GreenServ: Energy-Efficient Context-Aware Dynamic Routing for Multi-Model LLM Inference

Thomas Ziller
TU Wien
Vienna, Austria
thomas.ziller@tuwien.ac.at

Shashikant Ilager
University of Amsterdam
Amsterdam, The Netherlands
s.s.ilager@uva.nl

Alessandro Tundo
TU Wien
Vienna, Austria
alessandro.tundo@tuwien.ac.at

Ezio Bartocci
TU Wien
Vienna, Austria
ezio.bartocci@tuwien.ac.at

Leonardo Mariani
University of Milano-Bicocca
Milan, Italy
leonardo.mariani@unimib.it

Ivona Brandic
TU Wien
Vienna, Austria
ivona.brandic@tuwien.ac.at

Abstract

Large language models (LLMs) demonstrate remarkable capabilities, but their broad deployment is limited by significant computational resource demands, particularly energy consumption during inference. Static, one-model-fits-all inference strategies are often inefficient, as they do not exploit the diverse range of available models or adapt to varying query requirements.

This paper presents GreenServ, a dynamic, context-aware routing framework that optimizes the trade-off between inference accuracy and energy efficiency. GreenServ extracts lightweight contextual features from each query, including task type, semantic cluster, and text complexity, and routes queries to the most suitable model from a heterogeneous pool, based on observed accuracy and energy usage. We employ a multi-armed bandit approach to learn adaptive routing policies online. This approach operates under partial feedback, eliminates the need for extensive offline calibration, and streamlines the integration of new models into the inference pipeline.

We evaluated GreenServ across five benchmark tasks and a pool of 16 contemporary open-access LLMs. Experimental results show that GreenServ consistently outperforms static (single-model) and random baselines. In particular, compared to random routing, GreenServ achieved a 22% increase in accuracy while reducing cumulative energy consumption by 31%. Finally, we evaluated GreenServ with RouterBench, achieving an average accuracy of 71.7% with a peak accuracy of 75.7%. All artifacts are open-source and available as an anonymous repository for review purposes here: GitHub repository

Keywords

Computational Sustainability, Large Language Model, Inference Routing

1 Introduction

The rise of large language models (LLMs) is considered one of the major breakthroughs in machine learning, opening a new era of artificial intelligence (AI) with new capabilities such as human-like text and image generation. LLM-based autonomous agents have become an integral part of various applications and workflows, accelerating automation in areas such as customer service and market research [26]. However, training and the use of LLMs require a considerable amount of resources (e.g., energy) that raise several concerns about their computational sustainability [18].

Although approaches to reducing training costs for LLM have received a lot of attention, inference resource demands are often overlooked. In fact, the cumulative inference energy can exceed the training energy: a single ChatGPT query is estimated to consume approximately 2.9 Wh of energy for a total amount of 10 TeraWh per year [38]. Data centers hosting both training and inference already draw an appreciable share of global electricity, and their demand continues to grow [39].

Current LLM inference often relies on static *one-model-fits-all* strategies, routing all queries to the same large model regardless of complexity or quality [23]. While this dominates industry use, it wastes resources: studies show that many noncritical tasks (e.g., basic translation) can be handled by smaller, cheaper models with minimal quality loss [31, 55]. The open source ecosystem adds to the challenge by offering over 230,000 text generation models on platforms like HuggingFace [4], including fine-tuned variants and optimized architectures (quantized, distilled, etc.), creating opportunities but complicating the decision process.

In fact, selecting an optimal LLM is not trivial. First, non-expert users often lack the technical expertise or explicit criteria needed to evaluate trade-offs between accuracy, cost, and latency, leading many to default to larger models assuming better capabilities. Second, the landscape is highly dynamic: leaderboards such as the HuggingFace Open LLM[1] and the CRFM HELM[2] show that the top 50 models vary widely in size and specialization, and choices that are near-optimal today may become outdated within months. Third, performance is highly task-dependent; for example, smaller models can match or even outperform larger ones on focused tasks such as MMLU [35], but tend to underperform on broader challenges such as summarization [32].

To overcome the limitations of single model inference and improve the efficiency and performance of LLM inference, researchers have introduced two main computation paradigms: *model cascading* [28] and *routing-based inference* [49]. Cascading approaches attempt to reduce cost by initially using a small, lightweight model to handle the request while then iteratively selecting more capable models until the output does not meet predefined quality thresholds. Although this method can improve efficiency, it often involves

[1] https://huggingface.co/open-llm-leaderboard
[2] https://crfm.stanford.edu/helm/capabilities/latest/

multiple inferences per request by design, significantly increasing both latency and computational cost [23].

On the other hand, routing-based methods, such as RouteLLM [49], MixLLM [58], LLMBandit [45], and Eagle [62], aim to assign each inference request to the most appropriate model in a single step, based on a learned/heuristic decision process. Although these approaches show promising results, they are still affected by key limitations.

Limited continual learning. Most routers lack continual learning capabilities, operating statically after initial calibration, making them vulnerable to query distribution shifts and model degradation.

Reliance on proxy cost metrics. Quality-cost trade-offs often rely on synthetic proxies (API prices, token budgets) over actual production metrics like GPU or energy use, not directly measuring real resource consumption and limiting dynamic optimization.

Underutilization of new models due to calibration overhead. The availability of open-source model repositories with a diverse range of capabilities is constantly increasing, but many remain underexplored due to the calibration overhead associated with incorporating them. In fact, no approach supports zero-calibration integration, leading to underutilization of available model capabilities despite growing repository diversity.

