainability costs associated with inference, which together motivate optimization of inference for better efficiency. The next chapter explores related work on inference optimization in general and routing solutions in particular.

CHAPTER 3

Related Work

Building on the foundational concepts and sustainability concerns discussed in the previous chapter, this chapter investigates related work on efficient *inference* with LLMs. Inference can be described as the process of utilizing a pre-trained model to generate responses or predictions without the intent of tuning the model parameters like during model training. Given the significant resources this process requires (see Chapter 2), considerable research has focused on its *optimization* to improve efficiency without compromising output accuracy. While this chapter focuses specifically on inference optimization, it is part of the wider research effort towards sustainable AI and encompasses areas like efficient training algorithms and hardware development [WRG+22, SDSE20].

Optimizing LLM inference involves strategies across multiple abstraction levels. Understanding these diverse optimization strategies across data, model, and system levels provides essential context for inference routing, which aims to intelligently select among models potentially employing such optimizations, rather than optimizing a single model instance. Zhou et al. [ZNH+24] propose a taxonomy of optimization techniques that hierarchically group techniques into data-, model-, and system-level optimization. In the following sections, we will explore each of these categories, starting with general inference optimization techniques before focusing specifically on inference routing systems.

3.1 Overview of Inference Optimization Techniques

3.1.1 Data-level Optimization

The focus of data-level optimization is on reducing the computational load by modifying the data processed during inference. Techniques typically alter the input prompts or the data representation during inference.

Input Compression

The main goal of input compression is to shorten the effective input sequence length processed by the model, which directly reduces the computational cost. Common strategies include **prompt pruning**, which identifies and removes less relevant tokens or sections from the input prompt [JWL+23]. Another approach is **prompt summarization**, which uses a separate, more efficient model to condense the original prompt into a shorter summary which still captures the core intent and is then subsequently passed to the main LLM [XSC24]. Additionally, **soft prompt-based compression** utilizes embeddings trained to represent longer contexts or specific instructions in a much shorter input, which effectively compresses the semantic content of the input [MLG23]. **Retrieval-Augmented Generation (RAG)** retrieves relevant data snippets from an external knowledge base depending on the input query instead of relying solely on the LLM's internal parameters. The retrieved context is then added to the original prompt which helps with generating more factual, relevant, and up-to-date responses beyond the training cut-off date. RAG can indirectly improve efficiency as it enables smaller models, paired with retrieved knowledge, to accomplish tasks that might otherwise require much larger models [LPP+20]. Although not strictly shortening the input, it is sometimes categorized here as it selectively includes relevant external information instead of potentially much longer context [ZNH+24].

Output Organization

Strategies for output organization aim to accelerate generation by parallelizing parts of the process, rather than generating tokens purely sequentially. Techniques like Skeleton-of-Thought (SoT) [NLZ+24] first generate a high-level structure or skeleton of the response, then expand different parts concurrently to reduce end-to-end latency [ZNH+24].

3.1.2 Model-Level Optimization

Model-level optimization intends to optimize inference efficiency by influencing the underlying model structure. Specifically, it focuses on changing the model architecture or its parameters before deployment. Techniques are broadly categorized into efficient architecture design and model compression strategies.

Efficient Structure Design

Efficient architecture design focuses on developing more efficient model structures, usually in terms of computational complexity. Key innovations include the attention mechanism discussed in Section 2.1.1. Other notable approaches include improvements in calculating attention efficiently like Multi-Query Attention [Sha19] and Grouped-Query Attention [ALTdJ+23]. Both reduce memory requirements by sharing key K and value V projections across attention heads. Other techniques, such as sparse attention [CGRS19] and linear-complexity attention [KVPF20], intend to reduce the quadratic computational complexity of attention. Mixture-of-Experts (MoE) represents another important architectural

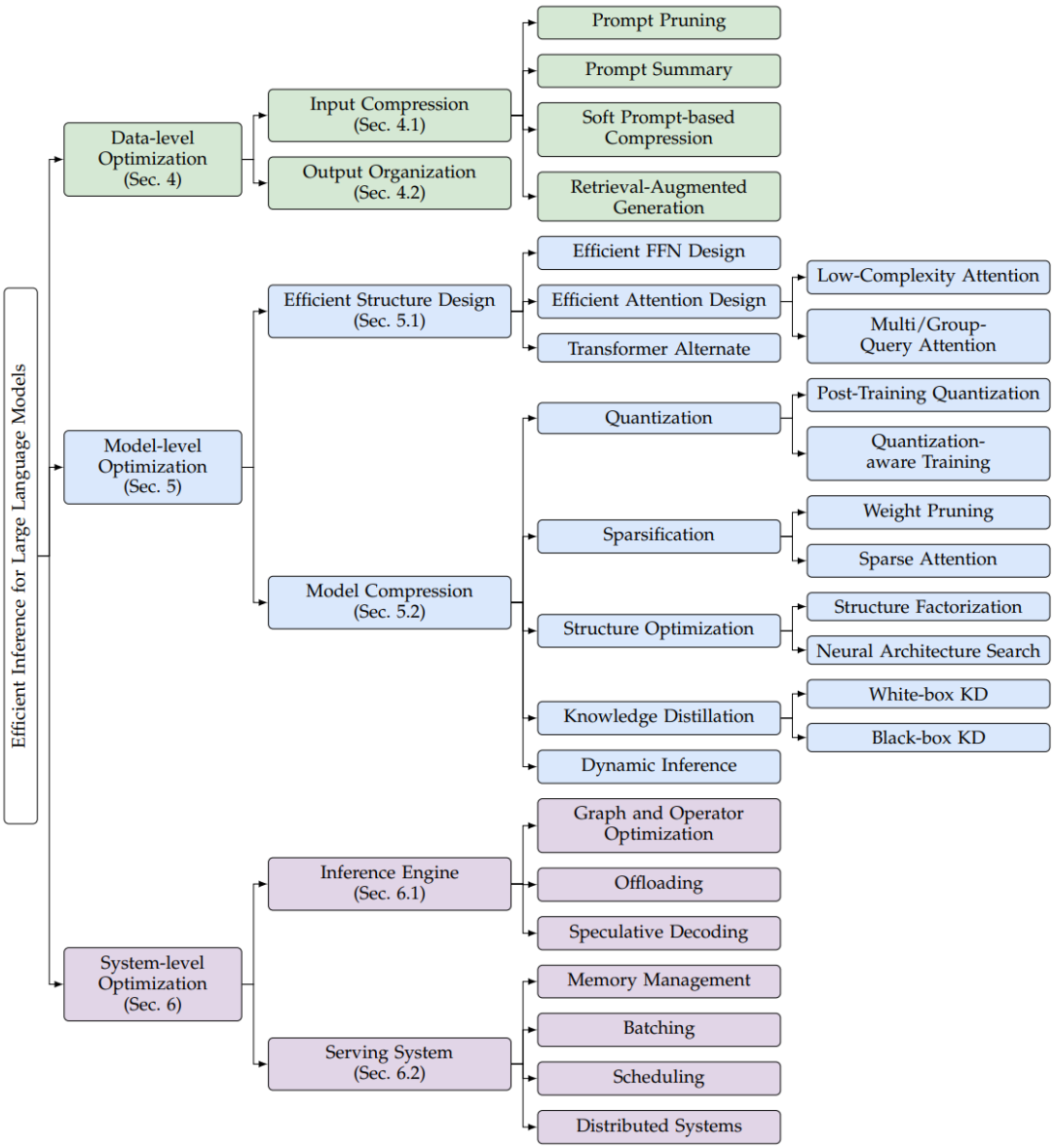

Figure 3.1: **Taxonomy of LLM Inference Optimization Techniques** [ZNH+24].

innovation, which reduces computational costs by activating only subsets of model parameters (*experts*) based on input tokens [SMM+17]. While similar, MoE differentiates itself from inference routing by focusing on the structure within a single model, thus, is inherently static. Furthermore, recent progress in non-Transformer architectures like State Space models such as Mamba [GGR22, GD24] reach performance comparable to Transformers with improved efficiency. However, as of now, non-Transformer models are not yet widely adopted outside of model-level research.

Model Compression

Model compression strategies reduce computational load and memory usage by converting pre-trained models into more efficient versions. Quantization is one of the most adopted techniques as it has established itself as a reliable method to reduce memory usage. At its core, it revolves around shifting high-precision weights and activations (typically 16- or even 32-bit) to lower precision (e.g., 8-bit, 4-bit) [FAHA23, XLS+23]. Sparsification increases zero-value elements, usually through weight pruning, that identifies and removes less important weights based on criteria such as magnitude or sensitivity [FA23]. Structure Optimization alters the model architecture itself, e.g., through low-rank factorization (LRF) [LYZ+23] which approximates weight matrices with smaller ones, or automated Neural Architecture Search (NAS) methods that try to identify more efficient model structures [ZL17]. Knowledge Distillation (KD) trains a more efficient model (*student*) to mimic the outputs of a superior, usually larger model (*teacher*) [SDCW19, HVD15]. Dynamic Inference techniques position themselves adjacent to inference routing as they also adapt the computation performed based on the input, however, they usually do not consider a pool of models with the task of selecting the most appropriate model before inference but rather focus on adapting during the execution of inference, e.g. through early-exiting or iteratively escalating model capabilities (*cascading*) [CZZ23].

3.1.3 System-Level Optimization

System-level optimization increases efficiency by improving how models are handled at runtime and managed within their deployment environment. It includes the development of serving systems, inference engines, and optimization of infrastructure components.

Inference Engine

Inference engines focus on the core computation during inference passes. At the operator level, we find methods improving typical operations executed during computations in neural networks (e.g., matrix multiplication, attention). In this regard, FlashAttention [DFE+22] represents a breakthrough by reducing memory overhead required for calculating attention through sophisticated tensor tiling and kernel design. Similarly, FlashDecoding [DHMS23] and FlashDecoding++ [HDX+23] optimize the decoding part with the latter implementing reducing dimensions through specialized linear operations. Inference engines such as TensorRT-LLM [NVI24], DeepSpeed [ARA+22], and vLLM

[KLZ+23] apply comprehensive graph-level optimizations that greatly reduce inference time via techniques. Speculative decoding [LKM23] represents another promising direction. It effectively allocates computational costs for multiple token generations by using smaller models to predict multiple tokens which are subsequently only verified by the superior model.

Serving System

Serving system optimizations improve the handling of multiple concurrent inference requests and coordinate inference across (distributed) infrastructure. Advanced memory management techniques like paged attention [KLZ+23] have been shown to improve GPU utilization, while batching methods increase throughput [YJK+22]. For large scale deployments, optimization for distributed systems [PCZ+24] and split computing patterns for cloud or edge environments [MSD+24, TYL+23] are used to manage resources effectively.

3.1.4 Limitations of Existing Methods

The optimization techniques discussed in Sections 3.1.1, 3.1.2, and 3.1.3 have proven to be effective in reducing inference cost and latency in various aspects. However, despite their individual strengths, they often operate under a static model paradigm that applies the same efficiency strategy (e.g., quantization level, batching method) to all queries, usually still bound to a single, pre-selected model. This means these techniques improve efficiency within a given model or system setup, but they generally do not address the problem of selecting the most appropriate model from an evolving heterogeneous pool based on individual query requirements. As discussed in 2.3.4, different query types do not uniformly profit from a single, globally optimized setup. Some tasks demand complex reasoning or domain-specific knowledge, which might require the full capacity of a state-of-the-art LLM while more efficient models could handle simpler tasks effectively. This inherent limitation of static, single-model optimization strategies motivates the exploration of dynamic inference routing, which aims to select the most appropriate model from a heterogeneous pool on a per-query basis. To explore existing solutions to this problem, the next section discusses routing approaches.

3.2 Inference Routing

Inference routing solves a key problem in LLM deployment through selective model assignment for each query. It revolves around mapping input requests to specific models from a heterogeneous pool that consists of various parameter sizes, architectures, or optimization levels. This results in allocating computational resources according to each query's necessity.

In current research, routing systems show great variation in decision-making logic and performance targets. Resource-aware routers put emphasis on hardware metrics and

queue optimization to achieve maximum throughput. In contrast, accuracy-focused methods monitor benchmark scores across different task types to maintain strict quality standards. Many publications discuss frameworks that detect specific task categories and then direct these tasks to models with domain specialization. Recent advancements also explore probabilistic selection schemes that account for uncertainties in both query characteristics and model performance prediction. Overall, the research area shows an evolution from basic static policies towards more sophisticated frameworks, e.g. by incorporating both infrastructure status and request properties.

3.2.1 Dimensions of Routing Systems

For systematic analysis of routing approaches, we define seven key dimensions that describe their core design aspects and capabilities. These dimensions, consolidated from recurring themes and differentiators in the literature (summarized in Tables 3.1 and 3.1), represent characteristics that allow a structured comparison of routing systems. The classification is based on how methods were evaluated and discussed in their original research context, but many could arguably be extended with minor modifications.

Decision Timing (DT) specifies when route selection happens during inference. Pre-inference routing chooses a model before any computation starts, relying solely on query features to predict the most appropriate model. It usually aims for minimal overhead but cannot access actual model performance on the specific query. Mid-inference routing first processes inputs through one candidate model, then decides to continue or switch to a larger model based on intermediate results or confidence values, e.g., FrugalGPT [CZZ23]. This approach allows for more precise decisions by considering preliminary processing results but creates redundant computation. To maintain focus on lightweight routing mechanisms without the computational overhead of multiple inference operations per query, this review primarily analyzes pre-inference systems.

Router Calibration Time (RCT) describes when the decision logic of the actual router gets optimized. Pre-deployment (PD) approaches set routing parameters before system activation using past data. They work immediately at runtime but do not adapt well to new patterns. Runtime (RT) calibration keeps updating during operation using feedback from actual inference results. Many practical systems use a hybrid (HY) approach. They start with some degree of offline profiling but make selective adjustments online. The RCT dimension determines how systems balance immediate performance with long-term adaptation ability. We observe that pre-deployment calibration gives consistent initial behavior but often struggles with distribution shifts and adaptiveness while runtime approaches improve over time but may perform worse initially due to uncertainty and cost of exploration.

Adaptation Capability (AC) reflects how routing systems evolve with new information. Fixed Policy (FP) approaches keep the same decision rules after training, which makes behavior predictable but limits flexibility. Parameter Updates (PU) systems allow adjustments to specific components while keeping their core framework. Policy Evolution

(PE) methods continuously update their entire selection strategy through online learning algorithms. This capability determines how systems improve through experience and adapt to changing query patterns. We find this dimension especially important for deployments with diverse or evolving workloads, where initial assumptions about query distribution might become invalid over time.

Model Integration (MI) describes how easily routing systems handle changes to the model pool. Locked (LK) architectures need complete recalibration when adding or removing models. This creates significant operational overhead. Extensible (EX) frameworks support new models through partial calibration. Dynamic (DY) approaches use mechanisms that work with unseen models with marginal or no recalibration. MI grows more important as LLM ecosystems develop rapidly. New, more capable models appear constantly while others become outdated. A system's integration capabilities directly impact its practical usefulness in real-world applications as model availability changes frequently and staying current with state-of-the-art models is important.

Input Representation (IR) defines how routing systems convert raw queries into structured formats. The existence of diverse input representation techniques reflects the need to address the query-dependent performance variations observed in LLMs (see Section 2.3.4) to effectively capture relevant input characteristics. Attribute-based (AB) methods extract key metadata features (e.g., query length, keyword presence) that help characterize the query and provide a compact representation for the router to work with. Embedding-based (EB) approaches transform queries into dense semantic vectors that maintain relationships between words and concepts. In our analysis, we notice many systems now implement hybrid (HY) strategies. They combine feature extraction with embeddings to get benefits from both. This choice affects computational requirements and how accurately the router can match queries to models. Different approaches make different trade-offs – richer representations often need more computational resources and lack interpretability but enable more granular routing decisions.

Model Representation (MR) covers how routers encode information about available LLMs. Attribute-based (AB) approaches use simple model properties like size and architecture type. AB offers simplicity and interpretability but may miss subtle performance nuances between models. Embedding-based (EB) methods build vectors based on a learned embedding model that, for instance, captures how models perform on certain inputs. The vectors allow for the externalization of latent patterns that basic attributes cannot show, but often require more complex calibration and are usually less interpretable. The representation approach affects a router's ability to adapt to new models without extensive recalibration, with EB approaches usually requiring more elaborate pre-deployment processing for better training (or fine-tuning) of the embedding model.

Optimization Objective (OO) defines what metrics the router tries to maximize. Quality-Cost (QC) approaches balance response accuracy and secondary objectives like resource usage and latency simultaneously. Budget-Constrained (BC) systems work within strict resource or latency limits while trying to maximize quality. Hybrid (HY) objectives

combine QC, but treat certain metrics as budget constraints. The choice here mainly reflects the deployment considerations.

3.2.2 Trends in LLM Routing Systems

Analysis of the routing systems presented in Tables 3.1 and 3.1 reveals several noteworthy patterns that characterize the current state of routing research for LLM inference. When examining existing systems through our dimensional framework, we observe clear developments that show both limitations and opportunities in this rapidly evolving field.

Router Calibration, Adaptation Patterns & Model Integration

Most existing routing systems use pre-deployment calibration with fixed policies. This indicates a preference for stable, predictable behavior over runtime adaptability. This approach gives immediate performance benefits but sacrifices long-term adaptability to shifting workload pattern and assume prior knowledge of model performance. From 14 systems we analyzed, 13 implement pre-deployment calibration, with only LLMBandit [Li25] and MixLLM [WLC+25] adopting a runtime approach.

This pattern reveals an important limitation: once deployed, most routers cannot respond to distribution shifts in query patterns or an evolving model landscape. Stable policies provide predictability but create weaknesses to changes in real-world deployments. As LLM applications expand and settings become more dynamic with evolving user needs, this limitation may become more and more significant.

Adjacent to these observations, we see a clear trend in model integration capabilities from early locked architectures toward more flexible approaches. Early systems (2023-early 2024) predominantly used locked configurations that require extensive recalibration when the available candidates in the model pool change, although some extensible systems like Tabi [WCTG23] and RouteLLM [OAW+25] emerged during this period. Newer approaches like EmbedLLM [ZWW+25], GraphRouter [FSY25], LLMBandit [Li25], and MixLLM [WLC+25] increasingly implement extensible frameworks that support partial recalibration. Universal Model Routing [JNR+25] stands out by introducing a dynamic approach that can handle unseen models with minimal adjustment, though their learned model representation will likely need complete recalibration once the new model characteristics drift further from existing ones.

In general, this evolution shows growing recognition that the LLM ecosystem constantly evolves. New models with improved capabilities are released, and older models become effectively outdated. Fixed routing architectures create significant operational overhead as they require frequent retraining to maintain alignment with available models.

Input and Model Representation

Input representation approaches show a strong preference for embedding-based (11) or hybrid (1) approaches over purely attribute-based ones (3). This preference comes from

embedding models' ability to capture semantic relationships between words compared to attribute-based systems. However, some systems like TABI [WCTG23] effectively use attribute-based methods that extract specific features from queries, especially for specialized deployment scenarios.

For model representation, we find a relatively balanced distribution between attribute-based (7) and embedding-based (8) approaches. Early systems mainly use attribute-based representations. Used components are size, architecture type, and quantization level. This approach offers simplicity but may fail to capture subtle performance differences. Recent systems increasingly adopt embedding-based model representations. They usually characterize models through their performance on representative query sets.

The transition to embedding-based model representations offers routing systems a potential advancement, though potentially with trade-offs. It allows routing systems to distinguish between performance patterns more granularly, rather than relying solely on predefined attributes. Systems like EmbedLLM [ZWW+25], Universal Model Routing [JNR+25], and LLMBandit [Li25] demonstrate certain benefits of embedding-based representations for generalization to new models. However, these approaches often require more complex processing pipelines and larger calibration datasets than attribute-based methods. Attribute-based representations remain valuable when computational efficiency or interpretability are prioritized.

Optimization Objectives

Quality-cost tradeoff optimization dominates with 13 of 15 systems targeting simultaneous optimization (QC). Only TABI [WCTG23] and OrchestraLLM [LCO24] implement strictly budget-constrained (BC) approaches. Clearly, the chosen optimization objective is mainly a question of research design and can potentially be interchangeable. Nonetheless, it shows that research is actively favoring flexible resource allocation that adapts to quality requirements rather than strict constraints.

Collectively, these observed trends indicate a heavy reliance on pre-deployment calibration and fixed policies (RCT=PD, AC=FP). This causes significant challenges for effectively addressing the core practical LLM routing that motivates this work. Static approaches struggle to optimally adapt to the complex, query-dependent trade-offs between accuracy and efficiency (RQ1) in real-time. Furthermore, evaluating efficacy under realistic operational conditions (RQ3) becomes difficult if the system cannot adapt online, as it would require frequent retraining and redeployment. The observed limited extensibility of many model pools further lessens the practical benefit derived from the diverse LLM ecosystem. These persistent gaps emphasize the need for routing frameworks capable of *online learning*, effectively handling the uncertainty stemming from *limited real-time information*, and explicitly balancing the *exploration* of model capabilities with the *exploitation* of incrementally gathered knowledge. Multi-armed bandit algorithms, explored in detail later in this work (see Section 4.5), offer a robust theoretical foundation and practical algorithms for addressing precisely these requirements.

3.2.3 Analysis of Dynamic Routing Systems

Our broad analysis reveals that especially earlier systems often rely on offline calibration and are limited in their adaptability. However, more recent approaches recognized these shortcomings and propose more sophisticated mechanisms. In the following section, we examine the three most dynamic routing systems in detail, and conclude with persistent gaps still present in these improved systems.