To overcome these limitations, we propose GreenServ, a *dynamic, context-aware LLM inference routing framework* that assigns user queries to the most suitable model in its pool based on lightweight *contextual features* (i.e., task type, semantic context, and text complexity) extracted from incoming user queries and learned knowledge about models' performance (i.e., accuracy and energy consumption). GreenServ learns an *adaptive routing policy online* to select the most suitable model at runtime using a *contextual multi-armed bandit (MAB) algorithm* [42]. This enables online integration of new models without requiring extensive offline calibration.

Experimental results show that GreenServ consistently outperforms single-model and random baselines by achieving superior energy-accuracy operating points. Compared to random routing, GreenServ achieved accuracy gains of 22% while reducing cumulative energy consumption by 31%. Furthermore, results show that GreenServ operates consistently close to or beyond the static optimal solutions, indicating effective control of the precision-energy trade-off based on the configurable parameter λ. The ablation study of contextual characteristics shows a substantial impact of the task type characteristic, dropping the median cumulative regret to ≈ 400. This identifies the task type as the most informative component of the context to guide model selection in our setup. Moreover, the results confirm a successful adaptation of GreenServ to the introduction of models into the model pool by integrating new and better-performing models into its routing strategy. Finally, the overhead results show that the total average overhead per query ranges between 6.68 and 7.77 ms, when processed sequentially, which indicates a negligible overhead for GreenServ compared to the general inference time. Finally, when evaluated with RouterBench [37], GreenServ achieves an average AIQ and accuracy of 0.607 and 71.7% respectively, with a peak accuracy of 75.7%.

In summary, this work provides the following key contributions.

An adaptive context-aware LLM routing framework. We propose an LLM routing framework capable of effectively balancing the trade-off between accuracy and energy consumption while meeting latency requirements. By leveraging a MAB algorithm, it assigns user queries to the most suitable available model, and it is capable of integrating new and better performing models into its routing strategy without requiring expensive offline calibration.

A multi-feature query context representation. We propose a multi-feature query representation (i.e., task type, semantic context, and text complexity) as a structured context vector, and study the impact of both single and combined features via ablation.

Comprehensive baseline evaluation. We employ LinUCB for model selection and evaluate GreenServ's performance against multiple baselines including static routing strategies (random, smallest model, largest model, most accurate model) and alternative MAB approaches (ϵ-Greedy, Contextual Thompson Sampling) using 5 benchmark tasks and a pool of 16 open-access LLMs from HuggingFace. In addition, we also performed a trade-off analysis to understand how GreenServ behaves for different accuracy-energy-consumption ratio.

An extensive empirical evaluation. In addition to the ablation study for context characteristics, we performed specific experiments to study the adaptability to model addition, performed an overhead analysis, and evaluated GreenServ using RouterBench [37]. Finally, we analyzed the time and space complexity of our router agent.

The remainder of the paper is organized as follows. § 2 discusses and categorizes related work, identifying key research gaps. § 3 provides definitions and the necessary preliminary details for our contextual routing problem. § 3.2 formalizes the problem statements. § 4 presents GreenServ, including its system architecture and a thorough description of the proposed solution. § 5 outlines the implementation details. § 6 reports the empirical evaluation and results, discusses the main findings, and highlights current limitations. Finally, § 7 offers concluding remarks and directions for future work.

2 Related Work

Inference routing optimizes LLM inference by assigning specific models to handle different types of queries [49]. The process maps input requests to particular LLMs from a heterogeneous pool, which can vary in terms of parameter sizes, architectures, or optimization levels. This approach ensures that computational resources are allocated according to the characteristics of each query.

Static Routing Systems. Early LLM routing works used static, pre-deployment calibration strategies. For example, Tryage [34] used BERT embeddings for classification-based routing, achieving 50.9% accuracy. TABI [59] focused on complexity-based routing to reduce latency by 21-40%. RouterBench [37] introduced an evaluation framework, while Hybrid LLM [27] leveraged meta-learning to reduce large model calls by 40%. Characterized by pre-deployment calibration (PD) and fixed policies (FP), these methods lack adaptability to dynamic environments and evolving models.

Embedding Representations and Learning-based Routing Systems. Recent work on model routing focuses on advanced training methodologies. For example, RouteLLM [49] introduced preference data-based routing using matrix factorization, demonstrating a cost reduction of up to 70% while preserving performance. Smoothie [33] introduced label-free routing via embedding-based similarity comparison and latent variable graphical models, eliminating the need for labeled data. RouterDC [24] employs dual contrastive learning for query-based routing, and EmbedLLM [63] developed compact model representations using encoder-decoder architectures. Furthermore, GraphRouter [29] uses graph neural networks to model task-query-LLM interactions. These systems demonstrated improved routing accuracy through advanced representation learning. For new model additions, they lack adaptability for fast learning and the addition of models at runtime.

Dynamic and Adaptive Routing Systems. The most recent works have focused on dynamic adaptation and runtime learning capabilities. TensorOpera [56] employed K-nearest neighbors for efficient inference, while Universal Model Routing [40] introduced cluster-based representation for unseen test-time LLMs, achieving dynamic model integration. LLMBandit [45] proposed preference-conditioned dynamic routing using a multi-armed bandit formulation, enabling runtime preference specification and achieving significant cost reductions. MixLLM [58] presented the contextual-bandit-based routing, incorporating tag-enhanced embeddings and continual learning capabilities, achieving 97.25% of GPT-4's quality at 24.18% of the cost.