Universal Model Routing

Universal Model Routing [JNR+25] presents a cluster-based approach for handling unseen LLMs at test time (MI=DY) without retraining the router itself. The system first represents each available LLM (both training and unseen test models) by a feature vector derived from its prediction correctness on a small, shared set of validation prompts (*correctness vector*). To reduce overfitting and improve generalization, it then proposes grouping prompts into clusters (using a large unlabeled training set) and representing each LLM by its average error rate within each cluster. Routing for a new query identifies its cluster and selects the LLM with the lowest expected cost-adjusted error for that cluster. However, a critical limitation is its reliance on pre-deployment calibration (RCT=PD, AC=FP). The clustering and related error rates are determined offline and are not updated based on online performance, which limits adaptiveness to potential shifts over time.

LLMBandit

LLMBandit [Li25] introduces a dynamic routing system that focuses on user preferences for the accuracy-cost trade-off. Its offline preparation involves two main stages. First, an Item Response Theory (IRT) based model is trained to generate *model identity vectors* that capture model capabilities and characteristics from evaluation data. Second, a contextual routing policy via reinforcement learning (RL) using Proximal Policy Optimization (PPO), which uses these identity vectors, query information, and a user-provided preference parameter ω as input to determine routing decisions. This offline training allows the policy to learn complex routing strategies (PE). At runtime, the system adapts based on the user's specified preference parameter ω and conditions the behavior of the pre-trained policy. A key feature is its ability to incorporate new models (MI=EX) by computing their identity vectors through a *quizzing* process (evaluating on a small subset of prompts), without retraining the core routing policy. Despite the bandit framing for the selection problem, the presented work relies on offline RL training to determine the reward and continuous online learning using live feedback is mentioned as future work [Li25]. The routing policy parameters remain fixed after the offline phase. Runtime adaptation is achieved through the reusable representations and preference conditioning.

MixLLM

MixLLM [WLC+25] uses a dynamic contextual bandit approach with hybrid learning capabilities (RCT=HY, AC=PE). Initial offline training establishes *independent*, LLM-specific predictors for quality and response length (cost), which are subsequently refined through online updates using performance signals. The system introduces tag-enhanced query embeddings (IR=HY) for improved context understanding and explicitly incorporates mechanisms to penalize predicted latency and account for prediction uncertainty within its quality-cost optimization objective (OO=QC). A main distinction from energy-focused methods lies in its cost component, which relies on external pricing structures such as API costs rather than direct physical measurements like energy consumption. While this design with independent predictors allows for extending the model pool by training only new predictors (MI=EX) and supports continual learning, its cost optimization remains tied to potentially fluctuating external metrics, which do not guarantee to reflect actual resource usage [WLC+25].

3.2.4 Research Gaps in Existing LLM Routing Systems

Early work treated routing as a static optimization problem. After collecting a calibration set and fitting a classifier that predicts the optimal model for each prompt, freeze the policy before deployment and re-use it for all incoming requests. Fixed-policy routers of this type, e.g. Tryage [HT23], RouteLLM [OAW+25], Hybrid LLM [DMW+24] and the original RouterBench [HBL+24] baselines, established the usefulness of selective execution but share three structural weaknesses that remain only partially resolved by recent dynamic systems.

1. **No continual learning.** Except MixLLM [WLC+25], virtually all routers are strongly calibrated once and thereafter operate unchanged or only slightly nudged by configuration parameters. This design commonly eliminates exploration overhead but leaves the system blind to distributional drift in the incoming query stream or to gradual degradation of relative model performance. While MixLLM [WLC+25] shows some online adaptation, LLMBandit [Li25] presents a preference-conditioned policy that learned via PPO whose weights are frozen at the inference stage, only allowing a scalar user preference to be changed on-the-fly. Universal Model Routing [JNR+25] is even more static as its cluster map is computed offline and never updated. In environments where patterns evolve regularly, such rigidity might be a significant limitation.

2. **Reliance on proxy cost metrics.** Most studies optimise a quality-cost objective, yet the cost term is usually a proxy: public API price, a synthetic token budget, or even parameter count. These proxies correlate inconsistently with the resources that dominate production budgets such as GPU utilization, energy draw, and latency. While MixLLM [WLC+25] takes a step further by adding a waiting-time penalty, its actual cost signal is still derived from external price information rather than

measured power or similar metrics. LLMBandit [Li25] relies on pre-calibrated cost estimates as well. We argue that a router that cannot observe its actual resource footprint cannot guarantee to comply with efficiency constraints in practice.

3. **Calibration overhead for every new model** The LLM landscape changes rapidly, with new models of all sizes appearing on a regular basis. For instance, the number of models specifically tagged for text generation on the Hugging Face Hub grew from approximately 15,000 in July 2023 to over 220,000 by May 2025[1]. A truly practical router must integrate new models efficiently, ideally without requiring a cumbersome per-model calibration or extensive profiling phase. Recent solutions reduce this overhead compared to early fully static approaches, but often do not eliminate it entirely. Even the most dynamic systems reviewed require some model-specific onboarding: Universal Model Routing [JNR+25] requires prompt quizzing to derive a model-identity vector, LLMBandit [Li25] uses a smaller set of prompts but still demands labeled examples, and MixLLM [WLC+25] requires training a dedicated prediction model for each new LLM. While the effort varies, none of these dynamic approaches currently accept a completely unseen LLM without at least some model-specific effort, which limits seamless adaptation and capitalizing on the dynamic LLM ecosystem.

Consequently, integration of completely unseen models still requires a degree of model-dependent profiling or retraining. These limitations point to a clear research gap that an effective router must be able to handle unseen models without offline calibration, and update its policy dynamically online with partial feedback based on measured, relevant efficiency metrics (like energy consumption) that reflect actual inference costs. Chapter 4 formalizes different aspects of the routing problem. Chapter 5 then presents a routing architecture that addresses these requirements using online learning multi-armed bandits.

Summary

The review of existing LLM inference optimization and routing techniques indicates significant efforts and progress but also reveals persistent gaps, especially regarding adaptability and practical deployment. Numerous methods improve efficiency, however, many rely on static configurations or extensive offline calibration which limits their adaptability. Even recent dynamic routing systems appear to not fully adhere to paradigms like continual online learning, integration of direct resource metrics, and seamless integration of entirely new models without per-model overhead.

This thesis aims to address these shortcomings and proposes an adaptive inference routing framework with the following key characteristics:

[1] Approximate model counts retrieved from huggingface.co/models (filtered for text-generation) via direct access (May 2025) and the Internet Archive's Wayback Machine (May 2024, July 2023).

- **Online Learning and Adaptation:** Utilize contextual multi-armed bandits to continuously learn and adapt the routing policy based on real-time feedback (accuracy and measured energy) to enable responsiveness to model performance characteristics without offline retraining.
- **Direct Energy Optimization:** Incorporate actual energy consumption measurements into the optimization objective to move beyond proxy metrics like API costs or token counts and better align routing decisions with actual resource usage.
- **Reduced Calibration Overhead:** Design to eliminate the need for extensive offline calibration or per-model profiling when integrating new, initially unknown models into the candidate pool to draw benefits from the diverse and vast LLM ecosystem.

By contextual multi-armed bandit algorithms, this thesis intends to provide a more practical, scalable, and adaptive solution for context-aware, efficient LLM inference. The following chapters detail the formal problem definition (see Chapter 4) and the applied methodology (see Chapter 5) designed to achieve these goals.

Table 3.1: **Categorization of Routing Approaches for LLM Inference (Part 1).** This table categorizes relevant LLM inference routing systems based on shared design dimensions defined in Section 3.2.

Note: RCT=Router Calibration Time, AC=Adaptation Capability, MI=Model Integration, IR=Input Representation, MR=Model Representation, OO=Optimization Objective.

Publication	Year	Description	RCT	AC	MI	IR	MR	OO
Tryage [HT23]	2023	Classification-based query router using BERT embeddings	PD	FP	LK	EB	AB	QC
RouteLLM [OAW+25]	2024	Preference-data-based router using matrix factorization	PD	FP	EX	EB	EB	QC
TABI [WCTG23]	2023	Multi-level inference system with complexity-based routing	PD	FP	EX	AB	AB	BC
Hybrid LLM [DMW+24]	2024	Cost-efficient router using meta-learning for query complexity estimation	PD	FP	LK	EB	AB	QC
RouterBench [HBL+24]	2024	Benchmark framework for multi-model routing with diverse query types and routers	PD	FP	LK	EB	AB	QC
Smoothie [GCC+24]	2024	Label-free router using embedding-based similarity comparison	PD	FP	LK	EB	AB	QC
RouterDC [CJL+24]	2024	Query-based router using dual contrastive learning for assembling LLMs	PD	FP	LK	EB	EB	QC
OrchestraLLM [LCO24]	2024	Efficient orchestration of language models for dialogue state tracking	PD	FP	LK	EB	AB	BC

Table 3.1: **Categorization of Routing Approaches for LLM Inference (Part 2).** This table continues the categorization of LLM inference routing systems. It highlights more recent approaches and their characteristics along the defined dimensions, moving towards more dynamic and practical routing systems.

Note: RCT=Router Calibration Time, AC=Adaptation Capability, MI=Model Integration, IR=Input Representation, MR=Model Representation, OO=Optimization Objective.

Publication	Year	Description	RCT	AC	MI	IR	MR	OO
Automix [AMA+24]	2024	Multi-task mixture-of-experts approach for automated model selection	PD	FP	LK	AB	AB	QC
EmbedLLM [ZWW+25]	2024	LLM embedding framework for compact representation of model capabilities	PD	FP	EX	EB	EB	QC
GraphRouter [FSY25]	2024	Graph-based router using task and model connectivity for LLM selection	PD	FP	EX	EB	EB	QC
TensorOpera [SXH+24]	2024	Router using k-NN (k-Nearest Neighbors) for efficient LLM inference	PD	FP	LK	EB	AB	QC
Universal Model Routing [JNR+25]	2024	Router using cluster-based representation for unseen test-time LLMs	PD	FP	DY	EB	EB	QC
LLMBandit [Li25]	2025	Cost-efficient LLM generation via preference-conditioned dynamic routing	HY	PE	EX	EB	EB	QC
MixLLM [WLC+25]	2025	Dynamic contextual-bandit-based routing system for query-LLM assignment with tag-enhanced embeddings	HY	PE	EX	HY	EB	QC

CHAPTER 4

Definitions and Problem Formulation

This chapter formalizes the routing problem in which we aim to dynamically assign incoming requests to a single model from a pool of heterogeneous LLMs while balancing two primary objectives: *accuracy* and *efficiency*. Section 4.1 introduces the general problem setting and fundamental notations. Section 4.2 explains how we can represent a query as a feature vector. Section 4.3 and 4.4 define the core performance metrics and how to balance them into a single multi-objective optimization problem (MOOP). Section 4.5 shows how the elaborated routing problem based on input features, and partial feedback structure leads to a *contextual multi-armed bandit* problem that will serve as the basis for our solution approach.

4.1 Problem Overview and Notation

We consider an inference system that processes a sequential stream of user queries $\{q_t\}_{t=1}^{T}$. Each query q_t arrives at a discrete timestep $t = 1, 2, \ldots, T$, where T is the total number of queries.

Each query q_t may vary in specific characteristics (e.g., task type, text complexity), and thus, may require different levels of model capacity to be answered effectively. We assume access to a pool of K heterogeneous candidate LLMs:

$$M = \{m_1, m_2, \ldots, m_K\}.$$

Each $m_k \in M$ is a model with distinct characteristics. Possible characteristics include (i) architecture and parameter count, (ii) fine-tuning on domain (e.g., medical, legal), and (iii) quantization level (e.g., 8-bit, 4-bit precision). When query q_t arrives, the system's

routing policy π selects exactly one model m_t for inference:

$$m_t = \pi(q_t).$$

The policy aims to balance two key objectives – accuracy and efficiency – based on measured historical performance of each model in related contexts. The following sections specify how we represent the query context, define and measure accuracy and efficiency, and condense their trade-offs into a MOOP.

4.2 Feature Space and Query Representation

First, we focus on how to transform each query q_t into a structured feature vector to enable informed routing decisions. To convert the raw query text of q_t and its metadata into a structured, explicit *context vector* x_t, we define a mapping:

$$x_t = f(q_t) \in X,$$

where X is the *context space*. Features in the context space can be:

- **Task Type:** Identifying whether the query is for summarization, question-answering (QA), machine translation, code generation, commonsense and mathematical reasoning, creative writing, etc., can strongly influence which model in M is best for q_t (see Section 2.3.4).
- **Domain:** Certain models excel in specialized domains (e.g., legal, medical). Thus, encoding the semantic domain of q_t may help the system to select a better model.
- **Linguistic Complexity:** Features like readability metrics, average sentence length, or perplexity can hint at whether a simpler or more powerful model might be necessary.
- **Input Sequence Length:** Models vary in how efficiently they handle long inputs (context window), so including input length is crucial to sort out unsuitable models.
- **Additional User Constraints:** A user might favor certain objectives (high accuracy, fast response time) over others. Incorporating the possible inclusion of preferences can improve practicality and impact of our optimization system.

The necessity for such differentiation between queries arises directly from the significant heterogeneity observed in LLM performance across varying inputs (see Section 2.3.4). Comprehensive studies involving benchmarks consistently demonstrate this as they show that only state-of-the-art models may achieve high accuracy on certain complex tasks, but the performance gap often narrows considerably for simpler tasks like summarization or translation, where more efficient models can suffice. Consequently, context-agnostic routing strategies that treat all queries identically appear fundamentally suboptimal and

risk either the utilization of computationally expensive models for simple requests or the failure to meet accuracy requirements for complex ones. The focus in prior inference routing research (see Chapter 3) on extracting and representing query characteristics further underscores the widely acknowledged importance of leveraging context for improved decision-making.

Therefore, building upon these insights, we propose to represent each query q_t as a structured *context vector* x_t. The features chosen for x_t are selected based on existing literature and empirical observations that indicate predictive value regarding LLM inference outcomes. In principle, the feature extraction function f can range from simple methods like token counting to more complex approaches like dedicated characteristic-focused embedding models. Specifically, for the experiments in this thesis, we extract task type, semantic domain, and text complexity to inform the routing decision. The set of features, and thus the dimensionality of x_t, could potentially be expanded over time if new relevant query characteristics are identified.

4.3 Metrics

Having established the representation of query context x_t, we now define the criteria used to evaluate model accuracy and efficiency. Since the routing policy aims to balance the accuracy of the model's output against the energy consumed, we require metrics that quantify both and are measurable in an online setting. The following subsections detail these core metrics.

4.3.1 Accuracy

We denote $A_m(q_t)$ as the *accuracy* of model m's response to query q_t. Defining a single accuracy metric across all LLM tasks is challenging. For tasks with clear ground truth, objective metrics like **Exact Match** (EM) [RZLL16], **ROUGE** [Lin04], or **BLEU** [PRWZ02] are applicable. However, evaluating open-ended generation often requires subjective assessments (e.g., user feedback), which are difficult to automate and simulate reliably [CWW+24]. Therefore, for deterministic evaluation within the scope of this thesis, we focus exclusively on tasks where accuracy can be measured objectively against an available ground truth using such unambiguous metrics. We assume $A_m(q_t)$ is normalized to the interval $[0, 1]$, where higher values indicate a higher accuracy.

4.3.2 Energy Consumption

Efficiency in our setting refers to the effective utilization of computational resources to generate responses. It can be assessed though different aspects such as processing speed, memory footprint, or energy consumption (see Section 2.4). Among these factors, energy consumption stands out as a traceable and broadly relevant indicator for evaluating efficiency. Generally, the more efficient a system, the less energy consumption is required

to produce responses of comparable quality, given the same hardware and software configuration.

Thus, energy consumption will serve as our proxy metric to judge efficiency. It can be formally defined by integrating the *instantaneous power draw* $P_m(t)$ of model m over inference duration $T_proc(m, q_t)$:

$$C_m(q_t) = \int_0^{T_{\text{proc}}(m,q_t)} P_m(\tau)\, d\tau$$

.

The integral captures the true energy consumption, however, practical measurement limitations (detailed in Chapter 6) require approximations, which form the basis for the experimental evaluation of $C_m(q_t)$. The shown general formulation captures the fluctuations in energy usage across different phases of inference (e.g. data transfer, computation, memory operations). However, in practice, accurately obtaining such fine-grained measurements is impractical due to hardware monitoring limitations (e.g., costly external energy meters), or workload execution on shared server resources, especially in large-scale or real-time settings. Within our theoretical MOOP, $C_m(q_t)$ thus defines a theoretical measure of efficiency that must be approximated to allow experimental evaluation of our proposed solution.

4.4 Problem Formulation

This section extends our discussion of accuracy and energy consumption into a formal *multi-objective* routing problem. Building on the introduced definitions, we formalize the inherent conflicts between model quality and resource utilization. Additionally, we discuss how latency differs from other efficiency aspects and how to effectively incorporate it into the laid out MOOP. Furthermore, we introduce the concept of Pareto optimality, and how multiple objectives can be reduced to a single scalar.

4.4.1 Latency Constraint - Feasibility

Total latency represents the time from query submission to response and generally, captures any delays due to data transfer, optimization-related processes, actual inference procedure, and queuing. To simplify system evaluation, we consider latency aspects for inference optimization $L_{opt}(q_t)$ and inference processing time itself $L_m(q_t)$:

$$L_{\text{total}}(m, q_t) = L_{opt}(q_t) + L_m(q_t).$$

In a real-world deployment, additional latency components such as client-server-communication and queuing delay may introduce strong deviations through geographic proximity between user and service provider, request concurrency, and dynamic infrastructure extension through load-balancing mechanisms difficult to replicate given the scope and resources available for our work (further discussed in Section 6). Like energy consumption, latency

can be difficult to measure precisely and assign when resource usage fluctuates over time. Nevertheless, it remains a fundamental metric in scenarios where users expect responses within strict time bounds.

While both accuracy A_m and energy consumption C_m contribute continuously and directly to the trade-off optimization process. Latency L_m, however, is handled differently to simplify the MOOP. Instead of treating it as a third optimization objective, we argue latency has two key implications:

1. **Increased Resource Utilization:** Depending on the execution step in the inference pipeline, prolonged latency may indicate higher resource utilization (e.g., extended GPU execution time). However, this notion is already accounted for through direct energy consumption measurements and thus, makes optimizing for low latency redundant.

2. **User Dissatisfaction:** Extensive latency may negatively impact user experience. However, the relationship between latency and perceived quality degradation is not linear. Users typically tolerate latency up to a certain threshold, beyond which satisfaction declines sharply [Nah04]. To reflect this, we introduce *maximum acceptable latency* L_{max} which ensures that only models that are expected to meet this constraint are considered for inference.

A consequence of treating latency as a hard constraint is that the system won't actively minimize it below the threshold L_{max}; instead, it focuses on the accuracy-energy trade-off among the feasible models. To formally incorporate this constraint, we define the set of feasible models for query q_t as:

$$M_t^* = \{m \in M | L_m(q_t) \leq L_{max,t}\}.$$

Analogous to established techniques for handling time requirements in systems with Quality-of-Service (QoS) constraints [ZBN+04], a model exceeding $L_{max,t}$ is considered *infeasible* and is discarded in the candidate selection at time step t. By enforcing a hard latency constraint instead of treating latency as a separate optimization objective, we maintain practical usability constraints while also reducing problem complexity. This leads to effective routing decisions without unnecessary trade-offs between latency and other objectives.

4.4.2 Pareto Optimal Solutions

In multi-objective optimization, *Pareto optimality* [MA04] (also called *Pareto efficiency*) helps formalize what it means to achieve optimal balance between competing objectives. Generally speaking, it describes a state where no criterion can be improved without making another worse. Crucial to Pareto optimality is the concept of *dominance*:

A model m *dominates* another model m' in q_t if at least one of the following inequalities is strict

$$A_m(q_t) \geq A_{m'}(q_t)$$
$$C_m(q_t) \leq C_{m'}(q_t).$$

Intuitively, m is at least as good as m' in every dimension while being strictly better in at least one. A model m is called *Pareto Optimal* or *non-dominated* if there is no other $m' \in M$ that dominates it.

This leads us to the concept of the *Pareto front*, which is the subset of all non-dominated models. Models on this front are all relevant to specific user groups, depending on whether they value higher accuracy or lower energy consumption. From a theoretical standpoint, if we had complete information of each model's performance in our three key metrics, we could simply choose from the Pareto to accommodate the user-specific priorities. However, in a dynamic online setting with *incomplete information*, we require an adaptive selection and learning mechanism that is able to approximate Pareto optimality.

4.4.3 Objective Scalarization

The introduced multi-objective optimization settings allows us to consider multiple goals simultaneously, however, many learning-based decision-making systems still require a single numerical value to compare and conclude choices. *Scalarization*, specifically the *Weighted Sum Method* [MA04], presents itself as a well-established method that transforms multiple objectives into a unified score. Additionally, it easily integrates the discussed feasibility (latency) constraint. For our proposed routing problem, we apply scalarization to combine accuracy and energy consumption into a single reward function. For a query q_t, we generally define *reward* as follows:

$$r_t(m, q_t) = \alpha A_m(q_t) - \beta C_m(q_t), \quad \text{subject to } m \in M_t^*.$$

We parameterise the weights as $\alpha = 1 - \lambda$, $\beta = \lambda$ with $0 \leq \lambda \leq 1$, so $\lambda = 0$ results in an accuracy-only policy and $\lambda = 1$ in an energy-only policy. This provides a straightforward way to combine objectives but implicitly assumes a constant rate of mutual substitution between accuracy and energy. In the experiments, we present a parameter sweep evaluation for λ.

4.4.4 Partial Observability and Feedback

A central component of the discussed problem emerges because the system only observes the performance of the chosen model m_t on a given query q_t. The outcomes of all unselected models (i.e., what would have happened if we had chosen a different model) remains unknown. In the literature, this problem aspect is referred to as *partial feedback* or *bandit feedback* [LS20]. This setting fundamentally differs from *full-setting* scenarios where the performance of all candidates for every query $q \in Q$ is observed. Identifying the problem of dynamically routing queries not as a *full feedback* but a *partial feedback* one inherently motivates the use of MAB algorithms.