These systems represent significant progress in adaptive routing, moving from static configurations toward dynamic, learning-based approaches. Our approach extends this trajectory by combining minimal offline calibration with continuous online learning (hybrid calibration), enabling policy evolution through contextual bandits, supporting dynamic model integration without retraining, and optimizing multiple objectives using direct energy measurements rather than proxy metrics. We present a comprehensive routing framework capable of handling complex, evolving LLM ecosystems while maintaining near Pareto-optimal accuracy-energy trade-offs.

3 Preliminaries and Problem Formulation

In this section, we formalize the routing problem, which aims to dynamically assign incoming requests to a single model from a pool of heterogeneous LLMs while balancing two primary objectives: *accuracy* and *energy-consumption*. In the following subsections, we provide details about metrics and problem formulation.

3.1 Metrics

The following section specifies how we define and measure accuracy and efficiency, which are then condensed into a MOOP through scalarization (see § 3.2.1) to form the basis of our routing problem. 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.

3.1.1 Accuracy. We denote $Acc_m(q_t)$ 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) [52], ROUGE [46], or BLEU [50] are applicable. However, evaluating open-ended generation often requires subjective assessments (e.g., user feedback), which are difficult to automate and simulate reliably [22]. Therefore, for deterministic evaluation within the scope of this work, we focus exclusively on tasks where accuracy can be measured objectively against an available ground truth using such unambiguous metrics. We assume $Acc_m(q_t)$ is normalized to the interval $[0, 1]$, where higher values indicate a higher accuracy.

3.1.2 Energy Consumption. Efficiency in our setting refers to the effective utilization of computational resources to generate responses. It can be assessed through different aspects such as processing speed, memory footprint, or energy consumption. 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 $T_{\text{proc}(m,q_t)}$:

$$C_m(q_t) = \int_0^{T_{\text{proc}(m,q_t)}} P_m(\tau)\, d\tau \tag{1}$$

This formulation provides the mathematical foundation for our multi-objective optimization. In practice, we measure $C_m(q_t)$ directly via GPU power monitoring (§5).

3.2 Problem Formulation

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), among others. In Equation 2, when query q_t arrives, the system's routing policy π selects exactly one model m_t for inference.

$$m_t = \pi(q_t). \tag{2}$$

The policy π aims to balance two key objectives – accuracy and energy efficiency – based on measured historical performance of each model in related contexts.

3.2.1 Multi-Objective Optimization with Latency Constraints. Our routing problem balances accuracy and energy consumption as two competing objectives while respecting latency constraints that ensure acceptable Quality-of-Service (QoS) [61].

Inference latency represents the time from query submission to response. However, the latency components, such as data transfer and queuing delays, depend on specific deployment environments and operating conditions, and introduce strong deviations. Therefore, we model the controllable latency components: optimization overhead $L_{opt}(q_t)$ and inference processing time $L_m(q_t)$:

$$L_{\text{total}}(m, q_t) = L_{opt}(q_t) + L_m(q_t). \tag{3}$$

Users typically tolerate latency up to a threshold $L_{max,t}$, beyond which satisfaction declines sharply [48]. We therefore define the set of feasible models for query q_t as:

$$M_t^* = \{m \in M | L_m(q_t) \leq L_{max,t}\}. \tag{4}$$

A model exceeding $L_{max,t}$ is considered *infeasible* and is discarded in the candidate selection at time step t, analogous to established QoS techniques [61].

To handle the accuracy-energy trade-off, we apply the *Weighted Sum Method* [47]. For our routing problem, we combine accuracy and energy consumption for a query q_t as follows:

$$r_t(m, q_t) = \alpha Acc_m(q_t) - \beta C_m(q_t), \text{ subject to } m \in M_t^*. \tag{5}$$

where $\alpha = 1 - \lambda, \ \beta = \lambda, \ 0 \leq \lambda \leq 1$. This parameter λ conveniently allows interpolation between accuracy-only ($\lambda = 0$) and energy- only ($\lambda = 1$) policies. While this approach assumes a fixed rate of trade-off between objectives, it enables efficient, adaptive decision-making in practice. In our experiments, we perform a parameter sweep over λ to evaluate these trade-offs. However, solving Equation 5 online requires complete observations across models, hardware settings, and tasks; an assumption rarely feasible in practice. This necessitates a learning-based routing strategy under partial feedback. Consequently, in the next section, we describe how we tackled this challenge using a contextual multi-armed bandit framework.

3.2.2 Bandit Problem Formulation. In dynamic online settings, we need to learn optimal routing policies without exhaustive prior knowledge, i.e., we can only observe the performance of the selected model m_t on each query q_t as outcomes for unselected models remain unknown. This *partial feedback* structure [43] strongly motivates the application of Multi-Armed Bandit (MAB) algorithms. Each query q_t corresponds to a decision point, and each feasible model $m \in M_t^*$ represents an arm. After selecting m_t, the system observes the scalarized reward r_t based on accuracy and energy (Eq. 5). Moreover, we provide a key extension of classical MAB algorithms, with the inclusion of context features (described in § 4.2). Instead of treating all queries identically, *contextual bandits* leverage relationships between input characteristics and reward outcomes [44]. Utilizing context vector x_t extracted from q_t, we learn a policy $\pi : x_t \longrightarrow m_t$ 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 strategies. Over time, the policy continuously gains information and refines its estimates to approximate the true reward functions for x_t, allowing the system to adapt to new query distributions and handle integration of new models in online settings.