4.5 Bandit Problem Formulation

The partial observability described in Section 4.4 strongly suggests a solution following the *multi-armed bandit* framework. To summarize, in the presented routing problem, each query q_t corresponds to a distinct decision point in time, and each feasible model $m_t \in M_t^*$ represents an *arm* (or action). After selecting a model m_t to process a query q_t, the system observes the scalarized reward based on the models accuracy and energy consumption during inference while the performance of other models at time step t remains unknown.

4.5.1 Context-aware Bandits

One key extension of classical MAB algorithms is the inclusion of context features, as described in Section 4.2. Instead of treating all queries identically, *contextual bandits* leverage relationships between input characteristics and reward outcomes [LCLS10]. Utilizing the extracted feature vector of q_t, we aim to learn a policy:

$$\pi : x_t \rightarrow m_t$$

that selects a feasible model $m_t \in M_t^*$. Since different queries may demand varying model capacities, this approach is expected to outperform static or non-contextual strategies given enough time to learn the policy.

4.5.2 Exploration and Exploitation

The problem of partial observability leads us to the challenge of balancing *exploration* and *exploitation* [LS20]. Repeatedly choosing the same arm (exploitation) may yield short-term gains if the policy learned that this arm performs well. However, this might overlook other potentially superior arms (exploration) for certain features x_t. There exists a wide range of adaptions that allow MABs to contextualize their decisions, but fundamentally, all MABs must address this dilemma. This involves balancing the following considerations:

1. **Exploitation:** Choosing models that have historically reached high reward (strong accuracy-efficiency trade-off) for similar query contexts.

2. **Exploration:** Periodically trying less-explored models to improve estimates and learn their effectiveness for the given context.

Over time, the policy continuously gains information and refines its estimates to approximate the true reward functions for x_t. This allows the system to adapt to new query distributions and handle integration of new models gracefully.

4.6 Regret-Based Evaluation

To evaluate the performance of a policy, we measure the performance gap using the concept of *regret* [LS20]. It reflects an essential aspect of bandit problems that try to learn selecting near-optimal actions over time based on partial feedback. Regret quantifies the gap between the observed performance of the chosen action (selected model) and the performance that could have been achieved by selecting the optimal action in hindsight. The *oracle* policy with complete knowledge of model performances would have selected the optimal model:

$$m_t^* = \arg \max_{m \in M_t^*} r_t(m, q_t),$$

where $r_t(m_t, q_t)$ is the reward function introduced in Section 4.4.3) and M_t^* again defines the set of feasible models at time step t. Following this, *instantaneous regret* is defined as:

$$\Delta_t = r_t(m_t^*) - r_t(m_t),$$

where m_t is the model selected by the non-oracle routing policy. This quantifies the missed out on reward due to selecting a sub-optimal model. If the optimal model was chosen, meaning $\pi(x_t) = m_t = m_t^*$, regret is zero.

After T iterations, *cumulative regret* is defined as:

$$\text{Regret}(T) = \sum_{t=1}^{T} \left(r_t(m_t^*) - r_t(m_t) \right).$$

The objective of a routing strategy is to minimize $\text{Regret}(T)$ as t grows and approximate the oracle policy.

Since MAB is optimizing a multi-objective trade-off between accuracy and efficiency, minimizing regret should line up with *Pareto Optimality* (see Section 4.4.2). Over time, an effective policy aims to learn to select models that approximate the context-specific Pareto fronts and, thus, discard models that are consistently dominated.

Summary

This chapter formally defined the dynamic LLM routing problem. We introduced the core notation, further justified the need for query context representation, defined accuracy and energy consumption as key metrics, and formulated the problem as a multi-objective optimization problem (MOOP). Ultimately, we framed the setting as a bandit problem evaluated via regret. The next chapter details the proposed methodology and designed system architecture to solve this problem using contextual multi-armed bandit algorithms.

CHAPTER 5

Methodology and Solution Approach

In this chapter, we present the specific methodology and system architecture designed to implement the dynamic, context-aware LLM inference routing system. Section 5.1 presents the overall system architecture and its core modules. Section 5.2 describes the brief offline training process necessary for the task classifier. The subsequent Section 5.3 elaborates on the online phase and covers the feature extraction pipeline (Section 5.3.1), the routing module including feasibility checks and specific bandit algorithms (Section 5.3.2), and the inference engine (Section 5.3.3).

5.1 System Model

Our system architecture consists of four core modules that convert the theoretical construct of query set Q, model set M, and reward r_t (4.4.3) into a functional framework, as shown in Figure 5.1:

1. Feature Extraction Module
2. Routing Module
3. Inference Engine
4. Model Pool

The main system diagram (Figure 5.1) focuses on the core components that are involved in the *online adaptive routing phase*. The offline steps, revolve around providing configuration and relevant metadata on model properties like maximum context size and are detailed separately in Section 5.2.

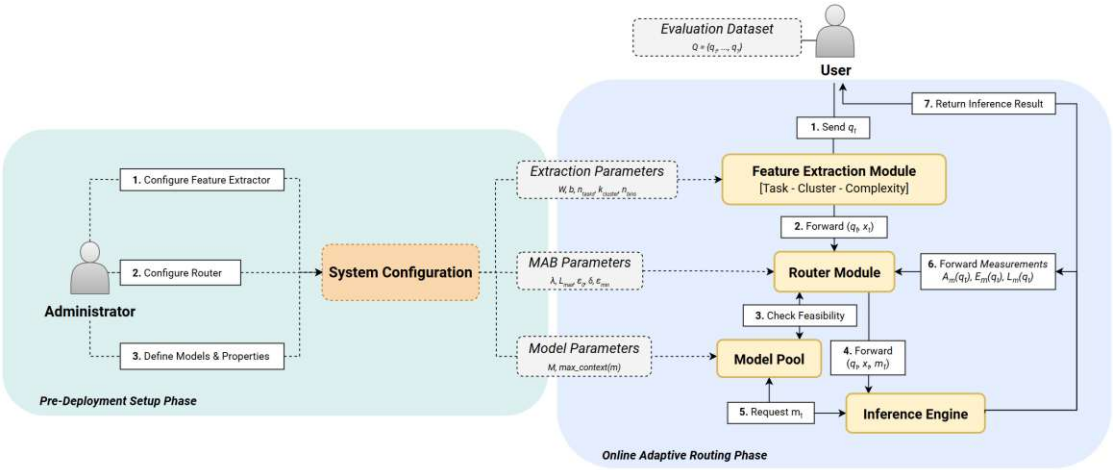

Figure 5.1: **High-Level View of Our System Model for Dynamic LLM Inference Routing.** An End User sends a query q_t, which is processed by the Feature Extraction Module to generate context vector x_t. The Routing Module uses these features and information from the Model Pool to determine M_t^* to select an appropriate model. The Inference Engine then performs inference with the chosen model, and produces a response and performance metrics (accuracy, energy, latency). These metrics are fed back to the Routing Module to adapt its routing policy online. The generated response is returned to the End User. Before the online phase, setup of the system configuration includes defining a set of models, configuring constraints and trade-off parameter λ, and providing the weights for the light-weight task classifier component.

The general processing flow is as follows: For each incoming query q_t, the feature extraction pipeline extracts the defined context vector x_t. Next, the Routing Module determines the feasible set of models M_t^* and selects one model m_t from this feasible set based on its current policy π_t and the trade-off parameter λ. Based on the chosen model m_t, the Inference Engine loads the model and returns the response to the user. Crucially, in this carefully simplified view, the Inference Engine also sends the measured performance metrics (accuracy, energy, latency) back to the Routing Module. Based on this feedback, the router updates its policy π_t online as soon as new information is available. Over time, the system learns an adaptive policy that intends to approximate Pareto-optimal solutions.

Figure 5.2: **Offline Phase Pipeline for Task Classification.** This diagram provides a high-level overview of the offline training phase. Labeled queries undergo instruction extraction and embedding, followed by training a Logistic Regression classifier. The trained model is then evaluated and stored for later deployment in the online phase. The blue coloring indicates that these components are part of the Feature Extraction module in the overall system.

5.2 Offline Phase

We rely on a lightweight text classification approach to identify high-level task types (e.g., summarization, QA) using a *Logistic Regression* (LR) model on top of semantic embeddings [BN06]. Specifically, we train the LR model as follows (illustrated in Figure 5.2):

1. We extract the instruction text $q_{\text{instr},t}$ from q_t. We assume that instructions are located in the first few lines of the prompt. This assumption aligns with common prompt structures in instruction-based tasks.

2. We compute the embedding of the extracted text using a pre-trained transformer-based embedding model [RG19]:

$$e_{\text{instr},t} = \text{embedding}(q_{\text{instr},t}),$$

   where embedding generates dense vector representations which capture semantic information.

3. The LR model is trained by optimizing the cross-entropy loss over a labeled dataset [BN06]. Given training pairs $(e_{\text{instr},t}, l_t)$, where l_t is the ground-truth task label, we minimize:

$$\mathcal{L} = -\sum_t \log p(l_t \mid e_{\text{instr},t})$$

   where

$$p(l \mid e_{\text{instr},t}) = \sigma(W e_{\text{instr},t} + b).$$

   This process updates W and b, the learned parameters that actually define the trained LR model by encapsulating the decision boundary for classification.

The resulting model is a simple classifier that extracts the discrete task label. Logistic Regression was chosen specifically for its computational simplicity, speed and low resource

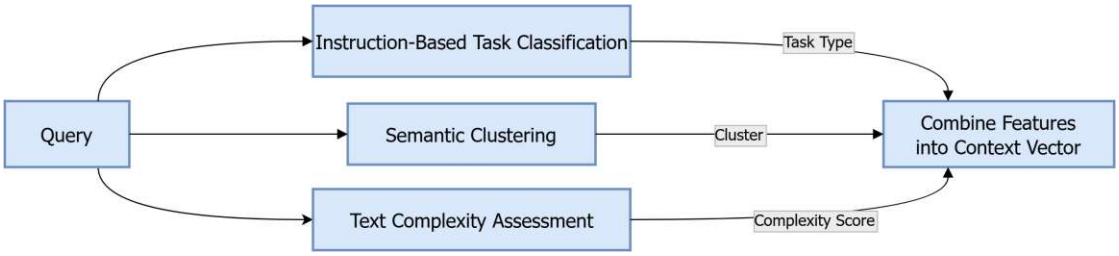

Figure 5.3: **Feature Extraction Overview.** The pipeline processes an incoming query through three parallel steps: *Instruction-Based Task Classification*, *Semantic Clustering*, and *Text Complexity Assessment*. The resulting features (*Task Type*, *Cluster*, *Complexity Score*) are then combined to form the context vector x_t used by the router.

footprint to avoid excessive offline overhead for a component that provides only one of multiple features for the routing decisions.

For training data, we sample a small portion from our evaluation dataset described in 7.1.1. Each query is annotated with a ground-truth task label based on its dataset of origin. We split the data into training and validation subsets, train the LR model, and evaluate its performance on the validation set using standard classification metrics such as the F1 score [Pow11]. After training, we store the LR parameters and proceed to the online phase.

More advanced classification methods (e.g., deep neural networks, support vector machines) could be integrated for potentially improved accuracy, but the emphasis here is on computational efficiency and robust task identification.

5.3 Online Phase

5.3.1 Feature Extraction

To make informed routing decisions based on the query-dependent LLM performance variations noted in Chapter 2, the feature extraction module constructs a context vector x_t from the input query q_t. We focus on three features grounded in related work: (1) task type, classified to capture broad query intent [HBL+24], (2) semantic context, derived via clustering embeddings to group similar queries (reflecting common embedding-based input representations such as EmbedLLM [ZWW+25]), and (3) text complexity, measured using Flesch Reading Ease score [Fle48] as an attribute [WCTG23] hinting at processing load (aligning with data-level optimization, Section 3.1.1). Figure 5.3 shows this pipeline. While more features exist, these were chosen based on related work and for simplicity.

Instruction-Based Task Classification

Given input query q_t, we extract the instruction text $q_{\text{instr},t}$ using the same procedure as during training. We then compute its semantic embedding using a pre-trained embedding

model [RG19]:

$$e_{\text{instr},t} = \text{embedding}(q_{\text{instr},t}), \quad e_{\text{instr},t} \in \mathbb{R}^{d_{emb}}.$$

The trained LR model classifies the embedding into one of the predefined task categories:

$$p(l \mid e_{\text{instr},t}) = \sigma(W e_{\text{instr},t} + b),$$

and the predicted task label is derived as:

$$\hat{l} = \arg\max_{l} p(l \mid e_{\text{instr},t}).$$

This predicted label $\hat{l}$ serves as the task type feature for subsequent routing decisions.

Semantic Clustering

For grouping the queries into domain clusters, we begin with embedding the entire query q_t using a pre-trained transformer-based embedding model [RG19]:

$$e_{full,t} = \text{embedding}(q_t).$$

Utilizing these full embeddings, we perform an online K-Means clustering algorithm [BB94] with a fixed number of clusters, K (specified in Section 7.1), to group queries based on semantic similarity. The algorithm assigns a query to the cluster with the most similar centroid by maximizing the cosine similarity between the query's embedding and the cluster centroids:

$$c_t = \arg\max_{c} \frac{e_{\text{full},t} \cdot \mu_c}{\|e_{\text{full},t}\| \|\mu_c\|}.$$

The centroids μ_c are updated online based on the number of points assigned to the cluster so far (N_c) as new queries $e_{\text{full},t}$ are observed:

$$\mu_{c_t} \leftarrow \mu_{c_t} + \frac{1}{N_{c_t} + 1}(e_{\text{full},t} - \mu_{c_t}),$$

where c_t is the cluster assigned to the current query $e_{\text{full},t}$ and N_{c_t} is the count of previous points assigned to that cluster. This implements a standard incremental update with a decaying learning rate. Online K-Means is chosen for its ability to adapt centroids incrementally without storing all past embeddings or requiring pre-determined clusters. The number of clusters K is fixed (value specified in Section 7.1) as a tunable hyperparameter to balance granularity and stability. Initial centroids are chosen from the first K distinct query embeddings. The online update allows for the cluster centroids to incrementally adapt to shifts in the topics of incoming queries over time. Cluster label c_t represents the semantic domain (or potentially topic) of a query q_t.

Text Complexity Assessment

We calculate a single numeric score that represents the complexity of query q_t based on the Flesch Reading Ease formula [Fle48]:

$$p(q_t) = 206.835 - 1.015 \cdot \left(\frac{\text{Words}_t}{\text{Sentences}_t}\right) - 84.6 \cdot \left(\frac{\text{Syllables}_t}{\text{Words}_t}\right).$$

For each q, this results in a value $\in [0, 100]$ where a low score indicates high text complexity. To convert this numerical score into a categorical feature suitable for the context vector, we bin the scores into N_{bins} distinct categories using equal-width binning. Again, the specific number of bins (N_{bins}) and the corresponding score ranges are detailed in the evaluation setup (Section 7.1).

Context Vector

Collectively, these features create a vector that encodes relevant characteristics of a query:

$$x_t = [l_t, c_t, p_t]$$

The resulting vector is forwarded to the router which aims to leverage these features to make more informed, query-dependent model choices. For the contextual bandit algorithms introduced in the next section, categorical features l_t, c_t, and p_t are converted into a numerical feature vector using one-hot encoding [BN06]. Additionally, we add an intercept term (bias) by appending a constant value of 1, thus, the resulting context vector $x_t \in \mathbb{R}^d$ has dimension $d = N_{\text{tasks}} + K + N_{\text{bins}} + 1$. The specific parameter values used in our experiments were determined through non-exhaustive tuning described in Section 6.2 and intend to balance feature granularity with dimensionality. The specific number of task types (N_{tasks}) is determined by the evaluation datasets defined in Section 7.1.1.

5.3.2 Routing

The Routing Module enforces feasibility based on latency and context window constraints and applies the MAB policy to pick a model from the feasible model set $m_t \in M_t^*$. The following subsections explain how both steps are implemented in practice and give insights in the technical details of three bandit algorithms we deploy and compare: ϵ-Greedy, LinUCB, and Thompson Sampling.

Model Feasibility Check

For each arriving query q_t, the RM determines the set of feasible models $M_t^* \subseteq M$ by applying two checks. First, a model m is judged infeasible if the query's input token count $\text{Tokens}_{in}(q_t)$ exceeds the model's maximum context window $\text{MaxContext}(m)$. The check considers only the input length to align with standard practice. Second, the model's predicted latency $\hat{L}_m(q_t)$ must not exceed the maximum allowed latency L_{max}.

Parameter L_{max} represents a configurable system or user constraint and can vary based on application requirements or user preference respectively. Its specific implementation during experimental evaluation is discussed in Chapter 7.

Since obtaining true latency $L_m(q_t)$ would require actually performing the inference, the latency feasibility check ($\hat{L}_m(q_t) \leq L_{max}$) requires an estimate of the inference time. We compute this estimate $\hat{L}_m(q_t)$ based on a single, overall throughput metric $S_{total}(m)$ (tokens/sec) maintained for each model as a simple cumulative average, updated after each inference based on the total inference time and the total number of tokens processed (input + output). We use the predefined maximum number of new tokens allowed for the query's task type (MaxNewTokens), as a conservative estimate for the output length component. In real deployments, more adaptive methods like Simple Moving Averages (SMA) or Exponential Moving Averages (EMA) for throughput might be beneficial.

The estimated latency is then calculated by dividing the total tokens (input + maximum allowed output) by the average overall throughput:

$$\hat{L}_m(q_t) = \frac{\text{Tokens}_{in}(q_t) + \text{MaxNewTokens}(\hat{l}_t)}{S_{total}(m)}.$$

With both the context window and latency constraints, the final set of feasible models is defined as:

$$M_t^* = \{m \in M : \text{Tokens}_{in}(q_t) \leq \text{MaxContext}(m) \text{ and } \hat{L}_m(q_t) \leq L_{max}\}$$

Multi-Armed Bandit

The core routing decision relies on multi-armed bandit (MAB) algorithms, which were chosen specifically for their ability to perform online learning and adaptation under partial feedback. From the feasible set M_t^*, each model $m \in M$ represents an *arm*. Based on the extracted context vector x_t, the MAB policy π must choose exactly one model m_t to process q_t. The MAB algorithms aim to maximize the expected scalarized reward:

$$r_t(m, q_t, \lambda) = (1 - \lambda)A_m(q_t) - \lambda C_m(q_t).$$

Here, the trade-off parameter λ balances accuracy and energy consumption, and can be interpreted as a system-wide goal or user preferences. $\lambda = 0$ prioritizes accuracy, while $\lambda = 1$ prioritizes energy efficiency. The performance of the MAB policy π is evaluated by its cumulative regret over T time steps and is defined as the total difference between the maximum reward possible by the optimal (often referred to as the *oracle*) and the actual reward obtained by π:

$$R(T) = \sum_{t=1}^{T} [r_t(m_t^*, q_t, \lambda) - r_t(m_t, q_t, \lambda)],$$

where $m_t = \pi(x_t)$ is the model chosen by the policy and $m_t^* = \arg\max_{m \in M_t^*} r_t(m, q_t, \lambda)$ is the best possible choice at time step t given the context and feasibility constraints.

The specific hyperparameters we applied for the MAB algorithms discussed below (ϵ-Greedy, LinUCB, Contextual Thompson Sampling) are defined via preliminary tuning as described in Section 6.2.

ϵ-Greedy

The ϵ-Greedy algorithm is one of the most fundamental exploration-exploitation algorithms [SBe98]. With probability ϵ, the system selects a random model out of all feasible options (exploration), and with complementary probability $1 - \epsilon$, the system chooses the model with the highest expected reward. For the basic (non-contextual) ϵ-Greedy algorithm, which we included as base bandit variation (see Algorithm 5.1), the highest expected reward is simply the observed *global average reward* of a model, independent of the current context x_t.

When a query arrives and the feasible set M_t^* is determined, for ϵ-Greedy we process as follows:

1. A value u is sampled from a uniform distribution $U(0, 1)$. If $u < \epsilon$, the router explores the model pool by selecting a random model from M_t^*. Otherwise, it follows *greedy exploitation* by choosing the model with the highest estimated average reward avg_reward[m].

2. Once a model is selected, it performs inference on the current query q_t. We observe energy consumption and latency during inference, evaluate the outcome to obtain accuracy, and store the resulting metrics.

3. The observed metrics are combined in the reward function r_t and we perform an incremental update of the average reward of m_t.

4. Over time, as more data is gathered, ϵ decays until it reaches a minimum exploration rate. This allows us to use the collected performance data to determine better choices while still occasionally discovering better-performing models or adapt to changes through exploration. Instead of the global average reward, contextual ϵ-Greedy variants with a reward model similar to those described for LinUCB and CTS can also be applied. However, the exploration step typically remains random.

ϵ-Greedy introduces minimal overhead and its internals are easy to interpret. However, its non-contextual variant limits its ability to adjust decisions to specific query characteristics.