To evaluate the performance of a policy, we measure the performance gap using the concept of *regret* [43]. The oracle policy with complete knowledge of model performances would have selected the optimal model:

$$m_t^* = \arg\max_{m \in M_t^*} r_t(m, q_t), \tag{6}$$

where $r_t(m_t, q_t)$ is the reward function introduced in § 3.2.1 and M_t^* defines the set of feasible models at time step t. Thus, *instantaneous regret* is defined as:

$$\Delta_t = r_t(m_t^*) - r_t(m_t), \tag{7}$$

where m_t is the model selected by the routing policy. If the optimal model was chosen, regret is zero. After T iterations, cumulative regret is defined as:

$$\text{Regret}(T) = \sum_{t=1}^{T} \big(r_t(m_t^*) - r_t(m_t)\big). \tag{8}$$

The routing strategy aims to minimize Regret(T). Since MAB is optimizing a multi-objective trade-off between accuracy and energy efficiency, minimizing regret aligns with approximating context-specific Pareto fronts and discarding consistently dominated models.

4 GreenServ: Learning Energy-Efficient Context-Aware Dynamic Routing

In this section, we first define the system model along with its core components. Subsequently, we present the solution methodology for GreenServ.

4.1 System Model

Figure 1 illustrates the high-level view of the GreenServ's system model. GreenServ comprises three main components: (i) `Query Context Generator`, which extracts the necessary metadata to construct a query-specific context vector, capturing the unique characteristics of each query; (ii) `Router Agent Trainer`, where trains an agent using bandit learning to identify efficient routing configurations and (iii) `Online Deployment`, in which the trained router is deployed to handle inference requests in real time. It is important to note that GreenServ is capable of adapting to the addition of new models to the existing model pool.

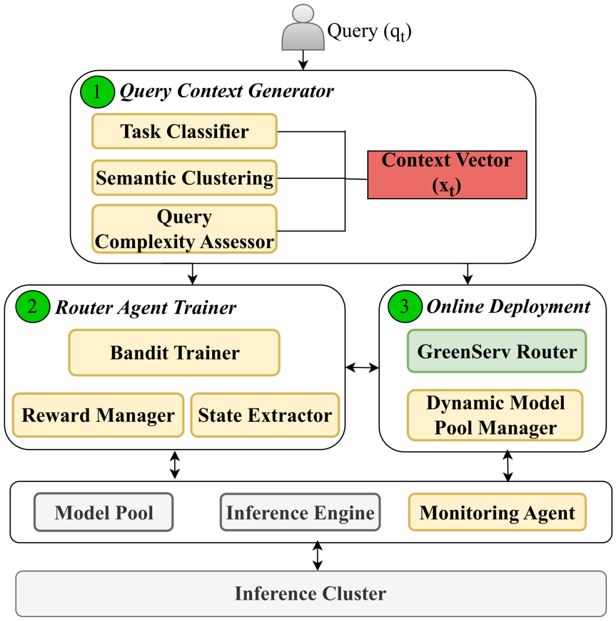


Figure 1: The system model of GreenServ.

4.2 Query Context Generator

The `Query Context Generator` processes each query through three components: *Task Classifier*, *Semantic Clustering*, and *Query Complexity Assessor*. These modules extract three features: *Task Type* (query intent [37]); *Cluster* (semantic context via embedding-based clustering); and *Complexity Score* (textual complexity). These are then combined into context vector x_t. By encapsulating multiple dimensions of the query, x_t allows the router to make informed decisions about model selection to effectively balance inference performance against energy efficiency.

4.2.1 Task Classifier. We rely on a lightweight text classification approach to identify high-level task types (e.g., summarization, QA). Specifically, we train a *Logistic Regression* (LR) model on top of semantic embeddings [20].

We extract the instruction text $q_{\text{instr},t}$ from the initial lines of the prompt q_t, following the common structure of instruction-based tasks. Its embedding $e_{\text{instr},t} = \text{embedding}(q_{\text{instr},t})$ is computed using a pre-trained transformer model [53], yielding a semantic vector representation.

An LR model is then trained using cross-entropy loss over labeled pairs $(e_{\text{instr},t}, l_t)$, where l_t denotes the ground-truth label [20]. The probability distribution is modeled as $p(l \mid e_{\text{instr},t}) = \sigma(We_{\text{instr},t} + b)$ with parameters W and b optimized to learn a decision boundary for classification.

For training data, we sample a small portion of our evaluation dataset described in §6.1.2. 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 [51]. After training, we store the LR parameters for further use. The resulting model is a simple classifier that extracts the discrete task label. We specifically choose LR for its computational simplicity, speed, and low resource footprint to avoid excessive offline overhead.

4.2.2 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 [53] as $e_{full,t} = embedding(q_t)$. Utilizing these full embeddings, we perform an online K-Means clustering algorithm [21] with a fixed number of clusters, K (specified in §6.1), 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:

$$c_t = \arg\max_c \frac{e_{\text{full},t} \cdot \mu_c}{\|e_{\text{full},t}\| \|\mu_c\|}. \tag{9}$$

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:

$$\mu_{c_t} \leftarrow \mu_{c_t} + \frac{1}{N_{c_t} + 1}(e_{\text{full},t} - \mu_{c_t}), \tag{10}$$

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 predetermined clusters. The number of clusters K is fixed (value specified in §6.1) 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.

4.2.3 Query Complexity Assessor. We calculate a single numeric score that represents the complexity of query q_t based on the Flesch Reading Ease formula [30]:

$$p(q_t) = 206.835 - 1.015 \cdot \left(\frac{\text{Words}_t}{\text{Sentences}_t}\right) - 84.6 \cdot \left(\frac{\text{Syllables}_t}{\text{Words}_t}\right). \tag{11}$$

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 (§6.1).

4.2.4 Context Vector. Our context vector encodes relevant characteristics of a query as $x_t = [l_t, c_t, p_t]$. The resulting vector is forwarded to the router (described in § 4.3), 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 [20]. Additionally, we add an intercept term (bias) by appending a constant value of 1, and 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 by non-exhaustive tuning described in § 6.1.5 and are intended to balance feature granularity with dimensionality. The specific number of task types (N_{tasks}) is determined by the evaluation datasets defined in § 6.1.2.

4.3 Router Agent Trainer

To handle unseen queries and out-of-domain distributions, we adopt a multi-armed bandit (MAB) approach to learn a policy for selecting models from the feasible set $m_t \in M_t^*$. We employ LinUCB [44], treating each model $m \in M$ as an arm and selecting model m_t based on context vector x_t.

State Extractor. GreenServ constructs the state using three parameters: accuracy $Acc_m(q_t)$, energy consumption $Em(q_t)$, and latency $L_m(q_t)$. Accuracy and energy are captured by interacting with the inference engine and monitoring agent. For latency, we use the predefined maximum output tokens (MaxNewTokens) for the query's task type as a conservative estimate.

Reward Manager. The MAB maximizes the scalarized reward (Equation (5)) parameterized by $\lambda \in [0, 1]$, where $\alpha = 1 - \lambda$ and $\beta = \lambda$. This balances accuracy ($\lambda = 0$) and energy efficiency ($\lambda = 1$). Performance is evaluated by cumulative regret over T steps:

$$R(T) = \sum_{t=1}^{T} [r_t(m_t^*, q_t, \lambda) - r_t(m_t, q_t, \lambda)], \tag{12}$$

where $m_t = \pi(x_t)$ is the chosen model and the optimal choice at time t is $m_t^* = \arg\max_{m \in M_t^*} r_t(m, q_t, \lambda)$.

Bandit Trainer. Algorithm 1 presents GreenServ's context-aware routing. It takes query stream $\{q_t\}_{t=1}^T$, model pool $\mathcal{M}$, and trade-off parameter λ as inputs. For each query, it extracts context vector $\mathbf{x}_t$, selects model m_t using LinUCB, executes inference, computes reward r_t (balancing accuracy and energy via λ), and updates MAB parameters for continuous online learning.

Algorithm 1 GreenServ: Context-Aware Routing for Multi-Model LLM Inference

```
Require: Query stream $\{q_t\}_{t=1}^T$, model pool $\mathcal{M}$, LinUCB algorithm $\mathcal{A}$, parameters $\lambda$
Ensure: Model selections $\{m_t\}_{t=1}^T$
Initialize LinUCB parameters $\{\mathbf{A}_m, \mathbf{b}_m\}$ for each $m \in \mathcal{M}$
Initialize task classifier $W$, cluster centroids $\boldsymbol{\mu}$
for $t = 1$ to $T$ do
    $\mathbf{x}_t \leftarrow$ GenerateContext($q_t$)    {Task, cluster, complexity}
    $m_t \leftarrow$ SelectModel($\mathbf{x}_t, \mathcal{M}_t^*, \mathcal{A}$)    {LinUCB routing}
    response $\leftarrow$ InferenceExecution($m_t, q_t$)
    accuracy, energy, latency $\leftarrow$ Monitor(response)    {Performance metrics}
    $r_t \leftarrow (1 - \lambda) \cdot$ accuracy $- \lambda \cdot$ energy
    UpdateMAB($\mathbf{A}_{m_t}, \mathbf{b}_{m_t}, \mathbf{x}_t, r_t$)
end for
return $\{m_t\}_{t=1}^T$
```

GreenServ employs LinUCB [44], a contextual bandit algorithm that assumes a linear relationship between context and reward, $\hat{r}_m(\mathbf{x}_t) = \boldsymbol{\theta}_m^T \mathbf{x}_t$, and maintains parameters $\mathbf{A}_m \in \mathbb{R}^{d \times d}$ and $\mathbf{b}_m \in \mathbb{R}^d$ for each model. Parameters are estimated as $\hat{\boldsymbol{\theta}}_m = \mathbf{A}_m^{-1}\mathbf{b}_m$ and updated as $\mathbf{A}_{m_t} \leftarrow \mathbf{A}_{m_t} + \mathbf{x}_t \mathbf{x}_t^T$ and $\mathbf{b}_{m_t} \leftarrow \mathbf{b}_{m_t} + r_t \mathbf{x}_t$ after observing rewards.

LinUCB employs systematic uncertainty quantification for exploration. It augments the expected reward with an exploration bonus proportional to parameter uncertainty:

$$m_t = \arg\max_{m \in \mathcal{M}_t^*} \left(\hat{\boldsymbol{\theta}}_m^T \mathbf{x}_t + \alpha \sqrt{\mathbf{x}_t^T \mathbf{A}_m^{-1} \mathbf{x}_t} \right), \tag{13}$$

where $\mathbf{x}_t^T \mathbf{A}_m^{-1} \mathbf{x}_t$ quantifies the reward estimate variance in context $\mathbf{x}_t$. The upper confidence bound approach ensures exploration targets regions of high uncertainty over uniform randomness. For baseline comparison, we also implement ϵ-Greedy [57], which employs random exploration with probability ϵ and greedy exploitation otherwise. Contextual Thompson Sampling [19] uses Bayesian posterior sampling over model parameters. Both baselines rely on the same linear reward model as LinUCB.