LinUCB

LinUCB is one of the three contextual MAB variants we applied [LCLS10]. It approximates each model's expected reward as a *linear* function of the context vector x_t. It assumes the relationship between the context vector features x_t and the expected scalarized reward r_t is approximately linear. This linearity assumption is common in

Algorithm 5.1: Non-Contextual ϵ-Greedy Algorithm with Decay for Model Selection

Input: Initial exploration probability ϵ_0, decay factor δ, minimum exploration probability ϵ_{min}, time horizon T, model set M

Output: Model selection strategy balancing exploration and exploitation

for *each model* $m \in M$ **do**
  count$[m] \leftarrow 0$ avg_reward$[m] \leftarrow 0.0$
end
for $t \leftarrow 1$ **to** T **do**
  Determine feasible set M_t^* based on constraints Sample $u \sim U(0,1)$
  if $u < \epsilon$ **then**
    chosen_model $\leftarrow$ random model in M_t^*
  else
    chosen_model $\leftarrow \arg\max_{m \in M_t^*}$ avg_reward$[m]$
  end
  Observe reward r_t(chosen_model)
  count[chosen_model] $\leftarrow$ count[chosen_model] $+ 1$
  avg_reward[chosen_model] $\leftarrow$ avg_reward[chosen_model] $+ \frac{r_t(\text{chosen\_model}) - \text{avg\_reward[chosen\_model]}}{\text{count[chosen\_model]}}$
  $\epsilon \leftarrow \max(\epsilon_{\text{min}}, \epsilon \cdot \delta)$; `// Apply epsilon decay`
end

contextual bandit literature for its computational efficiency, ease of implementation, and interpretability. However, it only presents an approximation of likely more complex, non-linear relationships between context features and reward. Non-linear reward functions could again improve results but would also add computational overhead. For stability and to prevent singularity issues, LinUCB typically applies Ridge Regression principles [HK70]. For each internal reward model, LinUCB maintains a matrix $A_m \in \mathbb{R}^{d \times d}$ and a vector $b_m \in \mathbb{R}^d$. At first, A_m is set to $\lambda_{reg} I_d$, where I_d is the identity matrix and $\lambda_{reg} > 0$ is a set regularization parameter, and b_m is initialized as a zero vector. A_m accumulates $x_t x_t^T$ and b_m accumulates rewards, both terms scaled by the context vector.

Once a query arrives, we process:

1. For each model $m \in M_t^*$, we begin by computing the estimated feature weight vector θ with ridge regression:

$$\hat{\theta}_m = A_m^{-1} b_m.$$

   The expected reward for model m in the current context is computed as:

$$\hat{r}_m(x_t) = \hat{\theta}_m^T x_t.$$

2. To favor models with high uncertainty, we compute an exploration bonus:

$$\alpha\sqrt{x_t^T A_m^{-1} x_t},$$

where $\alpha \geq 0$ is a tunable parameter that balances exploration and exploitation. The bonus quantifies the uncertainty of model m in the current context x_t. A high value indicates that the model has been observed rarely in this context region.

3. The UCB score is calculated by combining the estimated reward and the exploration bonus:

$$\text{UCB}_m(x_t) = \hat{\theta}_m^T x_t + \alpha\sqrt{x_t^T A_m^{-1} x_t}.$$

4. The score aims to balance exploiting known well-performing models and exploring those with high uncertainty. We select the model with the highest UCB score:

$$m_t = \arg\max_{m \in M_t^*} \text{UCB}_m(x_t).$$

5. Once the chosen model m_t completes inference and we observe reward $r_t(m_t, q_t)$, LinUCB updates parameters:

$$A_{m_t} \leftarrow A_{m_t} + x_t x_t^T,$$

$$b_{m_t} \leftarrow b_{m_t} + r_t(m_t) x_t.$$

6. Updating A_{m_t} and b_{m_t} refines the estimate for $\hat{\theta}_{m_t}$ and thus, provides a more accurate mapping from context to expected reward as more and more data is collected.

Contextual Thompson Sampling

Contextual Thompson Sampling (CTS) is the second contextual MAB variant we applied [AG13]. Unlike LinUCB, which chooses the model with the highest upper confidence bound of expected reward, CTS maintains a posterior distribution of the model performance. This posterior represents the MABs quantified beliefs and uncertainty about each model's reward function given the observed (partial) feedback. After sampling from these posteriors, CTS selects the model with the highest sampled reward. This process inherently balances exploration without a deterministic optimism bonus and introduces randomness via *Bayesian inference*.

Similar to LinUCB, CTS assumes a linear reward function. It maintains matrices $A_m \in \mathbb{R}^{d \times d}$ and vectors $b_m \in \mathbb{R}^d$ that are initialized similarly as in LinUCB (though potentially including prior information, discussed in Chapter 6).

For each query, CTS performs the following steps:

Algorithm 5.2: LinUCB Algorithm with Regularization for Model Selection

Input: Exploration parameter α, Regularization parameter λ_{reg}, time horizon T, model set M

Output: Model selection strategy balancing exploitation and exploration

for *each model* $m \in M$ **do**
  $A[m] \leftarrow \lambda_{\text{reg}} I_d$; `// Initialize matrix with regularization`
  $b[m] \leftarrow 0_d$; `// Initialize reward accumulation vector`
end
for $t \leftarrow 1$ **to** T **do**
  Determine feasible set M_t^* based on constraints
  for *each model* $m \in M_t^*$ **do**
    $\hat{\theta}[m] \leftarrow A[m]^{-1} b[m]$ $s[m] \leftarrow \sqrt{x_t^T A[m]^{-1} x_t}$; `// uncertainty term`
    $\text{UCB}[m] \leftarrow \hat{\theta}[m]^T x_t + \alpha s[m]$
  end
  $\text{chosen\_model} \leftarrow \arg\max_{m \in M_t^*} \text{UCB}[m]$
  Observe reward $r_t(\text{chosen\_model})$
  $A[\text{chosen\_model}] \leftarrow A[\text{chosen\_model}] + x_t x_t^T$
  $b[\text{chosen\_model}] \leftarrow b[\text{chosen\_model}] + r_t(\text{chosen\_model}) x_t$
end

1. We assume a Gaussian posterior over θ_m, compute the mean $\hat{\theta}_m = A_m^{-1} b_m$ and the posterior covariance matrix $\Sigma_m = \sigma^2 A_m^{-1}$, where σ represents the standard deviation of the observation noise. Next, we sample a weight vector from this posterior:
$$\tilde{\theta}_m \sim \mathcal{N}(\hat{\theta}_m, \Sigma_m).$$

2. Estimate the expected reward using the sampled weights:
$$\tilde{r}_m(x_t) = \tilde{\theta}_m^T x_t.$$

3. The model with the highest sampled reward is chosen:
$$m_t = \arg\max_{m \in M_t^*} \tilde{r}_m(x_t).$$

4. After observing the actual reward $r_t(m_t, q_t)$, we update A_m and b_m as follows:
$$A_{m_t} \leftarrow A_{m_t} + x_t x_t^T,$$
$$b_{m_t} \leftarrow b_{m_t} + r_t(m_t) x_t.$$

Algorithm 5.3: Contextual Thompson Sampling Algorithm for Model Selection

Input: Noise standard deviation σ, time horizon T, model set M
Output: Model selection strategy balancing exploitation and exploration

for *each model* $m \in M$ **do**
  $A[m] \leftarrow I_d$ $b[m] \leftarrow 0_d$
end
for $t \leftarrow 1$ **to** T **do**
  Determine feasible set M_t^* based on constraints
  for *each model* $m \in M_t^*$ **do**
    $\hat{\theta}[m] \leftarrow A[m]^{-1}b[m]$ $\Sigma[m] \leftarrow \sigma^2 A[m]^{-1}$; `// use square of input sigma`
    Sample $\tilde{\theta}[m] \sim \mathcal{N}(\hat{\theta}[m], \Sigma[m])$ $\tilde{r}[m] \leftarrow \tilde{\theta}[m]^T x_t$; `// sampled reward`
  end
  chosen_model $\leftarrow \arg\max_{m \in M_t^*} \tilde{r}[m]$
  Observe reward r_t(chosen_model)
  A[chosen_model] $\leftarrow A$[chosen_model] $+ x_t x_t^T$
  b[chosen_model] $\leftarrow b$[chosen_model] $+ r_t$(chosen_model)x_t
end

5.3.3 Inference Engine

The Inference Engine processes the incoming query q_t and records relevant metadata for optimization. Specifically, it performs the following steps:

- If m_t is not already in memory, it is loaded from disk into GPU memory. The model then performs inference to generate a response for query q_t.
- To track model efficiency, the system measures energy consumption and latency from the start till the end of inference.
- Additionally, the system logs important metadata of the inference process, including number of input tokens, number of generated output tokens, and generation parameters such as *max_new_tokens* and *temperature*.

Summary

This chapter presented the concrete methodology we propose for realizing the dynamic routing framework. The modular system design separates feature extraction, routing, and inference. We discussed how task types, semantic clusters, and text complexity assessments are condensed into feature vectors to characterize queries, and how we use these features in established contextual bandit algorithms (LinUCB, CTS) inform our routing decisions under partial feedback. These methodological choices prioritize adaptability and efficiency for the routing process itself. Furthermore, the online learning

nature and the exploration inherent in the MAB approach allow the system to adapt to dynamic environments to integrate new models with minimal calibration overhead. The next chapter describes the practical implementation of this methodology.

CHAPTER 6

Implementation

This chapter describes aspects of how the system introduced in the previous chapter was implemented. We begin by detailing our testbed and environment including hardware characteristics. Next, we provide details on the implementation of the individual components. Finally, we list the key software dependencies required to run the system.

6.1 Testbed and Environment

Our experiments run on a dedicated server provided by the TU Wien HPC Research Group. The server is reserved during experiment runs to avoid any measurement interference and to ensure consistent results. It operates on Ubuntu 22.04.5 LTS, is powered by an AMD EPYC 9354P processor with 32 cores and has up to 503 GB of RAM available. The system uses an NVIDIA A100 GPU with 80GB VRAM which allows us to evaluate models of up to roughly 40B parameters at 16-bit precision. The GPU runs on CUDA version 12.2 and supports compute optimized bfloat16 acceleration [KMM+19].

We use Python 3.10 within a dedicated virtual environment to keep our dependencies consistent. Incoming requests are handled by a `FastAPI`[1] endpoint and queued for inference using an async connection to a `Redis`[2] container. A `PostgreSQL`[3] container stores prompts, model identifiers, inference results, and metrics. We process each request individually (batch_size = 1) to isolate and accurately measure latency, energy consumption, and accuracy. This design decision was made for simplicity given the many free-floating parameters already present in the system requiring analysis and to properly isolate requests for per-instance measurements. Model weights are assumed to be available on disk, not requiring any download during runtime. Datasets are loaded using

[1] https://fastapi.tiangolo.com/
[2] https://redis.io/
[3] https://www.postgresql.org/

the `datasets` library provided by HuggingFace[4], and random samples are extracted with Python's random package. Details on the specific dataset and the number of samples will be provided in Section 7.1.

Measuring GPU energy for very brief inference tasks required a special approach. An initial attempt through direct `pynvml` calls often failed and resulted in zero energy measurements due to NVIDIA Management Library (NVML) sampling limitations. We therefore utilized the `zeus` library[5], which is specifically designed for robust energy monitoring in machine learning workloads. `zeus` employs techniques, such as reading the GPU's cumulative energy counter, to provide reliable non-zero energy readings even for short durations where simpler methods fail. We focus measurement efforts on the GPU, as it constitutes the main energy draw during LLM inference tasks [HGSS24].

6.2 Prototype Implementation

For the implementation of feature extraction, we compute transformer-based embeddings with `sentence-transformers`[6], specifically using the popular `'all-MiniLM-L6-v2'` model. We classify the task using a simple Logistic Regression model [BN06] and group similar queries using an online K-Means clustering algorithm [BB94], supported by utilities from `scikit-learn`[7]. Initial K-means centroids were derived from the first K queries encountered during the experiment runs. We assess text complexity using the `textstat` library[8], utilizing the Flesch Reading Ease score [Fle48]. The resulting Flesch Reading Ease scores were discretized using equal-width binning based on the range observed during preliminary profiling.

The multi-armed bandit strategies (ϵ-Greedy [SBe98], LinUCB [LCLS10], Contextual Thompson Sampling [AG13]) are implemented as custom classes built upon `NumPy`[9] arrays. Before calculating the reward, we normalize energy consumption (calculated as energy consumption per token (input + output)) between [0,1] using min-max scaling. We chose min-max normalization because the energy metric is combined directly with accuracy (also between 0 and 1) and we assume a linear importance of the trade-off metrics. While other methods like z-score or log-scaling were considered, min-max normalization was selected as it maps energy consumption directly to the same bounded [0,1] scale as the accuracy metric. Consistent scaling ensures the trade-off parameter λ provides clear control over the relative weighting of the objectives. Initial energy bounds for normalization were determined through minimal profiling of representative models from the pool before the main experiment runs (see Section 7). While the system could potentially update bounds dynamically, the bounds established during profiling proved stable and were used consistently throughout the reported experiments.

[4] `https://huggingface.co/docs/datasets/index`
[5] `https://github.com/ml-energy/zeus`
[6] `https://www.sbert.net/`
[7] `https://scikit-learn.org/stable/`
[8] `https://pypi.org/project/textstat/`
[9] `https://numpy.org/`

For model inference, we load models from our model pool. Each model is a HuggingFace-compatible LLM loaded in `bfloat16` [KMM+19] precision to optimize GPU memory usage and to align with standard industry practices for inference. We use the `transformers` library[10] with `PyTorch`[11] for tokenization and response generation, and keep default generation parameters to reflect the most typical user setup. As mentioned in Section 6.1, we measure the GPU's energy consumption using the `zeus` library[3]. We process a single warm-up inference after loading a new model to account for lazy initialization that would otherwise skew measurements. Latency is captured with Python's `time` module by sampling measurements immediately before and after the response generation step, specifically excluding time spent on feature extraction, routing decisions, or model loading (models are assumed pre-loaded for this measurement). Consequently, measured latency also excludes queue time, as requests are processed sequentially in this setup. Finally, we evaluate the generated responses using exact match scores for precise answers and ROUGE metrics [Lin04] for summarization tasks using the `evaluate` library[12].

Prior to the main experiments, we conduct experiment runs for preliminary tuning to determine reasonable values for key hyperparameters. This involved adjusting parameters for the MAB algorithms such as initial exploration rate ϵ_0, and the feature extraction process (number of semantic clusters $K_{cluster}$ and N_{bins}) to evaluate the impact on cumulative regret. The goal was to identify settings that provided a good balance across parameters while leaning towards simpler configurations to mitigate sparsity in the resulting MAB reward models. The final MAB hyperparameters selected through this process are detailed in Table 6.1 and consistently used throughout the experiment unless specifically mentioned otherwise.

We also incorporate basic error handling and logging mechanisms. Failing requests are logged, retried, and requeued using our `Redis`-based queuing system. Although full error handling is still under development, the system is designed to automatically reprocess failed inferences to maintain robustness.

Summary

This chapter outlined how the methodology was translated into a concrete implementation. The experimental environment will revolve around a NVIDIA A100 GPU whose energy consumption is measured using the `zeus` library[3]. The chapter also documented essential software dependencies and the tuned hyperparameters for the MAB algorithms necessary for reproducibility of our results. The empirical evaluation of the implemented system is presented in the next chapter.

[10] `https://huggingface.co/docs/transformers/index`
[11] `https://pytorch.org/`
[12] `https://huggingface.co/docs/evaluate`

Table 6.1: **MAB Algorithm Hyperparameters.** Overview of key hyperparameters for the MAB algorithms used in the system evaluation. Values shown represent the best configurations identified during preliminary tuning and were used consistently across experiments unless otherwise specified.

Algorithm	Parameter	Value	Description
ϵ-Greedy	Initial ϵ_0	1.0	Starting exploration rate
	Decay factor δ	0.98	Multiplicative decay rate for ϵ per step
	Minimum ϵ_{min}	0.01	Lower bound for exploration probability
LinUCB	Exploration α	0.1	Exploration parameter for LinUCB
	Regularization λ_{reg}	0.05	Regularization parameter
Thompson Sampling	Noise Std Dev σ	0.01	Standard deviation assumption for reward noise

Table 6.2: **Software Dependencies.** Relevant Python packages and their versions that were used in the implementation of the routing framework and evaluation environment.

Package	Version	Purpose
datasets	3.1.0	Accessing evaluation datasets
evaluate	0.4.3	Standardized evaluation of model generations
fastapi	0.115.4	Handling incoming requests
nltk	3.9.1	Text normalization
zeus-ml	0.11.0.post1	Measuring GPU energy consumption
protobuf	6.30.2	Serialization format used by `transformers` library
redis	5.2.0	Async request buffering and queueing
rouge_score	0.1.2	ROUGE metric calculation
scikit-learn	1.5.2	Classification, clustering
scipy	1.14.1	Mathematical utilities
seaborn	0.13.2	Data visualization
sentence-transformers	3.2.1	Generating BERT-based embeddings
sentencepiece	0.2.0	Tokenization
textstat	0.7.4	Computing text complexity metrics
torch	2.5.1	Model inference with PyTorch
transformers	4.46.2	Model loading, tokenization, and response generation
uvicorn	0.32.0	ASGI server for FastAPI

CHAPTER 7

Evaluation

This chapter presents the experimental setup and plan, followed by results and analysis of the implemented dynamic routing framework. Section 7.1 details the environment and experiment components, including the selected datasets, model pool, and response evaluation metrics utilized for the experiments. Section 7.2 outlines a series of experiments, presented in 7.3, designed to assess the system outcomes against baselines, the impact of contextual features, the effect of different settings for the trade-off parameter λ, and how the system adapts its model selection policy to new models. We conclude our experiments with an overhead analysis on the example of latency. Finally, Section 7.4 discusses the key findings, their implications, and the limitations of the study.

7.1 Experimental Setup

7.1.1 Evaluation Datasets and Data Selection

To provide a robust and realistic evaluation of our proposed dynamic routing system, we require a selection of representative datasets that include diverse query types, complexity levels, and domains. This diversity is crucial not only for generalizability but also to capture the types of query-dependent performance variations discussed in Section 2.3.4, which present a core motivation for this work. The diversity of the query set is expected to strongly impact the system's ability to utilize the heterogeneous capabilities of different models and, consequently, its potential to outperform static, single-model approaches.

We selected evaluation datasets based on their popularity in LLM benchmarking, representation of different task types and whether model responses could be automatically and accurately evaluated by the system against a ground truth. While human feedback or LLM-as-a-judge [ZCS+23] would allow us to investigate a broader set of benchmark datasets, they introduce significant costs and potential inconsistencies as the evaluators themselves would need to be a representative group of the user population. Additionally,

determining accuracy on open-ended tasks is challenging even with human or LLM judges as interpretation is subjective and often lack clear criteria for evaluation. Thus, we focused on tasks with unambiguous ground truth answers available or established automatic metrics.

The Massive Multitask Language Understanding (**MMLU**) dataset [HBB+21] consists of multiple-choice questions across 57 subjects such as mathematics and humanities. It serves as an assessment of reasoning capabilities in domain knowledge. MMLU asks questions directly without additional context but provides answer options with only one being correct. It mimics standardized tests one would observe in school to assess knowledge of the examinee across domains. The questions also vary in difficulty and complexity from straightforward ones to others requiring analytical thinking.

HellaSwag [ZHB+19] tests commonsense inference abilities through sentence completion. It requires the model to select the most plausible continuation for a given uncompleted part of a text and measures situational reasoning capabilities. **Winogrande** [SBBC21] focuses similarly on commonsense reasoning and extends the famous Winogrande Schema Challenge (WSC) [LDM12] that was proposed in 2011 as an alternative to the over half a century old Turing test. It presents sentences containing ambiguous pronouns. To give an example, in the sentence "The trophy didn't fit into the suitcase because it was too big." the pronoun "it" could technically refer to either "the trophy" or "the suitcase". The model has to derive which noun the pronoun actually refers to based on its real-world knowledge and contextual reasoning abilities.

GSM8K [CKB+21] contains grade school math problems requiring multi-step reasoning which tests a model's structured analytical reasoning capabilities. Determining the correct answer requires arithmetic, algebra, and logical reasoning. Including the dataset in our system evaluation data allows us to assess how models handle mathematical tasks of heterogeneous complexity. To cover long-context generation, we include the **CNN/Daily Mail** benchmark [HKG+15] which requires synthesizing key information from provided news articles.

Table 7.1 provides a comprehensive overview of the discussed datasets showing the number of samples used from each test set, the evaluation metric we apply, and the task type label we categorize it in for the task type classification of the context vector extraction. To maintain a feasible experiment scope, we randomly sample 500 instances of each dataset to include in our system evaluation dataset using a fixed random seed for reproducibility of results.

We uniformly sample 500 items per dataset at a fixed seed to give the MAB algorithms reasonable data to learn on while keeping experiment preparation time low.

7.1.2 Model Selection

For our experiments' pool of models, we target publicly available LLMs that represent the LLM landscape. We limit our selection to open-access models for several reasons:

Table 7.1: **Overview of Evaluation Datasets.** 500 samples per dataset strike a balance between enough instances for the MABs to learn on and compute budget.

Dataset	Samples Used	Task Type	Evaluation Metric
MMLU	500	Question Answering	Exact Match
HellaSwag	500	Situation Completion	Exact Match
Winogrande	500	Commonsense Reasoning	Exact Match
GSM8K	500	Mathematical Reasoning	Exact Match
CNN / Daily Mail	500	Summarization	ROUGE

(1) they allow direct measurement of energy consumption on our local GPU, which is our chosen method to determine model efficiency; (2) they do not cause variable costs associated with commercial API credits, which is beneficial for excessive prototyping, development, and experiment runs; and (3) they represent a wider range of optimized and specialized capabilities than typically found in commercial offerings, which better reflects the heterogeneous use cases that real-world deployments might induce.

To demonstrate that our system is capable of incorporating a large number of diverse models, we choose representatives from different parameter scales and model families. Our selection criteria can be summarized as:

1. Diversity across parameter counts (from 0.5 B to 34 B), determined by the compatibility with our available compute resources
2. Coverage of major model families by relevant vendors (Phi[1], Gemma[2], Mistral[3], Llama[4], Qwen[5])
3. Accessibility of model weights for local deployment
4. Recent publication date to reflect state-of-the-art

Based on these criteria, we were able to systematically define a representative pool of models to be included in our system evaluation (see Table 7.2). The differences in model characteristics are expected to allow us to observe notable trade-offs between accuracy and efficiency in a dynamic routing setting. As discussed in Chapter 6, energy consumption of inference using these models was measured under controlled conditions using the `zeus` library.

[1]Microsoft Phi models, `https://azure.microsoft.com/en-us/products/ai-studio/phi`
[2]Google Gemma models, `https://ai.google.dev/gemma`
[3]Mistral AI models, `https://mistral.ai/`
[4]Meta Llama models, `https://www.llama.com/models/llama-3/`
[5]Alibaba Cloud Qwen models, `https://qwenlm.github.io/`

Table 7.2: **Overview of the Model Pool.** The table lists the LLMs used in the experiments, grouped by model family, along with their parameter counts and Hugging Face identifiers (HF Handles).

Family	Version	# Parameters (B)	HF Handle
Qwen	2.5	0.5	Qwen/Qwen2.5-0.5B-Instruct
	2.5	1.5	Qwen/Qwen2.5-1.5B-Instruct
	2.5	3	Qwen/Qwen2.5-3B-Instruct
	2.5	7	Qwen/Qwen2.5-7B
	2.5	14	Qwen/Qwen2.5-14B-Instruct
Mistral	v0.3	7	mistralai/Mistral-7B-Instruct-v0.3
Gemma	3	1	google/gemma-3-1b-it
	3	4	google/gemma-3-4b-it
	3	12	google/gemma-3-12b-it
	3	27	google/gemma-3-27b-it
Llama	3.1	1	meta-llama/Llama-3.1-1B-Instruct
	3.2	3	meta-llama/Llama-3.2-3B-Instruct
	3.1	8	meta-llama/Llama-3.1-8B-Instruct
Phi	4-mini	4	microsoft/Phi-4-mini-instruct
	4	14	microsoft/Phi-4-14B
Yi	-	34	01-ai/Yi-34B

7.1.3 Response Evaluation

Evaluating model responses requires consistent metrics for our diverse set of tasks. Our evaluation framework targets extracting the final answer from the free-form model output. We normalize both accuracy and energy consumption to allow for a consistent comparison and to combine them in a scalarized reward function in the range [0-1].

Answer Extraction

As LLMs typically respond with freeform text, we implement task-specific constraints as extraction techniques to obtain meaningful answers that can be evaluated against ground truths (summarized in Table 7.3).

Metric Normalization

We implement a unified normalization approach for the different evaluation metrics to harmonize different metric distributions and ranges. Each metric might produce scores in varying ranges with different distributions which makes direct comparison without normalization challenging. For example, a ROUGE-L score of 0.4 on a summarization task might represent solid performance (close to human-level or best performing LLM), while an accuracy of 0.4 on an easy multiple-choice task could indicate a poor score.

Table 7.3: **Answer Extraction Methods by Dataset.** This table outlines the specific technique used to extract the final answer from the models free-form output for each evaluation dataset, along with custom generation stopping criteria to ensure fair comparison.

Dataset	Extraction Method	Max Tokens	Stop Criteria
MMLU	Extract the chosen option (A-D)	1	Linebreak (`\n`)
HellaSwag	Extract the chosen option (A-D)	1	Linebreak (`\n`)
Winogrande	Extract the chosen option (A-B)	1	Linebreak (`\n`)
GSM8K	Extract the final numeric answer (via regex)	256	Linebreak (`\n`)
CNN / Daily Mail	Use the entire generated text as the final summary	256	Hashtags (`###`)

Without proper normalization, comparing different metrics would be misleading as it suggests equal performance across tasks. Thus, we apply min-max normalization to all evaluation metrics, which converts observed accuracy values to a range of [0,1] to allow a more consistent comparison across tasks:

$$\text{Normalized Accuracy} = \frac{A - A_{\min}}{A_{\max} - A_{\min}},$$

where $A_{\min}$ and $A_{\max}$ are determined from baseline profiling runs of representative models in our pool on the validation set for each specific task type. For establishing these bounds, we strategically selected models that are likely to represent accuracy extremes, i.e., using smaller, older models (e.g., Phi2-3B) to estimate minimal accuracy values and larger, newer models (e.g., Qwen2.5-32B) to estimate maximum accuracy thresholds. This approach avoids exhaustive evaluation seen in other frameworks (see Chapter 3), but still creates applicable normalization boundaries. Bounds are fitted on validation splits to avoid leaking accuracy data into the reward function. The derived values allow us to put accuracy into perspective across tasks and do not represent exact theoretical limits.

For energy consumption, we normalize measurements using a similar min-max scaling approach:

$$\text{Normalized Energy} = \frac{E_{\text{measured}} - E_{\min}}{E_{\max} - E_{\min}}$$

Boundaries are derived from the same profiling runs described above. While energy consumption in itself would be comparable without normalization, normalization allows us to combine it with accuracy scores on the same scale into a scalarized reward function. Before normalization, energy measurements are converted to energy consumption per token (input + output) for comparability of measurements across different query lengths.

We consider both input and output token count as some tasks involve very short outputs (e.g., single-character answers in multiple-choice tasks).

As discussed in Chapter 5, all applied MAB strategies rely on the same scalarizing reward function to balance the normalized accuracy and energy consumption for each task:

$$\text{Reward} = (1 - \lambda) \times \text{Normalized Accuracy} - \lambda \times \text{Normalized Energy},$$

where $\lambda \in [0, 1]$ is the configurable trade-off parameter that controls the relative weight of accuracy vs. energy efficiency. Notably, the latency feasibility constraint L_{max} was assumed to be set to a non-restrictive value to focus evaluation solely on the accuracy-energy trade-off.

7.2 Experimental Plan

To investigate the effectiveness of our dynamic routing framework for LLM inference, we propose the following experiments. They intend to answer five core aspects: First, we need to understand how dynamic routing compares to conventional approaches, which we define as baselines below. This will provide the most fundamental insights and justify further analysis. Second, we explore the impact of different contextual features on routing decisions, as feature extraction introduces overhead and not all features are expected to contribute equally to potential accuracy and efficiency gains. Third, we show how the system handles the trade-offs at different configurations by varying the λ parameter. This parameter sweep will demonstrate the system's ability to operate at different points along the Pareto front. Fourth, we test if the system adapts effectively to changes in the model pool to simulate the real-world scenario of new models being released regularly. Fifth, we evaluate the overhead the routing mechanism itself introduces to determine whether the benefits outweigh the costs.

We compare our MAB-based routing approach against multiple baselines. **Random** selection serves as an estimate of how the whole model population achieves accuracy and efficiency on average by randomly picking a model from the model pool for each query.

We implement four static model selection baselines that reflect different optimization priorities by utilizing their assigned model for all queries:

1. **Largest (Yi-34B)**. In many deployments, one finds the tendency to assume that larger models yield better results. This approach disregards resource demands and simulates real-world scenarios that place model accuracy above all else.

2. **Smallest (Qwen2.5-0.5B)** Some configurations follow strict resource constraints. A baseline with the smallest model shows a scenario where energy use and hardware requirements determine model choice. It indicates the accuracy loss associated when minimizing resource consumption.

3. **Highest-Accuracy (Gemma-3-27B)** This baseline selects the model that achieves the best possible average accuracy on the benchmark tasks with no concern for efficiency. We derive it through exhaustively profiling all models of the model pool. It acts as an upper limit of achievable accuracy in a single-model setting.

To assess the value of different contextual features, we run experiments with different feature configurations. Context-free routing uses no features and the router learns only their global average reward. Single-feature routing only has information about one of the three context dimensions (task type, semantic cluster, text complexity) during the whole experiment run. Full-context routing leverages all features by using all derived query characteristics.

We implement and evaluate three MAB algorithms. To briefly revisit: ϵ-Greedy explores with probability ϵ and exploits with probability $1 - \epsilon$, LinUCB selects arms based on an upper confidence bound derived from a learned linear reward model, and Contextual Thompson Sampling represents uncertainty in model performance through posterior sampling also based on a linear reward model. Additionally, we evaluate a contextual variant of ϵ-Greedy which uses a similar linear reward model similar to that used in LinUCB for its exploitation step. We investigate learning behavior based on cumulative and moving-average regret, and evaluate the resulting regret, accuracy, and energy consumption for each algorithm, with focus on cumulative and moving-average regret as a key metric for comparing effectiveness.

For trade-off analysis, we vary the λ parameter from 0 (accuracy only) to 1 (efficiency only) in increments of 0.1. For each value, we run the full system with the best performing MAB algorithm and all contextual features. The result will be a Pareto front that demonstrates achievable operating points and helps identify meaningful configurations for different preference profiles.

To assess adaptability, we introduce a new model which has shown high reward scores in previous experiment after 1000 queries and investigate if and how extensively the system incorporates it. This experiment gives insights on the system's ability to adapt on-the-fly without requiring recalibration or retraining like most approaches discussed in Chapter 3.

For all experiments, we use our evaluation dataset consisting of 500 samples from each of the 5 benchmarks which equates to a total sequence length of $T = 2,500$ queries per experiment run. For presenting MAB internals and regret, we report normalized accuracy, total energy consumption, model selection frequency, and regret values (see Chapter 5). To estimate system overhead, we report mean latency by component and model selection time per MAB algorithm. Results include 95% confidence intervals, where appropriate, to account for the inherent variance in LLM outputs and the MAB learning processes. Unless otherwise specified, each experiment runs for the full sequence length of $T = 2,500$, which allows the MAB algorithms to learn and stabilize.

Unless specified differently, the feature extraction parameters were set to $K = 3$ semantic clusters and $N_{\text{bins}} = 3$ text complexity bins. Following one-hot encoding and the addition

of an intercept term as described in Chapter 5, this results in a context vector dimension of $d = 12$ for the contextual bandit algorithms.

7.3 Results & Analysis

7.3.1 Baselines Comparison

Figure 7.1 presents the fundamental optimization space of our routing problem. We compare the average accuracy-energy outcomes during a single simulation run and report the mean normalized accuracy against the total cumulative energy consumption. The Y-axis shows the mean accuracy across all tasks (higher is better), normalized with min-max scaling. The X-axis shows the total energy consumed after processing the full set of queries (lower is better), reported in Wh.

Blue dots represent the individual models, and the red dashed line indicates the Pareto front (see Chapter 4). The positions of the static baselines reveal relevant insights. The *smallest* included model (Qwen2.5-0.5B) achieves the lowest accuracy and is located near the bottom-left of the frontier. The *largest* model (Yi-34B) consumes considerable energy without notably higher accuracy compared to the smallest models in the pool. The *highest accuracy* model (Gemma-3-27B) is situated on the right end of the frontier but consumes considerably more energy compared to its smaller 12 B variant at almost the same accuracy. Expectedly, the random baseline shows a worse accuracy-energy trade-off than the Pareto-optimal static choices on the frontier and is centered in the middle.

This static analysis displays the inherent limitations of single-model deployment. The existence of a clear Pareto front, combined with the availability of non-optimal choices located below or to the right of the front, strongly motivates the investigation of dynamic routing strategies to leverage the strengths of different models.

Table 7.5, presented below in 7.3.7, give detailed insights on the latency statistics and variation in efficiency for each model within the pool.

7.3.2 Initial Bandit Performance: ϵ-Greedy vs. Random

In Figure 7.2, we compare the cumulative regret of a simple, non-contextual ϵ-Greedy algorithm with the random baseline. The plot shows the mean cumulative regret over five runs with a 95% confidence interval to ensure robustness despite the variance in MAB learning. Both strategies begin with zero regret. For the random baseline, we can observe regret increase steeply and linearly. The ϵ-Greedy algorithm initially shows identical regret, in line with the expected high exploration behavior of the ϵ-Greedy algorithm. After around 200 queries, however, we see the ϵ-Greedy algorithm beginning to diverge from the baseline, over time widening the gap. This gap suggests that the algorithm is indeed learning and is able to find more optimal model choices than the random selection. At the end of the experiment, the ϵ-Greedy algorithm achieves an improvement of 20-25% in cumulative regret on average.

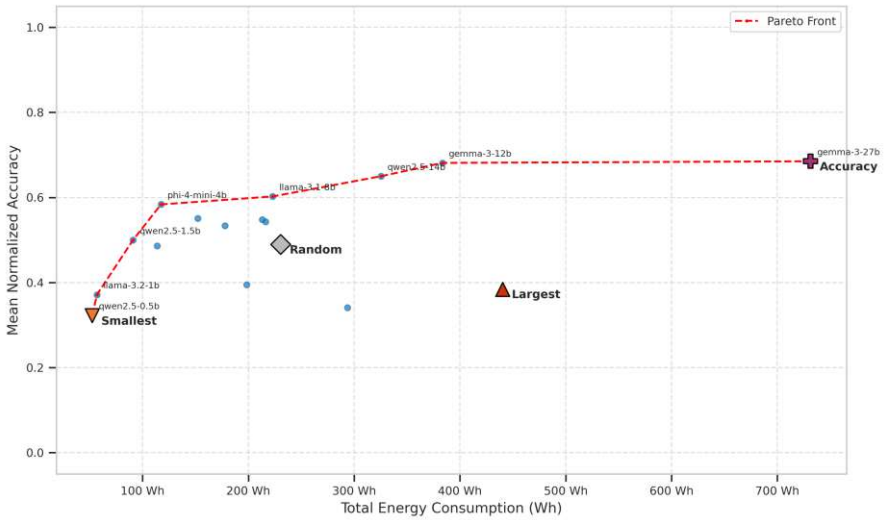

Figure 7.1: **Accuracy vs. Energy Trade-off for Static Baselines.** The plot shows the mean normalized accuracy against the cumulative energy consumption for each model in the pool (blue dots). Static baselines (largest, smallest, highest accuracy, random) are highlighted with distinct markers. The red dashed line indicates the Pareto frontier and represents the optimal trade-offs achievable with individual models.

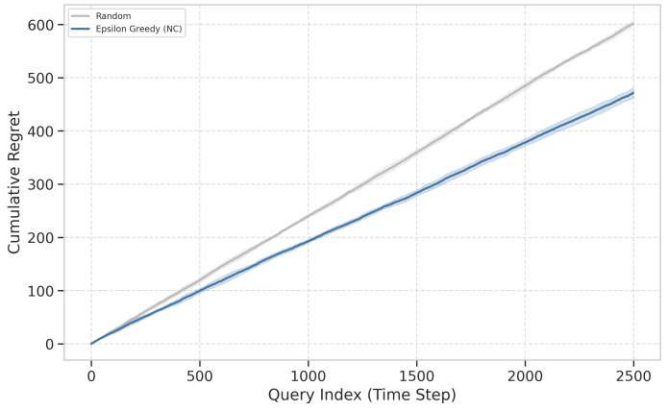

Figure 7.2: **Cumulative Regret: ϵ-Greedy vs. Random Routing.** The plot compares the cumulative regret of the ϵ-Greedy algorithm without context against a random baseline without context features at $\lambda = 0.4$ across all runs (n=5). Shaded areas represent the 95% confidence interval.

Figure 7.3 provides further insights into the same experiment. We show the selection frequency of each model within each run. While we do see great variability across selections, we are still able to derive clear patterns. The algorithm concentrates selections on a subset of generally well-performing models at or close to the Pareto front (Gemma-3-12B, Llama-3.1-1B, Phi-4-mini-4B, Qwen-2.5-3B). Furthermore, we clearly see suboptimal models (Gemma-3-27B, Yi-34B) being chosen at a very low frequency below 1%, which indicates that the algorithm is able to effectively learn about the unknown model characteristics and, at the given trade-off parameter λ, converge to one of the Pareto-optimal models.

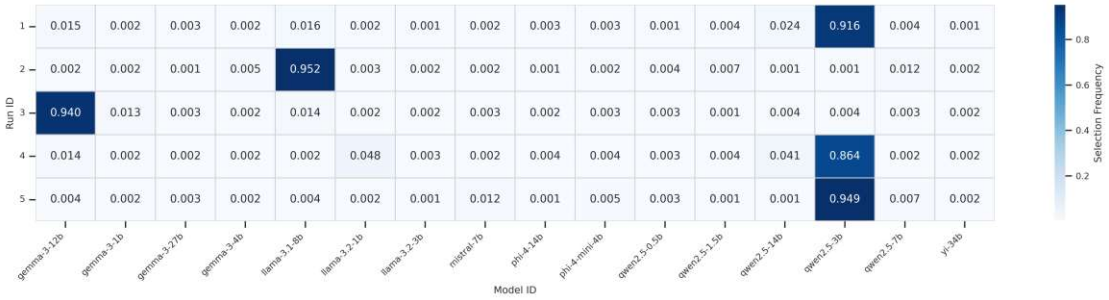

Figure 7.3: **Non-Contextual ϵ-Greedy Model Selection Patterns.** The heatmap shows the frequency each model was selected by the non-contextual ϵ-Greedy algorithm at $\lambda = 0.4$ in individual runs (n=5).

7.3.3 MAB Algorithm Comparison

We now compare the effectiveness of the different MAB algorithms proposed in Chapter 5. Figure 7.4 provides a high-level comparison of the mean normalized accuracy against the total energy consumption for each algorithm ($\lambda = 0.4$). The plot visualizes the accuracy-efficiency trade-offs of the dynamic strategies relative to the static baselines and the Pareto front. The plot supports the initial bandit insight from Section 7.3.2. Predominantly, the three contextual MAB algorithms – Contextual ϵ-Greedy, LinUCB, and Contextual Thompson Sampling – position themselves closely together in the more optimal top-left region when using the full feature set available. All algorithms achieve high accuracy (0.63-0.64) at comparably low energy consumption. More specifically, they out-perform the non-contextual ϵ-Greedy by reaching higher accuracy (0.64-0.65 vs. 0.59) with reducing energy consumption. Compared to the static baselines, improvements become even more prevalent with the contextual strategies reducing energy consumption substantially compared to the random, largest, and accuracy baseline, while simultaneously achieving superior accuracy to the random (≈ 0.51), largest (≈ 0.39), and smallest (≈ 0.33) baseline. The dynamic contextual strategies surpass the static Pareto front, which demonstrates that using context for dynamic routing can allow superior accuracy-efficiency balance by effectively combining the usage of multiple models.

Complementing this high-level perspective, Figure 7.5 shows the specific mean normalized

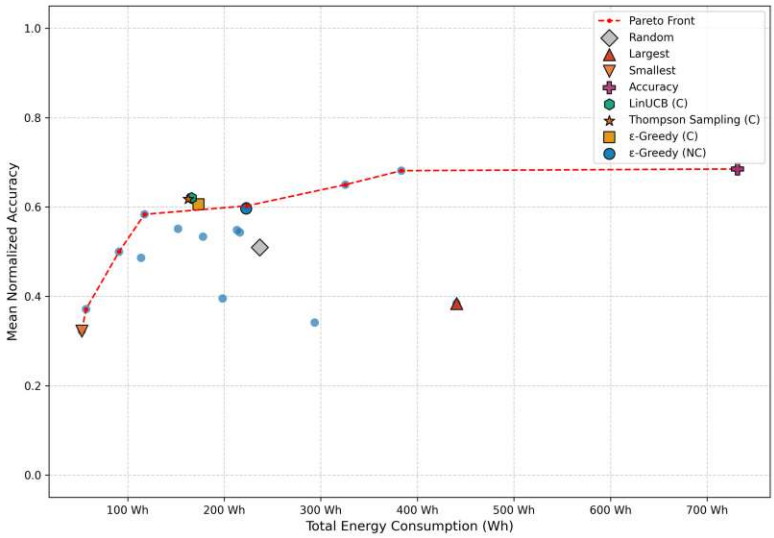

Figure 7.4: **Accuracy vs. Energy Trade-off: MAB Algorithms vs. Baselines.** The plot shows the mean normalized accuracy against the total energy consumption achieved by each MAB algorithm and static baselines. The static Pareto frontier is shown for reference.

accuracy and energy consumption of multiple experiment runs (n=50) and includes 95% confidence intervals to provide a more robust comparison than the single-run snapshot of Figure 7.4. The data supports the observation that the contextual MAB algorithms consistently achieve higher accuracy at lower energy consumption than our defined baselines. Critically, the confidence intervals for both accuracy and energy consumption largely overlap across the three contextual MAB algorithms, which suggests that there exists no significant difference in achieved accuracy or energy consumption between the proposed contextual MAB algorithms in our system.

Moving on to learning dynamics, Figure 7.6 illustrates the cumulative regret of each MAB algorithm over the query sequence. The plot confirms the expected linear regret increase of the static baselines and the random selection. The non-contextual ϵ-Greedy implementation demonstrates learning capacity. However, its regret grows visibly faster than the contextual variants. Contextual strategies show more effective learning, indicated by their flatter regret curves visually beginning to diverge after the initial 200-300 queries. This, on the other hand, shows the impact of early policy optimization on overall regret. Overall, Contextual ϵ-Greedy yields the lowest mean regret ($\approx$ 398), around 15% reduction compared to its non-contextual version ($\approx$ 466). LinUCB ($\approx$ 412) and Contextual Thompson Sampling ($\approx$ 400) achieve similar regret reductions.