Complexity Analysis. Processing T queries, GreenServ incurs a time complexity of $O(T \cdot (l + |M|d^3))$, where l is the input text length, $|M|$ the number of candidate models, and d the feature vector dimension. Space complexity for LinUCB is $O(|M|d^2)$ to maintain parameter matrices for each arm. We provide a detailed analysis of time and space complexity in Appendix B.

4.4 Online Deployment

During online deployment, the GreenServ router processes the query q_t. It computes the context vector x_t for the given query and utilizes a trained Multi-Armed Bandit (MAB) agent to select a suitable model. Specifically, if the chosen model m_t is not already present in memory, the router interacts with the inference engine to load the model into GPU memory. Once the model is loaded, the inference engine generates a response for the query.

Note that multiple models can reside in memory simultaneously, and further optimizations may be applied to reduce the cost of model loading. However, these optimizations are beyond the scope of this work, as our primary focus is on model selection.

The system tracks energy consumption and latency from the beginning to the end of the inference process. In addition, it logs key metadata, including the number of input tokens and generated output tokens. Moreover, if a new model is added to the model pool, GreenServ can dynamically learn and adapt to the newly introduced model through online interaction with the agent trainer.

5 Implementation

We implement the GreenServ prototype in Python 3.10. User requests are handled via an HTTP API built with `FastAPI` [2], queued for inference over an asynchronous connection to `Redis` [12], and stored in a `PostgreSQL` [10] database, which holds prompts, model identifiers, results, and metrics. To isolate performance characteristics, we process each request independently (`batch_size = 1`) and assume all model weights are locally available at runtime.

Datasets are loaded through the HuggingFace `datasets` library [4], with random sampling conducted via Python's `random` package. For feature extraction, we compute transformer-based sentence embeddings using `sentence-transformers` [13] (specifically, the `all-MiniLM-L6-v2` model). Basic classification is performed using Logistic Regression [20], and similar queries are clustered online via K-Means [21], both implemented with `scikit-learn` [14]. Text complexity is measured using `textstat` [15] and the Flesch Reading Ease score [30], which we discretize via equal-width binning based on empirical score ranges.

We implement LinUCB [44] as the core routing algorithm for GreenServ, along with ϵ-Greedy [57] and CTS [19] as baseline alternatives for comparison. All strategies are implemented as custom Python classes, with internal operations performed on `NumPy` arrays [9]. LLMs are stored locally and loaded as HuggingFace-compatible models using the `transformers` [16] library in conjunction with `PyTorch` [11], using `bfloat16` precision [41] to reduce GPU memory usage and ensure efficient inference. All models undergo a single warm-up inference post-load to account for lazy initialization that might otherwise skew latency measurements.

Prior work often relies on proxy metrics (API costs, token budgets). We measure actual GPU power draw in watt-hours for direct optimization of actual resource consumption using the `zeus` library [17]. Inference latency is measured with Python's `time` module, excluding queueing, feature extraction, routing, and model loading. Accuracy is evaluated using Exact Match (EM) and ROUGE [46] via HuggingFace's `evaluate` library [5]. The code and associated artifacts are provided via an anonymous GitHub repository for review purposes: `GitHub repository`[3].

6 Empirical Evaluation

This section presents our empirical evaluation. In particular, we first detail our experimental setup in Section 6.1 describing the testbed, the evaluation datasets and the LLMs we used in our model pool, the evaluation metrics, how we tune the hyperparameters, and the baselines we used for comparison. Further, we outline our experimental plan in § 6.2 and we present empirical results in § 6.3. Finally, § 6.4 discusses the key findings, their implications, and the current limitations of the study.

6.1 Experimental Setup

6.1.1 Testbed. We run our experiments on a server running Ubuntu 22.04.5 LTS and equipped with 512 GB of RAM, an NVIDIA A100 GPU with 80 GB VRAM (CUDA 12.2), and an AMD EPYC 9354P processor with 32 cores. The GPU supports compute optimized bfloat16 acceleration [41].

6.1.2 Datasets. We selected five publicly available datasets encompassing a broad range of query types, complexity levels, and domains. For each dataset, 500 instances were uniformly sampled from the test set partition using a fixed random seed to ensure reproducibility. Specifically, we utilize the MMLU [35] dataset for question answering, HellaSwag [60] for situation completion, Winogrande [54] for commonsense reasoning, GSM8K [25] for mathematical reasoning, and CNN / Daily Mail [36] for summarization. Evaluation for MMLU, HellaSwag, Winogrande, and GSM8K is performed using the exact match metric, whereas the CNN / Daily Mail dataset is assessed with the ROUGE metric [46].

6.1.3 Model Pool. We compose a pool of 16 publicly available LLMs, representing a range of parameter scales and model families. Our model pool selection is guided by four criteria:

(1) Diversity in parameter counts, spanning from 0.5B to 34B, based on compatibility with our computational resources;
(2) Popularity of model families provided by leading vendors, namely Phi [7], Gemma [3], Mistral [8], Llama [6], Qwen [1];
(3) Availability of model weights for local deployment;
(4) Recency of publication to ensure state-of-the-art performance.