Finally, Figure 7.7 visualizes the moving average regret for a window size of 50 to investigate learning stability and convergence patterns. An initial *cold start* period is evident for the MAB algorithms. We can observe high and erratic average regret

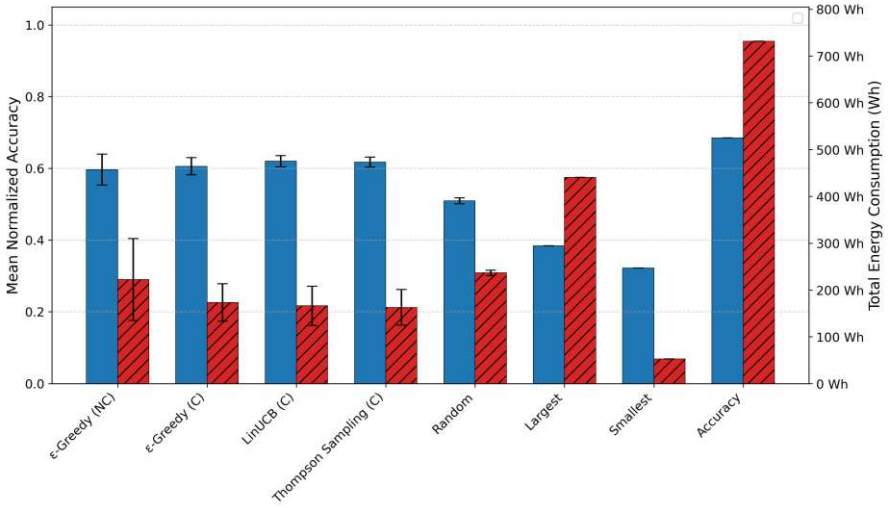

Figure 7.5: **Mean Accuracy and Total Energy Consumption: MAB Algorithms vs. Baselines.** Comparison of mean normalized accuracy (higher is better) and total energy consumption across different routing strategies. Contextual MAB algorithms use all available features. Error bars represent 95% confidence intervals.

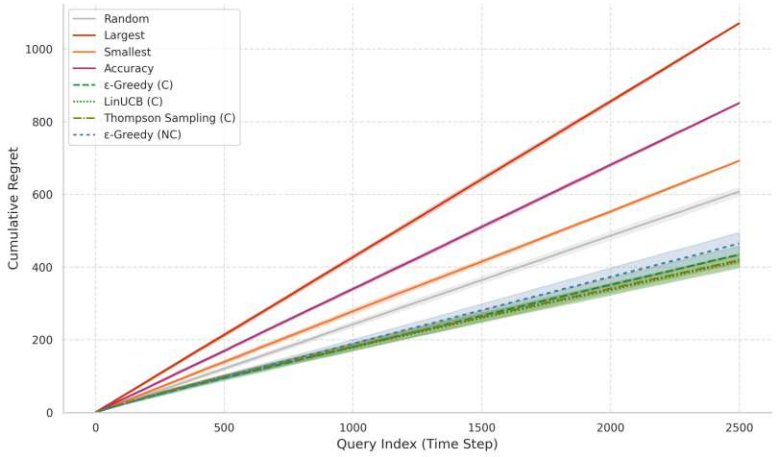

Figure 7.6: **Cumulative Regret Comparison of MAB Algorithms.** The plot shows the cumulative regret over time for different MAB algorithms using the Full Features context. Shaded areas represent 95% confidence intervals.

during the first ≈ 50 steps of exploration. Following this phase, both non-contextual and contextual algorithms stabilize relatively quickly. The non-contextual ϵ-Greedy stabilizes at a higher average regret level (≈ 0.18 vs. ≈ 0.16). In contrast, the contextual algorithms converge a bit slower to a slightly lower and more stable average regret range.

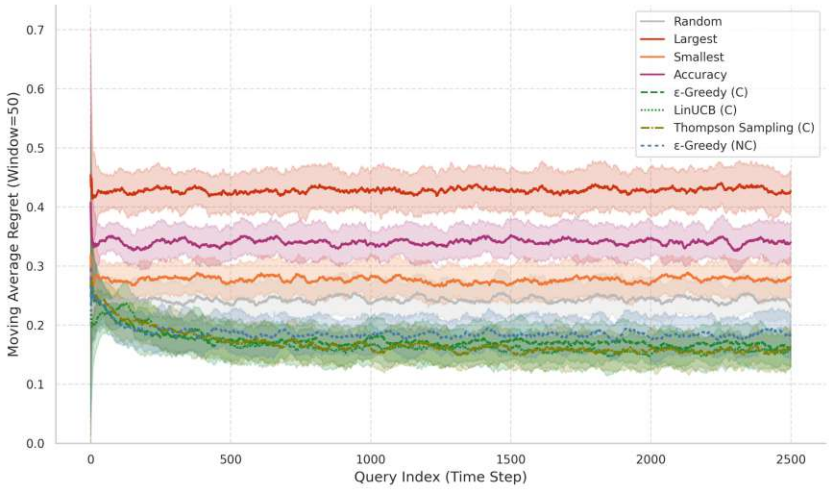

Figure 7.7: **Moving Average Regret Comparison.** The plot shows the moving average (window=50) regret over time for different MAB algorithms using the Full Features context, smoothing short-term fluctuations.

The frequently crossing lines of the contextual strategies visible in Figure 7.7 suggests that learned policies may be similar yet not identical. Figure 7.8 illustrates these nuances in model selection patterns across the four bandit algorithms. Similar to our initial bandit experiment in Section 7.3.2, certain models (Llama-3.1-8B, Phi-4-Mini-4B) demonstrate high selection frequency for multiple algorithms. The contextual algorithms (C) show more distributed selection patterns compared to the non-contextual (NC) variant. This indicates the ability to learn more fine-grained distinctions in model performance, which is particularly evident in the middle-tier models, that are consistent with the high density of options in this area. Collectively, such actions suggest the algorithms are identifying niches where certain models excel despite them not being optimal for all queries.

7.3.4 Trade-off Analysis (λ Sweep)

To understand how our dynamic routing system behaves across different trade-off preferences, we systematically vary the λ parameter in the reward function (see Section 4.4.3) from 0 (optimizing purely for accuracy) to 1 (optimizing purely for energy efficiency) in increments of 0.1 as presented in Figure 7.9. We evaluate the effectiveness of the four MAB strategies at each of these λ values.

Figure 7.9 presents the distribution of mean normalized accuracy and total energy consumption across multiple runs (n=20) for each algorithm as λ changes. As intended,

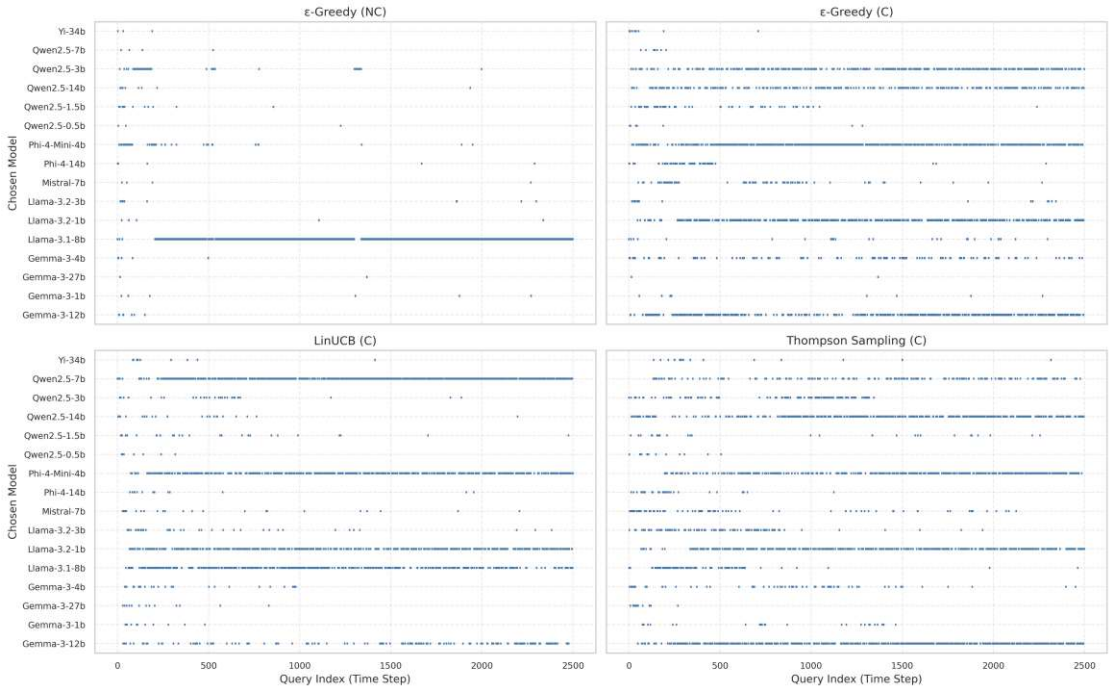

Figure 7.8: **Model Selection Timeline.** Visualizing the sequence of models chosen by a specific algorithm during a single run ($\lambda = 0.4$).

increasing λ leads to a clear trend. Both accuracy and energy consumption decrease as λ increases, which demonstrates the system's ability to prioritize either objective when instructed. The contextual MAB algorithms show similar trends, maintaining slightly higher accuracy, lower energy consumption and greater robustness compared to the non-contextual ϵ-Greedy across most λ values.

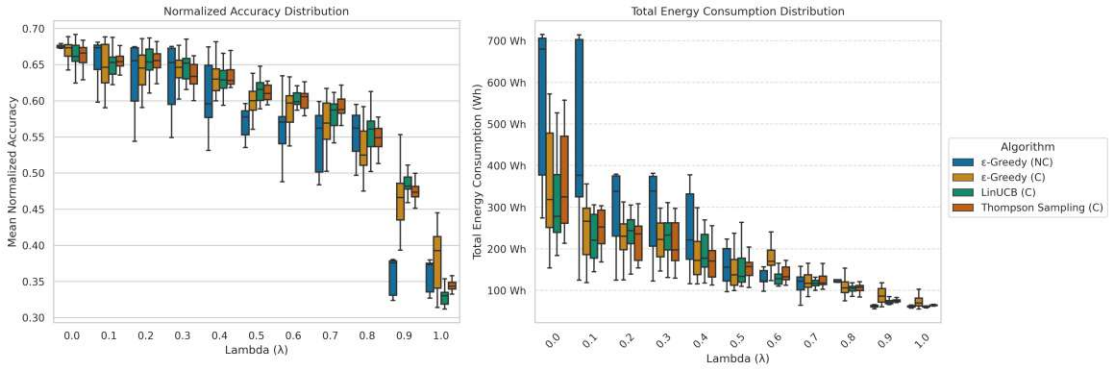

Figure 7.9: **Impact of λ on Accuracy and Energy Distributions.** Boxplots showing the distribution of Mean Normalized Accuracy (left) and Total Energy Consumption (right) for different MAB strategies across λ values (n=20).

Figure 7.10 provides another perspective on the trade-off by showing the resulting strategy aggregates on the accuracy-energy for different λ values (shown for increments of 0.2 for clarity). Each point represents the average accuracy-energy outcome of a strategy at a specific λ value. As λ increases, MAB results follow the Pareto front from the upper-right towards the lower-left. Remarkably, the contextual strategies consistently operate close to or beyond the static Pareto front (red dashed line).

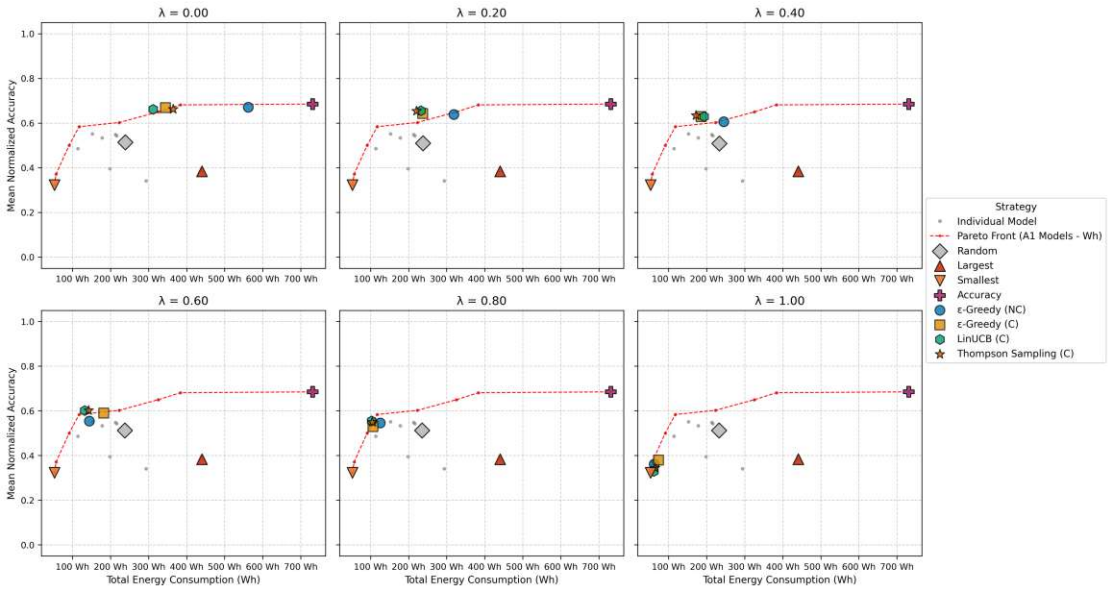

Figure 7.10: **Investigating the Accuracy-Energy Trade-off with λ.** Mean Normalized Accuracy vs. Total Energy Consumption (Wh) for different strategies at varying λ values (0.0 to 1.0, increments of 0.2). The static Pareto front is shown for reference.

7.3.5 Impact of Contextual Features

Having compared the different MAB algorithms using the full feature set, we now investigate the contribution of the different contextual signals. We evaluate different combinations of the three extracted features (task type, semantic cluster, and text complexity) with one of the contextual bandit algorithms (Contextual ϵ-Greedy). For the case without features, we employ the standard non-contextual ϵ-Greedy algorithm.

Figure 7.11 presents the distribution of final cumulative regret across 50 independent runs for each feature configuration. As earlier, lower values indicate lower regret. The results in Figure 7.11 cover all feature combinations: the context-free baseline (*None*), the three single features (*Task*, *Cluster*, *Complexity*), the three pairs (*Task + Cluster*, *Task + Complexity*, *Cluster + Complexity*), and the full feature set (*Full*).

Our analysis shows varying contribution of features. Both *Cluster* and *Complexity* result in regret slightly lower than the implementation without features. However, the most substantial reduction in regret appears to be linked to the inclusion of the *Task*

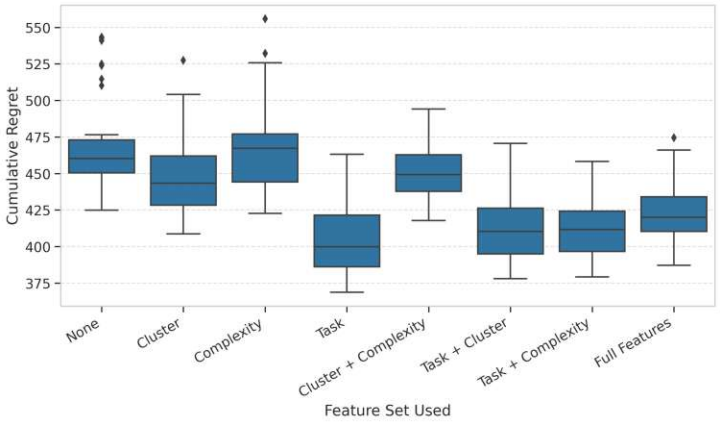

Figure 7.11: **Cumulative Regret Distribution by Feature Set.** Boxplots showing the distribution of final cumulative regret across multiple runs (n=50) for different contextual feature configurations at $\lambda = 0.4$.

feature, dropping median cumulative regret to ≈ 400. This identifies the task type as the single most informative component of context for guiding model selection in our setup. Combinations involving the *Task* feature (*Task + Cluster*, *Task + Complexity*) retain or slightly improve this level of regret reduction. However, including all features appears to raise regret levels notably. This might be attributed to the increased dimensionality which potentially slows convergence for the MABs during learning or introduce noise from less interactions informative feature compared to the strong signal provided by the task type alone.

Figure 7.12 further presents how context influences model selection behavior by showing the selection frequency of each model. The top heatmap displays patterns for the first half of the experiment (1-1250), while the bottom shows model selections for the second half (T=1251 to T=2500). Both are averaged over the last ten runs of the experiment. During the half, we observe more distributed selections across models as the algorithm actively explores the space. As more features are added to the configuration, exploration appears to increase further distributing loads between models. In contrast, the second half shows more concentrated selection patterns as policies stabilize. The baseline without features consistently favors Qwen2.5-7B (≈ 0.6 frequency in the second half), while contextual approaches develop more sophisticated strategies. When the *task* feature is included, the Contextual MAB appears to prefer Llama-3.2-1B (≈ 0.17) and Phi-4-mini-4B (≈ 0.19) amongst others. The *full* configuration demonstrates the most spread-out policies as it tries to match query requirements precisely.

7.3.6 Adaptability: Model Addition

To evaluate the system's ability to adapt to changes in the available model pool, we present an experiment where a new, competitive model (Gemma-3-12b) was introduced

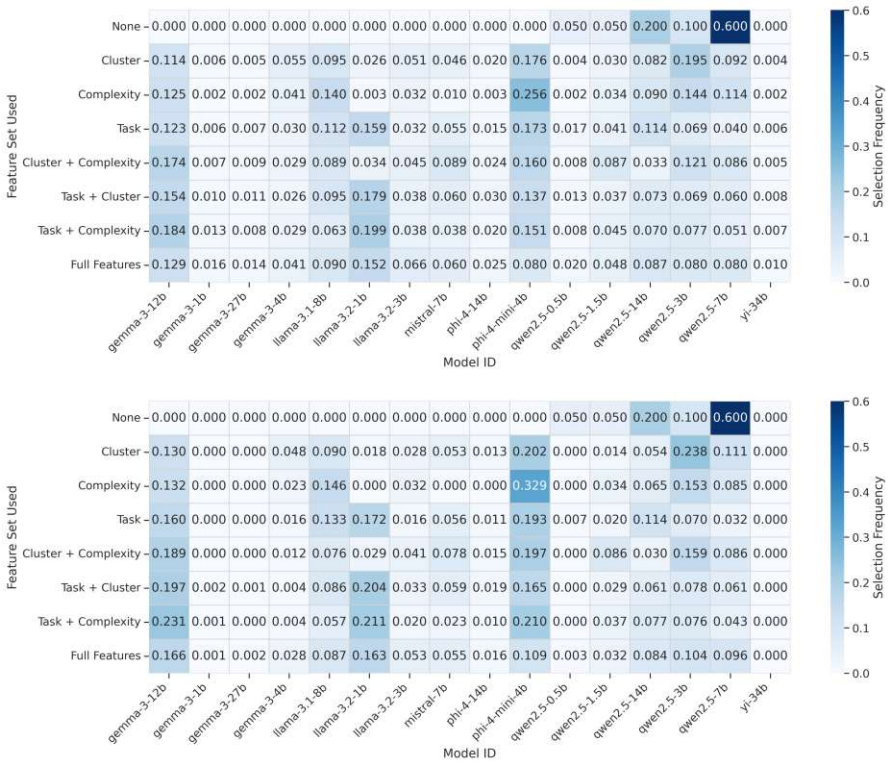

Figure 7.12: **Model Selection Frequency Heatmaps for Different Feature Sets.** The heatmaps show the selection frequency of each model for different feature configurations across runs (n=10). The top heatmap shows selections during the first half of the experiments (1-1250), while the bottom shows the second half (1251-2500). Darker blue indicates higher selection frequency. Without contextual features (None), selections concentrate on fewer models, while adding features leads to more diverse selection patterns that become increasingly focused as the algorithm learns.

only during mid-run after 1000 queries. We used the contextual Thompson Sampling algorithm with full features and $\lambda = 0.2$ to clearly favor the new high-accuracy model.

Figure 7.13 shows the mean selection frequency of each model over the course of the experiment with a window size of 25. Before the adaptation point (black line), the newly added model (Gemma-3-12b, dark blue) correctly shows zero selection frequency as it was not yet part of the pool. Immediately after its introduction, the algorithm begins to explore it, and its selection frequency rapidly increases. After a around 100 queries, the selection frequency stabilizes at around 20%-25%. This shifted selection frequency visibly comes at the cost of previously frequently selected models, which indicates successful adaptation by the system to the new model pool by integrating the model into its routing strategy.

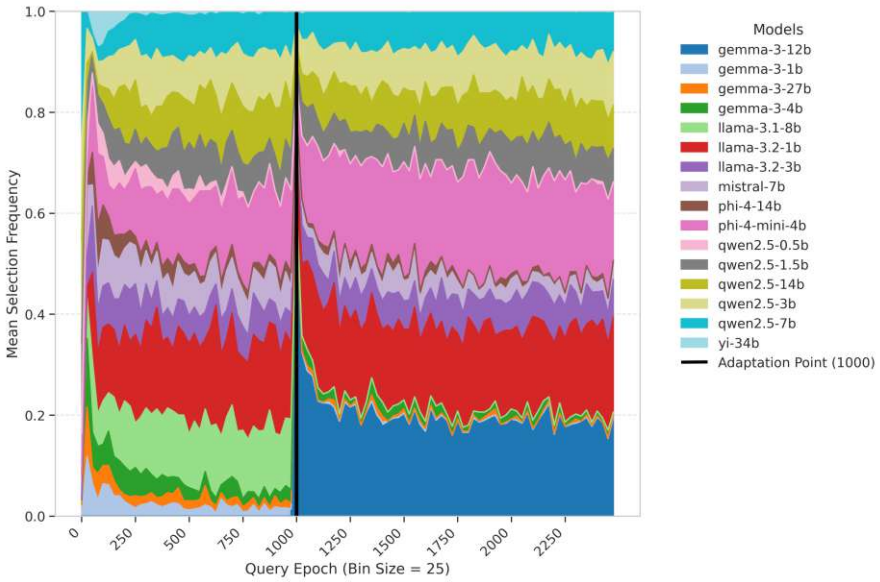

Figure 7.13: **Model Selection Frequency during Model Pool Change.** Mean selection frequency (stacked area) over time using Contextual Thompson Sampling ($\lambda = 0.2$) for a single run. Gemma-3-12b (dark blue) is added at Query Step 1000 (black line).