The final model pool consists of five Qwen 2.5 models (0.5B, 1.5B, 3B, 7B, 14B), Mistral v0.3 7B, four Gemma 3 models (1B, 4B, 12B, 27B), two Llama 3.1 models (1B and 8B) and Llama 3.2 3B, two Phi models (4-mini 4B and 4 14B), as well as Yi 34B. Appendix A.1 provides a summary table of the LLMs evaluated in our experiments, grouped by model family, along with their parameter counts and Hugging Face identifiers (HF Handles).

6.1.4 Evaluation Metrics. We report *mean normalized accuracy*, *total energy consumption (Wh)*, *model selection frequency*, and *cumulative* and *moving average regret* values in our experimental results. To estimate system overhead, we report *mean latency (ms)* and *model selection time.* Results include 95% confidence intervals, where appropriate, to account for the inherent variance in LLM outputs and the MAB learning processes.

The *normalized accuracy* is calculated through min-max normalization, which converts observed accuracy values to a range of $[0, 1]$. This allows a consistent comparison across the different evaluation metrics employed in the selected datasets.

$$\text{Normalized Accuracy} = \frac{Acc - Acc_{\min}}{Acc_{\max} - Acc_{\min}} \tag{14}$$

where $Acc_{\min}$ and $Acc_{\max}$ 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.

6.1.5 Hyperparameter Tuning. Prior to the main experiments, we conducted preliminary experiments to tune hyperparameters for LinUCB and baseline MAB algorithms, as well as feature extraction parameters (K_{cluster}, N_{bins}), evaluating their impact on cumulative regret. For LinUCB: $\alpha = 0.1$ and $\lambda_{\text{reg}} = 0.05$. For ϵ-Greedy: $\epsilon_0 = 1.0$, $\delta = 0.98$, and $\epsilon_{\min} = 0.01$. For CTS: $\sigma = 0.01$.

For feature extraction, we used $K = 3$ semantic clusters and $N_{\text{bins}} = 3$ text complexity bins, unless specified differently. After one-hot encoding and the addition of an intercept term as described in § 4, this yields a context vector dimension of $d = 12$ for the contextual bandit algorithms.

6.1.6 Baselines. We compare GreenServ against the following four baselines.

(1) *Random.* 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.
(2) *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.
(3) *Smallest (Qwen2.5-0.5B).* 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.
(4) *Highest Accuracy (Gemma-3-27B)*: This approach selects the model that achieves the highest possible average accuracy on benchmark tasks, without considering efficiency. It is derived through exhaustive profiling of all models in the model pool, and serves as the upper bound of achievable accuracy in a single-model setting.
(5) *ϵ-Greedy and Thompson Sampling*: These two serve as GreenServ variants with two different MAB strategies. Although GreenServ defaults to LinUCB, it allows extensibility to other bandit algorithms. ϵ-Greedy is a core exploration-exploitation method [57], with probability of ϵ, a random model is chosen (exploration), otherwise the model with the highest expected reward (exploitation). Thompson Sampling instead maintains a posterior over model performance [19].

[3] https://anonymous.4open.science/r/llm-inference-router-EBEA/README.md

It selects the model with the highest sampled reward from this distribution, balancing exploration and exploitation via *Bayesian inference.*

6.2 Experimental Plan

We study the effectiveness and efficiency of GreenServ through five experiments.

6.2.1 GreenServ vs. Baselines. This experiment studies the effectiveness of GreenServ by comparing it against baselines presented in § 6.1.6. We investigate learning behavior based on cumulative and moving-average regret, and evaluate the resulting regret, mean normalized accuracy, and total energy consumption.

6.2.2 Trade-off Analysis (λ Sweep). This experiment studies how GreenServ handles the trade-offs at different configurations by varying the λ parameter. We vary the λ parameter from 0 (accuracy only) to 1 (efficiency only) in increments of 0.1. For each value, we executed 20 runs for GreenServ and each baseline MAB algorithm with all contextual features activated.

6.2.3 Impact of Contextual Features. This experiment studies the impact of different contextual features on routing decisions, as not all features are expected to contribute equally to potential accuracy and efficiency gains. We run experiments with different feature configurations. In particular, 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 (i.e., task type, semantic cluster, text complexity) during the whole experiment run. Full-context routing leverages all features by using all derived query characteristics. We executed 50 runs for each configuration by employing LinUCB.

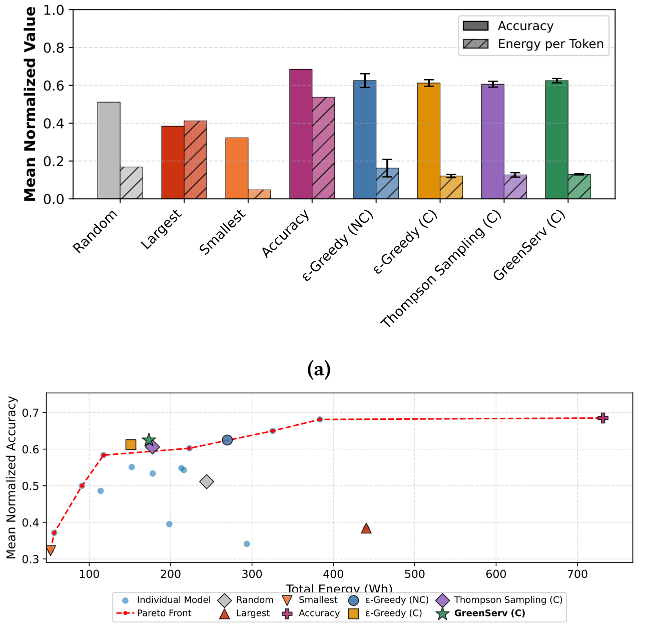
6.2.4 Adaptability: Model Addition. This experiment examines how GreenServ effectively adapts to changes in the model pool to simulate real-world scenarios of regular model releases. We introduce a new model (Gemma-3-12b), which showed high reward scores in a previous experiment after 1000 queries, and investigate if and how extensively the system incorporates it. In this experiment, we use LinUCB with full features and $\lambda = 0.2$ to favor the new high-accuracy model.