7.3.7 Overhead Analysis

Table 7.4: **Average Overhead per Query Component.** The table shows the average time taken (in milliseconds) for each step involved in the feature extraction and routing decision process per query.

Component	Avg. Time per Query (ms)
Task Type Classification	3.04
Semantic Cluster Identification	3.37
Text Complexity Calculation	0.15
ϵ-Greedy Routing Decision	0.02
LinUCB Routing Decision	0.86
Contextual Thompson Sampling Routing Decision	1.21
Total Pre-Inference Overhead	**6.68-7.77**

Our evaluation shows that dynamic routing offers potential benefits in optimizing accuracy and energy efficiency. However, it introduces computational overhead prior to performing model inference. This overhead comprises two main components: feature extraction and the routing decision process. While feedback computation entails some overhead, it can happen asynchronously to the model inference to not affect latency. Table 7.4 provides information on the overhead per query component.

Table 7.5: **Model Inference Latency Statistics.** Latency distribution metrics (ms per query) for each model across the evaluation dataset. Final column shows the relative overhead added by routing (assuming a 7.8 ms fixed overhead) compared to the median latency.

Model	Min	P25	Median	Average	P75	Max	Overhead (%)
Llama-3.2-1B	19.8	34.6	36.1	570.8	1,196.7	2,401.6	21.6
Qwen2.5-0.5B	14.0	43.2	44.3	686.1	1,402.9	3,350.8	17.6
Qwen2.5-1.5B	20.0	48.9	50.6	976.1	1,926.0	3,907.8	15.4
Llama-3.2-3B	16.4	47.1	51.6	977.9	1,955.6	3,936.4	15.1
Phi-4-mini-4B	18.3	47.6	52.7	982.6	1,605.1	4,557.5	14.8
Qwen2.5-3B	40.2	50.4	53.8	1,351.4	2,078.4	4,995.9	14.5
Qwen2.5-7B	38.4	48.0	56.4	1,880.2	3,525.7	10,523.9	13.8
Llama-3.1-8B	42.1	51.5	60.8	1,366.5	2,491.6	4,603.7	12.8
Mistral-7B	42.1	51.4	60.9	1,103.2	2,345.4	4,324.3	12.8
Gemma-3-1B	52.4	57.3	68.2	2,848.3	5,026.1	12,666.0	11.4
Phi-4-14B	33.9	62.7	76.8	1,553.8	2,420.8	6,790.7	10.2
Gemma-3-4B	72.6	75.8	81.1	2,225.8	3,595.8	9,039.5	9.6
Gemma-3-12B	48.2	61.3	83.0	3,104.4	5,771.0	14,296.9	9.4
Qwen2.5-14B	66.6	70.7	83.8	1,786.2	2,208.0	7,387.5	9.3
Yi-34B	71.5	113.8	163.4	2,195.5	3,924.0	14,571.3	4.8
Gemma-3-27B	158.4	167.1	199.7	5,088.3	6,639.8	37,972.6	3.9

Combined, the total average overhead per query added by our system is approximately 6.68-7.77 ms when processed sequentially. This overhead should be evaluated in perspective to the actual inference times, which varied significantly across our model pool, as detailed in Table 7.5. Furthermore, as these overhead tasks are primarily CPU-bound, they could potentially run during queue wait times in larger scale systems to reduce the drawbacks on end-to-end latency. Whether this overhead is acceptable for the gains achieved through our routing system depends on the specific latency requirements of an application. For batch processing or non-critical use cases, the resulting overhead is likely negligible compared to the overall inference cost. However, clearly, for latency-sensitive applications, feature extraction and routing steps could benefit from further optimization. This measured overhead has to be considered similarly to inference latency estimates as the latency estimates used for feasibility checks in Section 5.3.2.

7.4 Discussion

7.4.1 Summary of Key Findings

Dynamic routing with contextual bandits and lightweight features consistently outperformed all static and random baselines. At the default trade-off ($\lambda = 0.4$), our proposed strategies consistently exceeded boundaries set by the Pareto front presenting accuracy-energy operating points that were unreachable by single-model deployments in our experiments. Compared to the most balanced baselines (random), we achieved

accuracy gain of 22% while reducing energy consumption by 31%. Compared to the Smallest baseline, we improved accuracy by 90-100%, however, at a drastically increased energy consumption (+400%). In exchange for an accuracy degradation of 10-12%, were able to reduce energy consumption by 75%. Among algorithms, Contextual ϵ-Greedy, Contextual Thompson Sampling, and LinUCB converged to similar regret values, suggesting that feature engineering, model pool selection, and reward function design may matter more than the specific contextual bandit algorithm choice in this setup.

7.4.2 Implications for Dynamic LLM Routing

These results suggest that expensive reinforcement learning or offline-learned reward models are not a prerequisite for more efficient real-time model selection. A bandit that ingests three lightweight features can move the operating point of an inference system along the accuracy-energy front during runtime. Because our policy is model-agnostic, new model releases do not require retraining of the router policy which is critical in a field where state-of-the-art is redefined monthly. Finally, normalizing energy on a per-token basis yields a unified metric that allows comparing compute budgets across heterogeneous workloads.

7.4.3 Research Questions

Based on the findings presented in Chapter 7, we can now revisit the research questions posed in Chapter 1:

RQ_1: What key factors influence the accuracy-efficiency trade-off in large language model inference?
Our investigation confirmed that the accuracy-efficiency trade-off in LLM inference is significantly influenced by both inherent model characteristics (parameter count, architecture, training) and query-specific properties. As detailed in Section 2.3.4, empirical evidence shows substantial query-dependent performance variation, meaning models exhibit different accuracy and resource consumption levels based on the task type, domain, and complexity of the input query. To represent these factors quantitatively, we first defined measurable metrics: accuracy (using objective scores like Exact Match or ROUGE) and efficiency (using measured energy consumption as a primary indicator). We then formalized the problem as a Multi-Objective Optimization Problem (MOOP) in Chapter 4, aiming to maximize accuracy while minimizing energy consumption. Query context was explicitly incorporated by defining a feature extraction process (Section 4.2) yielding a context vector x_t (including task type, semantic cluster, complexity), and latency was handled as a practical feasibility constraint L_{max} rather than a third objective to maintain tractability (Section 4.4). This MOOP formulation, combining context representation, objective metrics, and constraints, provides the quantitative basis for the dynamic routing approach.

RQ_2: How can a dynamic routing system effectively manage the accuracy-efficiency trade-off online?

This thesis investigated contextual Multi-Armed Bandit (MAB) algorithms as a class of effective dynamic routing strategies suitable for managing the identified factors. MABs inherently handle the exploration-exploitation trade-off under the partial feedback conditions typical of this problem (Section 4.5). Their ability to incorporate contextual information (query features) allows them to adapt routing decisions based on the varying accuracy-efficiency performance of different models across diverse queries (a key factor from RQ1). We specifically implemented and evaluated Contextual ϵ-Greedy, LinUCB, and Contextual Thompson Sampling (Chapter 5). Our experiments demonstrated that these strategies effectively learn online to approximate optimal solutions within the MOOP framework, consistently outperforming static and non-contextual approaches and navigating the accuracy-energy Pareto front (Section 7.3.3, 7.3.4). While other complex strategies exist (e.g., Deep Reinforcement Learning), this work establishes contextual MABs as a highly effective and computationally lighter approach for this dynamic routing problem.

RQ_3: How can the effectiveness of the proposed dynamic routing system be validated under simulated operational conditions?
To evaluate the proposed MAB strategies, we designed an experimental methodology based on controlled, real-time simulation (Chapter 7). This involved processing a sequential stream of queries drawn from diverse benchmarks through the implemented routing framework connected to a pool of 16 contemporary LLMs. Efficacy was measured quantitatively using key metrics relevant to the MOOP and bandit setting: cumulative regret (to assess learning efficiency), final average accuracy, and total energy consumption (Sections 4.3, 4.6). We validated the strategies' effectiveness by comparing these metrics against several baselines (static single-model policies and random routing) under identical simulated conditions (Sections 7.3.1, 7.3.3). This simulation-based evaluation allowed us to gain crucial insights, confirming the positive impact of context (Section 7.3.5), the system's adaptability to new models (Section 7.3.6), and quantifying the associated latency overhead (Section 7.3.7).

7.4.4 Limitations

Our study has several limitations that suggest opportunities for future work:

- **Fixed Query Set and Benchmarks:** The evaluation relies on a fixed set of 2,500 queries derived from five specific benchmarks. While diverse, this static query pool implies the evaluation does not demonstrate the system's adaptability to shifting workloads or its robustness to the distribution shifts inherent in real-world query streams. Consequently, the experiments do not validate effectiveness on considerably broader query categories beyond those sampled.

- **Hardware Specificity:** Energy measurements and latency profiles were obtained on a single server equipped with an NVIDIA A100 GPU. Measurements could

differ significantly on other hardware platforms like CPUs, TPUs, or different GPU architectures.

- **Metrics:** The analysis focuses on mean accuracy, cumulative energy consumption and regret metrics with confidence intervals. More granular insights into the distribution of per-request outcomes could be beneficial.

- **Experiment Duration and Scalability:** Experiments were conducted over T=2,500 queries which was sufficient to demonstrate learning convergence and the benefits of a later exploration phase. However, due to scope of this work, we did not explore variation in query stream, or system behavior under significantly larger scales.

- **Hyperparameter Sensitivity:** The impact of contextual features was evaluated. However, the sensitivity to specific feature engineering choices (number of clusters K, complexity bins N) was not exhaustively explored and relied on preliminary tuning.

- **Overhead Impact:** As detailed in Section 7.3.7, the feature extraction and routing decisions introduce a noticeable overhead (6.68-7.77 ms). Table 7.5 shows that relative to actual inference times, this overhead could render the system impractical in highly latency-sensitive applications and represents a factual cost for the achieved routing benefits.

- **Simulated vs. Operational Conditions Discrepancy:** We acknowledge limitations of the simulated vs. real-world operational conditions. Our evaluation lacks aspects like concurrency of requests and batch processing. While we include the MABs' initial exploration-heavy phase (cold start), the simulation neglects the potentially substantial costs of *true* system cold starts (e.g., initial downloading and storage of model weights). Additionally, crucial overheads like network latency, queueing delays, and switching of models. We assumed models pre-loaded and that they fit within available GPU memory. The results presented should be interpreted accordingly.

- **Lack of Latency Constraint Application:** The latency feasibility constraint (L_{max}) to discard models excessively slow for a given query on the used hardware was not enforced in our evaluation. Consequently, results do not allow insights into the system's effectiveness under strict latency requirements.

- **Linear Reward Model Assumption:** The contextual MAB algorithms rely on a linear model to approximate the true reward function based on the context features. Combined with the one-hot encoding of categorical features, this might represent a potentially too restrictive approximation. If the true reward function involves non-linearities like complex interactions between features (e.g., specific task-complexity combinations), the system will not be able to reflect these.

- **Static Normalization Bounds:** The min-max normalization for accuracy and energy consumption is calculated on the basis of bounds established during initial profiling. While the system adapts to new models (Section 7.3.6), a new model significantly outside the previously observed performance range will skew the normalized [0,1] scale which potentially impacts the reward calculation. Either bounds are updated, requiring rescaling of previous observations, or the new model will not be appropriately represented by the accuracy and energy consumption values assigned.
- **Assumption of Stationarity:** The implemented MAB algorithms implicitly assume stationary reward distributions. They rely on cumulative statistics (e.g., averages for ϵ-Greedy, cumulative matrices for LinUCB) which means once fitted, they adapt slowly to potential drifts in model performance or query characteristics over longer periods. Based on the fixed evaluation dataset applied, current evaluation does not test robustness in non-stationary settings.

Summary

This chapter evaluated the dynamic routing framework, demonstrated its effectiveness in surpassing static baselines and maneuvering the accuracy-energy trade-off along or beyond the Pareto front. Key insights include the significant impact of contextual features (especially task type) on regret, the system's adaptability to new models, and the quantifiable overhead introduced by the routing logic. Limitations clearly exist, but the results validate the MAB-based approach for efficient, context-aware LLM inference routing in an online setting under partial feedback. The final chapter synthesizes this work and provides concluding remarks.

CHAPTER 8

Conclusions

In this thesis, we investigated the challenge of optimizing Large Language Model (LLM) inference for both accuracy and energy efficiency. To overcome existing shortcomings in existing approaches in the literature, we designed, implemented, and evaluated a dynamic routing framework that employs multi-armed bandits (MABs) to select the most suitable LLM out of a pool of 16 available models for each incoming query based on the query's context.

8.1 Summary of Contributions

First, we addressed the factors influencing the accuracy-efficiency trade-off in LLM inference (RQ1). We established a theoretical and formal basis by framing the dynamic LLM routing task as a multi-objective optimization problem (MOOP) that explicitly targets the trade-off between response accuracy and measured energy consumption and key problem properties. We utilized direct energy consumption measurements (approximated via GPU power draw) rather than common proxy metrics to better reflect actual resource usage. Furthermore, we mapped the problem to a contextual bandit setting to account for the partial feedback and uncertainty inherent in online decision-making, a facet yet little explored in the routing literature.

Second, to effectively manage this trade-off (RQ2), we proposed and developed a dynamic, context-aware routing framework. Our core methodological contribution is an online learning system that includes lightweight feature extraction (task type, semantic cluster, text complexity) to inform subsequent model selection. For decision-making, the framework utilizes contextual multi-armed bandit (MAB) algorithms (Contextual ϵ-Greedy, LinUCB, Contextual Thompson Sampling) which continuously adapt their routing policy based on observed (partial) performance, which crucially eliminates the need for extensive offline calibration often seen in the literature.

Finally, we validated the effectiveness and adaptability of this approach through empirical evaluation (RQ3). We used a diverse mix of benchmark queries and a pool of 16 LLMs of various sizes and capabilities. Our experiments demonstrated that the dynamic routing system consistently outperforms static baselines and the included non-contextual MAB variant. It achieves superior operating points along the accuracy-energy Pareto front, often exceeding it. In some settings, contextual strategies yielded accuracy gains of 22% while reducing cumulative energy consumption by 31% compared to random routing. The results also support the framework's ability to effectively align system objectives to specified preference (λ) and that it successfully adapts its policy to new models during runtime without recalibration.

8.2 Limitations

While our results are promising, this study, as discussed in Section 7.4, has several limitations that bound the generalizability of its findings:

- The evaluation used a fixed set of queries from specific benchmarks, which may not capture the full diversity of real-world workloads or open-ended generation tasks.
- Energy and latency measurements were specific to the hardware used; results might differ on other platforms.
- The analysis focused on aggregate metrics (mean accuracy, total energy, cumulative regret); finer-grained per-query outcome distributions could offer more insights.
- The experiment duration, while sufficient for observing learning convergence, did not explore long-term behavior or performance under drastic distribution shifts.
- Sensitivity to feature engineering hyperparameters (e.g., number of clusters, complexity bins) was not exhaustively studied.
- Certain deployment factors, such as concurrent query handling, batching strategies, and model loading/unloading overheads, were simplified or omitted.
- The pre-inference overhead (feature extraction, routing decision) introduces latency (approx. 7-8 ms) that might be prohibitive for highly latency-sensitive applications.

8.3 Future Directions

The findings and limitations of this thesis open several avenues for future research aimed at enhancing the framework's capabilities and exploring new applications:

Algorithmic and Objective Refinements

Future work could focus on improving the core decision-making intelligence of the routing framework. This includes exploring richer contextual features (e.g., user history, session context) and more sophisticated reward modeling techniques, such as non-linear reward models or distributional reinforcement learning, to potentially improve routing decisions, especially for complex or nuanced queries. Furthermore, investigating advanced MAB algorithms specifically designed for non-stationary environments, incorporating safety constraints, or leveraging meta-learning could allow faster adaptation to new tasks or models. Refining the handling of multiple objectives, for instance by integrating latency more directly as a penalized objective or exploring explicit Pareto optimization algorithms, could also provide finer-grained control in diverse deployment scenarios. Additionally, improving the robustness of metric normalization to mitigate performance drift could involve the investigation of dynamic bound updates, e.g., through periodic recalibration, moving windows, or statistics less prone to outliers.

System Implementation and Deployment Challenges

Addressing practical deployment challenges represents another key research direction. Extending the framework to handle concurrent queries and incorporating dynamic batching strategies are crucial steps for compatibility with production systems; this requires analyzing the impact on both system efficiency and MAB learning dynamics. Additionally, further investigation into methods for reducing the latency overhead of feature extraction and routing is warranted. Potential approaches include model distillation for the classifier, feature caching, asynchronous processing of non-critical path components, or even hardware acceleration for feature computation.

Evaluation Scope and Robustness

Finally, enhancing the validation scope and assessing long-term robustness is critical to ensure the system's long-term viability. Evaluating the framework on a wider range of tasks, including more open-ended generation benchmarks and out-of-domain queries, and across different hardware platforms (CPUs, TPUs, other GPUs) would strengthen claims of generalizability. Investigating performance over longer time horizons and explicitly testing resilience against various types of distribution shifts (e.g., changes in query topics, model performance degradation) are required to better understand the framework's robustness against *concept drift* in real-world conditions.

In summary, this thesis demonstrates that dynamic, context-aware routing using lightweight online learning algorithms offers a practical and effective approach for optimizing LLM inference. By strategically balancing accuracy and energy consumption based on query context and real-time feedback, the proposed framework provides a significant step towards more sustainable, efficient, and adaptable AI deployments, mitigating the limitations of static strategies in the face of the diverse and evolving LLM landscape.

Overview of Generative AI Tools Used

I confirm that AI tools were used solely as supportive aids in writing this thesis. The up-to-date version of GPT-4o via ChatGPT, Writefull via Overleaf, and Grammarly were used exclusively for checking grammar and spelling, as well as improving wording and sentence structure. No content was generated solely through prompts and included without substantial original input and revision.

Übersicht verwendeter Hilfsmittel

Ich bestätige, dass KI-Tools ausschließlich als unterstützende Hilfsmittel beim Verfassen dieser Arbeit dienten. Die tagesaktuelle Version von GPT-4o über ChatGPT, Writefull über Overleaf und Grammarly wurden ausschließlich zur Überprüfung von Grammatik und Rechtschreibung sowie zur Verbesserung von Formulierungen und Satzstruktur verwendet. Es wurde kein Inhalt ausschließlich durch Prompts generiert und ohne wesentliche eigene inhaltliche Beiträge und Überarbeitung übernommen.

List of Figures

List of Tables

List of Algorithms

Bibliography

[AAA+23a] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, and et al. GPT-4 Technical Report. *arXiv:2303.08774*, 2023.

[AAA+23b] Ebtesam Almazrouei, Hamza Alobeidli, Abdulaziz Alshamsi, Alessandro Cappelli, Ruxandra Cojocaru, Mérouane Debbah, Étienne Goffinet, Daniel Hesslow, Julien Launay, Quentin Malartic, and et al. The Falcon Series of Open Language Models. *arXiv:2311.16867*, 2023.

[AG13] Shipra Agrawal and Navin Goyal. Thompson Sampling for Contextual Bandits with Linear Payoffs. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 127–135. PMLR, 2013.

[ALTdJ+23] Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebron, and Sumit Sanghai. GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Proceedings of the 2023 conference on empirical methods in natural language processing*, pages 4895–4901, Singapore, 2023.

[AMA+24] Pranjal Aggarwal, Aman Madaan, Ankit Anand, Srividya Pranavi Potharaju, Swaroop Mishra, Pei Zhou, Aditya Gupta, Dheeraj Rajagopal, Karthik Kappaganthu, Yiming Yang, and et al. Automix: Automatically Mixing Language Models. *Advances in Neural Information Processing Systems (NeurIPS)*, 37:131000–131034, 2024.

[ARA+22] Reza Yazdani Aminabadi, Samyam Rajbhandari, Ammar Ahmad Awan, Cheng Li, Du Li, Elton Zheng, Olatunji Ruwase, Shaden Smith, Minjia Zhang, Jeff Rasley, and et al. Deepspeed-Inference: Enabling Efficient Inference of Transformer Models at Unprecedented Scale. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis (SC22)*, pages 1–15. IEEE, 2022.

[BB94] Leon Bottou and Yoshua Bengio. Convergence Properties of the K-Means Algorithms. *Advances in Neural Information Processing Systems (NeurIPS)*, 7:585–592, 1994.

[BKH16] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer Normalization. *arXiv:1607.06450*, 2016.

[BMR+20] 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, and et al. Language Models are Few-Shot Learners. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pages 1877–1901. Curran Associates, Inc., 2020.

[BN06] Christopher M Bishop and Nasser M Nasrabadi. *Pattern Recognition and Machine Learning*. Springer, 2006.

[BSF94] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning Long-Term Dependencies with Gradient Descent Is Difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994.

[BSMC24] Alberto Benayas, Miguel Angel Sicilia, and Marçal Mora-Cantallops. A Comparative Analysis of Encoder Only and Decoder Only Models in Intent Classification and Sentiment Analysis: Navigating the Trade-Offs in Model Size and Performance. *Language Resources and Evaluation*, pages 1–24, 2024.

[CGRS19] Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating Long Sequences with Sparse Transformers. *arXiv:1904.10509*, 2019.

[CJL+24] Shuhao Chen, Weisen Jiang, Baijiong Lin, James Kwok, and Yu Zhang. RouterDC: Query-Based Router by Dual Contrastive Learning for Assembling Large Language Models. *Advances in Neural Information Processing Systems (NeurIPS)*, 37:66305–66328, 2024.

[CKB+21] 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. Training Verifiers to Solve Math Word Problems. *arXiv:2110.14168*, 2021.

[CND+23] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Xavier Garcia, Denny Zhou, Daphne Ippolito, Hyeontaek Lim, and Barret Zoph. PaLM: Scaling Language Modeling with Pathways. *Journal of Machine Learning Research*, 24(1), 2023.

[Com25] McKinsey & Company. The State of AI: How Organizations Are Rewiring to Capture Value. Industry Report, 2025. Available at: https://www.mckinsey.com/capabilities/quantumblack/our-insights/the-state-of-ai Accessed: 2025-05-04.

[CWW+24] Yupeng Chang, Xu Wang, Jindong Wang, Yuan Wu, Linyi Yang, Kaijie Zhu, Hao Chen, Xiaoyuan Yi, Cunxiang Wang, Yidong Wang, and et al. A Survey on Evaluation of Large Language Models. *ACM Transactions on Intelligent Systems and Technology*, 15(3):1–45, 2024.

[CZZ23] Lingjiao Chen, Matei Zaharia, and James Zou. FrugalGPT: How to Use Large Language Models While Reducing Cost and Improving Performance. *arXiv:2305.05176*, 2023.

[DCLT19] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-Training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of NAACL-HLT 2019*, pages 4171–4186, Minneapolis, MN, 2019. ACL.

[DFE+22] Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness. *Advances in Neural Information Processing Systems (NeurIPS)*, 35:16344–16359, 2022.

[DHMS23] Tri Dao, Daniel Haziza, Francisco Massa, and Grigory Sizov. Flash-Decoding for Long-Context Inference. *Stanford Center for Research on Foundation Models*, 2023.

[DMW+24] Dujian Ding, Ankur Mallick, Chi Wang, Robert Sim, Subhabrata Mukherjee, Victor Rühle, Laks VS Lakshmanan, and Ahmed Hassan Awadallah. Hybrid LLM: Cost-Efficient and Quality-Aware Query Routing. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2024.

[dV23] Alex de Vries. The Growing Energy Footprint of Artificial Intelligence. *Joule*, 7(10):2191–2194, 2023.

[FA23] Elias Frantar and Dan Alistarh. SparseGPT: Massive Language Models Can Be Accurately Pruned in One-Shot. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 10323–10337. PMLR, 2023.

[FAHA23] Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. OPTQ: Accurate Quantization for Generative Pre-Trained Transformers. In *Proceedings of International Conference on Learning Representations (ICLR)*, 2023.

[Fle48] Rudolph Flesch. A New Readability Yardstick. *Journal of Applied Psychology*, 32(3):221, 1948.

[FLY+23] Zihao Fu, Wai Lam, Qian Yu, Anthony Man-Cho So, Shengding Hu, Zhiyuan Liu, and Nigel Collier. Decoder-Only or Encoder-Decoder? Interpreting Language Model as a Regularized Encoder-Decoder. *arXiv:2304.04052*, 2023.

[FSY25] Tao Feng, Yanzhen Shen, and Jiaxuan You. GraphRouter: A Graph-Based Router for LLM Selections. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2025.

[GCC+24] Neel Guha, Mayee Chen, Trevor Chow, Ishan Khare, and Christopher Re. Smoothie: Label Free Language Model Routing. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 37, pages 127645–127672, 2024.

[GD24] Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. In *First conference on language modeling*, 2024.

[GDJ+24] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, and et al. The Llama 3 Herd of Models. *arXiv:2407.21783*, 2024.

[GGR22] Albert Gu, Karan Goel, and Christopher Ré. Efficiently Modeling Long Sequences with Structured State Spaces. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2022.

[GSS+24] Samir Yitzhak Gadre, Georgios Smyrnis, Vaishaal Shankar, Suchin Gururangan, Mitchell Wortsman, Rulin Shao, Jean Mercat, Alex Fang, Jeffrey Li, Sedrick Keh, and et al. Language Models Scale Reliably with Over-Training and on Downstream Tasks. *arXiv:2403.08540*, 2024.

[HBB+21] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring Massive Multitask Language Understanding. In *Proceedings of International Conference on Learning Representations (ICLR)*, 2021.

[HBL+24] Qitian Jason Hu, Jacob Bieker, Xiuyu Li, Nan Jiang, Benjamin Keigwin, Gaurav Ranganath, Kurt Keutzer, and Shriyash Kaustubh Upadhyay. RouterBench: A Benchmark for Multi-LLM Routing System. In *Agentic Markets Workshop at ICML*, 2024.

[HBM+22] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, and et al. Training Compute-Optimal Large Language Models. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 30016–30030, 2022.

[HDX+23] Ke Hong, Guohao Dai, Jiaming Xu, Qiuli Mao, Xiuhong Li, Jun Liu, Kangdi Chen, Yuhan Dong, and Yu Wang. FlashDecoding++: Faster Large Language Model Inference on GPUs. *arXiv:2311.01282*, 2023.

[HGSS24] Erik Johannes Husom, Arda Goknil, Lwin Khin Shar, and Sagar Sen. The Price of Prompting: Profiling Energy Use in Large Language Models Inference. *arXiv:2407.16893*, 2024.

[HK70] Arthur E Hoerl and Robert W Kennard. Ridge Regression: Biased Estimation for Nonorthogonal Problems. *Technometrics*, 12(1):55–67, 1970.

[HKG+15] Karl Moritz Hermann, Tomas Kocisky, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. Teaching Machines to Read and Comprehend. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 28, pages 1693–1701, 2015.

[HT23] Surya Narayanan Hari and Matt Thomson. Tryage: Real-time, intelligent Routing of User Prompts to Large Language Models. *arXiv:2308.11601*, 2023.

[HVD15] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the Knowledge in a Neural Network. *arXiv:1503.02531*, 2015.

[HYW+24] Qi Hu, Zhisheng Ye, Zerui Wang, Guoteng Wang, Mengdie Zhang, Qiaoling Chen, Peng Sun, Dahua Lin, Xiaolin Wang, Yingwei Luo, Yonggang Wen, and Tianwei Zhang. Characterization of Large Language Model Development in the Datacenter. *arXiv:2403.07648*, 2024.

[HZRS16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[Int25] International Energy Agency. Energy and AI, 2025. Available at: https://www.iea.org/reports/energy-and-ai Accessed: 2025-05-04.

[JNR+25] Wittawat Jitkrittum, Harikrishna Narasimhan, Ankit Singh Rawat, Jeevesh Juneja, Zifeng Wang, Chen-Yu Lee, Pradeep Shenoy, Rina Panigrahy, Aditya Krishna Menon, and Sanjiv Kumar. Universal LLM Routing with Correctness-Based Representation. In *First Workshop on Scalable Optimization for Efficient and Adaptive Foundation Models*, 2025.

[JWL+23] Huiqiang Jiang, Qianhui Wu, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. LLMLingua: Compressing Prompts for Accelerated Inference of Large Language Models. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Proceedings of EMNLP 2023*, pages 13358–13376, Singapore, 2023. Association for Computational Linguistics.

[KAS+25] Tanishq Kumar, Zachary Ankner, Benjamin Frederick Spector, Blake Bordelon, Niklas Muennighoff, Mansheej Paul, Cengiz Pehlevan, Christopher Re, and Aditi Raghunathan. Scaling Laws for Precision. In *Proceedings of International Conference on Learning Representations (ICLR)*, 2025.

[KLZ+23] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient Memory Management for Large Language Model Serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles (SOSP)*, pages 611–626, 2023.

[KMH+20] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling Laws for Neural Language Models. *arXiv:2001.08361*, 2020.

[KMM+19] Dhiraj Kalamkar, Dheevatsa Mudigere, Naveen Mellempudi, Dipankar Das, Kunal Banerjee, Sasikanth Avancha, Dharma Teja Vooturi, Nataraj Jammalamadaka, Jianyu Huang, Hector Yuen, and et al. A Study of BFLOAT16 for Deep Learning Training. *arXiv:1905.12322*, 2019.

[KVPF20] Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are RNNs: Fast Autoregressive Transformers with Linear Attention. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 5156–5165. PMLR, 2020.

[LCG+20] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. ALBERT: A Lite BERT for Self-Supervised Learning of Language Representations. *arXiv:1909.11942*, 2020.

[LCLS10] Lihong Li, Wei Chu, John Langford, and Robert E Schapire. A Contextual-Bandit Approach to Personalized News Article Recommendation. In *Proceedings of the International World Wide Web Conference (WWW)*, pages 661–670, 2010.

[LCO24] Chia-Hsuan Lee, Hao Cheng, and Mari Ostendorf. OrchestraLLM: Efficient Orchestration of Language Models for Dialogue State Tracking. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, pages 1434–1445, 2024.

[LDM12] Hector J. Levesque, Ernest Davis, and Leora Morgenstern. The Winograd Schema Challenge. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning (KR)*, KR'12, pages 552–561. AAAI Press, 2012.

[Li25] Yang Li. LLM Bandit: Cost-Efficient LLM Generation via Preference-Conditioned Dynamic Routing. *arXiv:2502.02743*, 2025.

[Lin04] Chin-Yew Lin. ROUGE: A Package for Automatic Evaluation of Summaries. In *Text summarization branches out*, pages 74–81, 2004.

[LKM23] Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast Inference from Transformers via Speculative Decoding. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 19274–19286. PMLR, 2023.

[LLG+20] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension. In Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel Tetreault, editors, *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 7871–7880, Online, July 2020. Association for Computational Linguistics.

[LOG+19] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. RoBERTa: A Robustly Optimized BERT Pretraining Approach. *arXiv:1907.11692*, 2019.

[LPP+20] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, and et al. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. *Advances in Neural Information Processing Systems (NeurIPS)*, 33:9459–9474, 2020.

[LS20] Tor Lattimore and Csaba Szepesvári. *Bandit Algorithms*. Cambridge University Press, 2020.

[LSFA+22] Teven Le Scao, Angela Fan, Christopher Akiki, Ellie Pavlick, Suzana Ilić, Daniel Hesslow, and et al. BLOOM: a 176B-parameter open-access multilingual language model. *arXiv:2211.05100*, 2022.

[LTT+24] Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. AWQ: Activation-Aware Weight Quantization for On-Device LLM Compression and Acceleration. In P. Gibbons, G. Pekhimenko, and C. De Sa, editors, *Proceedings of machine learning and systems*, volume 6, pages 87–100, 2024.

[LYZ+23] Yixiao Li, Yifan Yu, Qingru Zhang, Chen Liang, Pengcheng He, Weizhu Chen, and Tuo Zhao. LoSparse: Structured Compression of Large Language Models Based on Low-Rank and Sparse Approximation. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 20336–20350. PMLR, 2023.

[MA04] R.T. Marler and J.S. Arora. Survey of Multi-Objective Optimization Methods for Engineering. *Structural and Multidisciplinary Optimization*, 26(6):369–395, 2004.

[Met24] Meta AI. Introducing LLaMA 3.1: Our Most Capable Models to Date, 2024. Available at: https://ai.meta.com/blog/meta-llama-3-1/ Accessed: 2025-05-04.

[MLG23] Jesse Mu, Xiang Li, and Noah Goodman. Learning to Compress Prompts with Gist Tokens. *Advances in Neural Information Processing Systems (NeurIPS)*, 36:19327–19352, 2023.

[MSD+24] Xupeng Miao, Chunan Shi, Jiangfei Duan, Xiaoli Xi, Dahua Lin, Bin Cui, and Zhihao Jia. SpotServe: Serving Generative Large Language Models on Preemptible Instances. In *Proceedings of the ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 1112–1127, 2024.

[Nah04] Fiona Fui-Hoon Nah. A Study on Tolerable Waiting Time: How Long Are Web Users Willing to Wait? *Behaviour & Information Technology*, 23(3):153–163, 2004.

[NLZ+24] Xuefei Ning, Zinan Lin, Zixuan Zhou, Zifu Wang, Huazhong Yang, and Yu Wang. Skeleton-of-thought: Prompting LLMs for efficient parallel generation. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2024.

[NVI24] NVIDIA Corporation. TensorRT: High Performance Deep Learning Inference Optimizer and Runtime. Technical Report, NVIDIA, 2024. Available at: https://developer.nvidia.com/tensorrt Accessed: 2025-05-04.

[OAW+25] Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E Gonzalez, M Waleed Kadous, and Ion Stoica. RouteLLM: Learning to Route LLMs from Preference Data. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2025.

[PCZ+24] Pratyush Patel, Esha Choukse, Chaojie Zhang, Aashaka Shah, Íñigo Goiri, Saeed Maleki, and Ricardo Bianchini. Splitwise: Efficient Generative LLM Inference Using Phase Splitting. In *Proceedings of the Annual International Symposium on Computer Architecture (ISCA)*, pages 118–132. IEEE, 2024.

[PGL+21] David Patterson, Joseph Gonzalez, Quoc Le, Chen Liang, Lluis-Miquel Munguia, Daniel Rothchild, David So, Maud Texier, and Jeff Dean. Carbon Emissions and Large Neural Network Training. *arXiv:2104.10350*, 2021.

[Pow11] David Powers. Evaluation: From Precision, Recall and F-Measure to ROC, Informedness, Markedness & Correlation. *Journal of Machine Learning Technologies*, 2(1):37–63, 2011.

[PRWZ02] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. BLEU: A Method for Automatic Evaluation of Machine Translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 311–318, 2002.

[RBC+21] Jack W Rae, Sebastian Borgeaud, Trevor Cai, Katie Millican, Jordan Hoffmann, Francis Song, John Aslanides, Sarah Henderson, Roman Ring, Susannah Young, and et al. Scaling Language Models: Methods, Analysis & Insights from Training Gopher. *arXiv:2112.11446*, 2021.

[RG19] Nils Reimers and Iryna Gurevych. Sentence-BERT: Sentence Embeddings Using Siamese BERT-Networks. In Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan, editors, *Proceedings of the Conference on Empirical Methods in Natural Language Processing and the International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3982–3992, Hong Kong, China, 2019. Association for Computational Linguistics.

[RNSS18] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving Language Understanding by Generative Pre-Training. *OpenAI*, 2018.

[RSR+20] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020.

[RWC+19] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, and others. Language Models are Unsupervised Multitask Learners. *OpenAI blog*, 1(8):9, 2019. Available at: https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf Accessed: 2025-05-04.

[RZLL16] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. SQuAD: 100,000+ Questions for Machine Comprehension of Text. In *Proceedings of EMNLP*, pages 2383–2392, 2016.

[SBBC21] Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. WinoGrande: An Adversarial Winograd Schema Challenge at Scale. *Communications of the ACM*, 64(9):99–106, 2021.

[SBe98] Richard S Sutton, Andrew G Barto, and et al. *Reinforcement Learning: An Introduction*, volume 1. MIT Press Cambridge, 1998.

[SDCW19] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. DistilBERT: A Distilled Version of BERT: Smaller, Faster, Cheaper and Lighter. *arXiv:1910.01108*, 2019.

[SDSE20] Roy Schwartz, Jesse Dodge, Noah A Smith, and Oren Etzioni. Green ai. *Communications of the ACM*, 63(12):54–63, 2020.

[Sha19] Noam Shazeer. Fast Transformer Decoding: One Write-Head Is All You Need. *arXiv:1911.02150*, 2019.

[SMM+17] Noam Shazeer, *Azalia Mirhoseini, *Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2017.

[SPDF24] Nikhil Sardana, Jacob Portes, Sasha Doubov, and Jonathan Frankle. Beyond Chinchilla-Optimal: Accounting for Inference in Language Model Scaling Laws. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 43445–43460. PMLR, 2024.

[SPN+22] Shaden Smith, Mostofa Patwary, Brandon Norick, Patrick LeGresley, Samyam Rajbhandari, Jared Casper, Zhun Liu, Shrimai Prabhumoye, George Zerveas, Vijay Korthikanti, and et al. Using DeepSpeed and Megatron to Train Megatron-Turing NLG 530B, A Large-Scale Generative Language Model. *arXiv:2201.11990*, 2022.

[SRR+22] Aarohi Srivastava, Abhinav Rastogi, Abhishek Rao, Abu Awal Md Shoeb, Abubakar Abid, Adam Fisch, Adam R Brown, Adam Santoro, Aditya Gupta, Adrià Garriga-Alonso, and et al. Beyond the Imitation Game: Quantifying and Extrapolating the Capabilities of Language Models. *Transactions on Machine Learning Research*, 2022.

[SUV18] Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. Self-Attention with Relative Position Representations. In Marilyn Walker, Heng Ji, and Amanda Stent, editors, *Proceedings of the 2018 conference of the north American chapter of the association for computational linguistics: Human language technologies, volume 2 (short papers)*, pages 464–468, New Orleans, Louisiana, June 2018. Association for Computational Linguistics.

[SXH+24] Dimitris Stripelis, Zhaozhuo Xu, Zijian Hu, Alay Dilipbhai Shah, Han Jin, Yuhang Yao, Jipeng Zhang, Tong Zhang, Salman Avestimehr, and Chaoyang He. TensorOpera Router: A Multi-Model Router for Efficient LLM Inference. In Franck Dernoncourt, Daniel Preoţiuc-Pietro, and Anastasia Shimorina, editors, *Proceedings of EMNLP, Industry Track*, pages 452–462, Miami, Florida, US, 2024. Association for Computational Linguistics.

[TDFH+22] Romal Thoppilan, Daniel De Freitas, Jamie Hall, Noam Shazeer, Apoorv Kulshreshtha, Heng-Tze Cheng, Alicia Jin, Taylor Bos, Leslie Baker, Yu Du, and et al. LaMDA: Language Models for Dialog Applications. *arXiv:2201.08239*, 2022.

[TLI+23] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. LLaMA: Open and Efficient Foundation Language Models, 2023.

[TYL+23] Haotian Tang, Shang Yang, Ji Lin, Jiaming Tang, Wei-Ming Chen, Wei-Chen Wang, and Song Han. TinyChat: Large Language Model on the Edge, 2023. Available at: https://hanlab.mit.edu/blog/tinychat Accessed: 2025-05-04.

[UE25] OAR US EPA. Greenhouse Gas Equivalencies Calculator - Calculations and References, 2025. Available at: https://www.epa.gov/energy/greenhouse-gas-equivalencies-calculator-calculations-and-references Accessed: 2025-05-04.

[VSP+17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention Is All You Need. *Advances in Neural Information Processing Systems (NeurIPS)*, 30, 2017.

[WARH24] Carole Jean Wu, Bilge Acun, Ramya Raghavendra, and Kim Hazelwood. Beyond Efficiency: Scaling AI Sustainably. *IEEE Micro*, 44(5):37–46, 2024.

[WCTG23] Yiding Wang, Kai Chen, Haisheng Tan, and Kun Guo. Tabi: An Efficient Multi-Level Inference System for Large Language Models. In *Proceedings of the Eighteenth European Conference on Computer Systems*, pages 233–248, Rome Italy, 2023. ACM.

[WLC+25] Xinyuan Wang, Yanchi Liu, Wei Cheng, Xujiang Zhao, Zhengzhang Chen, Wenchao Yu, Yanjie Fu, and Haifeng Chen. MixLLM: Dynamic Routing in Mixed Large Language Models. In *Proceedings of NAACL-HLT*, pages 10912–10922, 2025.

[WRG+22] Carole-Jean Wu, Ramya Raghavendra, Udit Gupta, Bilge Acun, Newsha Ardalani, Kiwan Maeng, Gloria Chang, Fiona Aga, Jinshi Huang, Charles Bai, and et al. Sustainable AI: Environmental Implications, Challenges and Opportunities. *Proceedings of Machine Learning and Systems*, 4:795–813, 2022.

[WTB+22] Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, Ed H. Chi, Tatsunori Hashimoto, Oriol Vinyals, Percy Liang, Jeff Dean, and William Fedus. Emergent Abilities of Large Language Models. *Transactions on Machine Learning Research*, 2022.

[XLS+23] Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. SmoothQuant: Accurate and Efficient Post-Training Quantization for Large Language Models. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 38087–38099. PMLR, 2023.

[XSC24] Fangyuan Xu, Weijia Shi, and Eunsol Choi. RECOMP: Improving Retrieval-Augmented LMs with Context Compression and Selective Augmentation. In *Proceedings of International Conference on Learning Representations (ICLR)*, 2024.

[YJK+22] Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soojeong Kim, and Byung-Gon Chun. Orca: A Distributed Serving System for Transformer-Based Generative Models. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 521–538, 2022.

[ZBN+04] Liangzhao Zeng, Boualem Benatallah, Anne HH Ngu, Marlon Dumas, Jayant Kalagnanam, and Henry Chang. QoS-Aware Middleware for Web Services Composition. *IEEE Transactions on software engineering*, 30(5):311–327, 2004.

[ZCS+23] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, and et al. Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena. *Advances in Neural Information Processing Systems (NeurIPS)*, 36:46595–46623, 2023.

[ZHB+19] Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. HellaSwag: Can a Machine Really Finish Your Sentence? In Anna Korhonen, David Traum, and Lluís Màrquez, editors, *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 4791–4800, Florence, Italy, 2019.

[ZL17] Barret Zoph and Quoc Le. Neural Architecture Search With Reinforcement Learning. In *Proceedings of International Conference on Learning Representations (ICLR)*, 2017.

[ZNH+24] Zixuan Zhou, Xuefei Ning, Ke Hong, Tianyu Fu, Jiaming Xu, Shiyao Li, Yuming Lou, Luning Wang, Zhihang Yuan, Xiuhong Li, Shengen Yan, Guohao Dai, Xiao-Ping Zhang, Yuhan Dong, and Yu Wang. A Survey on Efficient Inference for Large Language Models. *CoRR*, abs/2404.14294, 2024.

[ZRG+22] Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, and et al. OPT: Open Pre-Trained Transformer Language Models. *arXiv:2205.01068*, 2022.

[ZWW+25] Richard Zhuang, Tianhao Wu, Zhaojin Wen, Andrew Li, Jiantao Jiao, and Kannan Ramchandran. EmbedLLM: Learning Compact Representations of Large Language Models. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2025.