6.2.5 Overhead Analysis. This experiment evaluates the overhead the routing mechanism itself introduces to determine whether the benefits outweigh the costs. In particular, we account for the average latency introduced by each step involved in the feature extraction and the routing decision process per query.

For evaluation, we use our evaluation dataset consisting of 500 samples from each of the five benchmarks (i.e., total sequence length of $T = 2,500$ queries per experiment run). Unless otherwise specified, each experiment runs for the full sequence length of $T = 2,500$.

6.3 Results

6.3.1 GreenServ vs. Baselines. Figure 2a shows the mean normalized accuracy and energy consumption achieved by GreenServ ($\lambda = 0.4$) compared to baselines. The data are the results of 50 experiment runs and includes 95% confidence intervals. GreenServ and contextual baseline algorithms consistently achieve higher accuracy at lower energy consumption when compared to static and non-contextual baselines. In particular, GreenServ achieves an accuracy of ≈ 0.65 with significantly lower energy consumption (≈ 165 Wh). More specifically, GreenServ and contextual baselines outperform the non-contextual ϵ-Greedy by reaching higher accuracy (0.64-0.65 vs. 0.59), while reducing energy consumption by 29-38%. Compared to the static baselines, improvements become even more prevalent with GreenServ reducing energy consumption substantially compared to the random (31%), largest (64%), and accuracy (77%) baselines, while simultaneously achieving superior accuracy to the random (≈ 0.51), largest (≈ 0.39), and smallest (≈ 0.33) baselines. The confidence intervals for both accuracy and energy consumption largely overlap across GreenServ and the contextual baselines, indicating comparable performance and validating our selection of LinUCB based on its superior accuracy on external validation.



(a)

(b)

Figure 2: Comparison of mean normalized accuracy (higher is better) and total energy consumption. GreenServ and contextual baselines use all available features. Error bars represent 95% confidence intervals. The static Pareto frontier in Figure 2b is shown for reference.

Similarly, Figure 2b illustrates the trade-off of the same metrics but for a single run. The static Pareto front (red dashed line) is shown for reference. Predominantly, GreenServ (LinUCB) and the contextual baselines (Contextual ϵ-Greedy and Contextual Thompson Sampling) position themselves closely together in the more optimal top-left region. GreenServ and the contextual baselines 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.

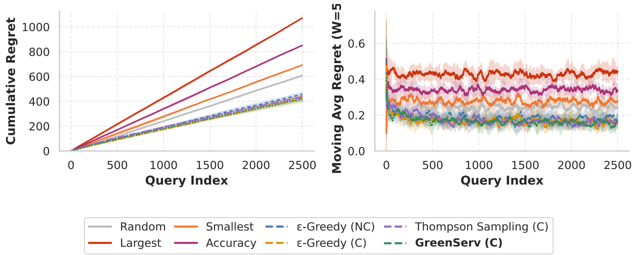


Figure 3: The left plot shows the cumulative regret over time for GreenServ and baseline MAB algorithms using the Full Features context. Shaded areas represent 95% confidence intervals. The right plot shows the moving average (window=50) regret over time for GreenServ and baseline MAB algorithms using the Full Features context, smoothing short-term fluctuations.

Figure 3 (left) illustrates the cumulative regret of GreenServ and the baseline MAB algorithms 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 GreenServ and the contextual baselines. GreenServ and the contextual baselines 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). GreenServ using LinUCB ($\approx$ 412) and Contextual Thompson Sampling ($\approx$ 400) achieve similar regret reductions.

Figure 3 (right) 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 GreenServ and the baseline algorithms. We can observe a high and erratic average regret during the first $\approx$ 50 steps of exploration. Following this phase, both non-contextual and contextual algorithms stabilize relatively quickly, with the non-contextual ϵ-Greedy stabilizing at a higher average regret level ($\approx$ 0.18 vs. $\approx$ 0.16). In contrast, GreenServ and the contextual baselines converge slower when compared to the non-contextual ones. The frequently crossing lines of GreenServ and the contextual baselines visible in Figure 3 (right) suggest that learned policies may be similar yet not identical. We report in Appendix A.2 additional results that illustrate these nuances in model selection patterns across GreenServ and the baseline algorithms.

6.3.2 Trade-off Analysis (λ Sweep). Figure 4 illustrates the trade-off between Mean Normalized Accuracy and Total Energy Consumption (Wh) across different λ values. As λ increases, MAB results follow the Pareto front from upper-right to lower-left. Remarkably, GreenServ and the contextual baselines consistently operate close to or beyond the static Pareto front (red dashed line). A detailed sensitivity analysis of λ, including absolute values and baseline comparisons, is presented in Appendix A.4.

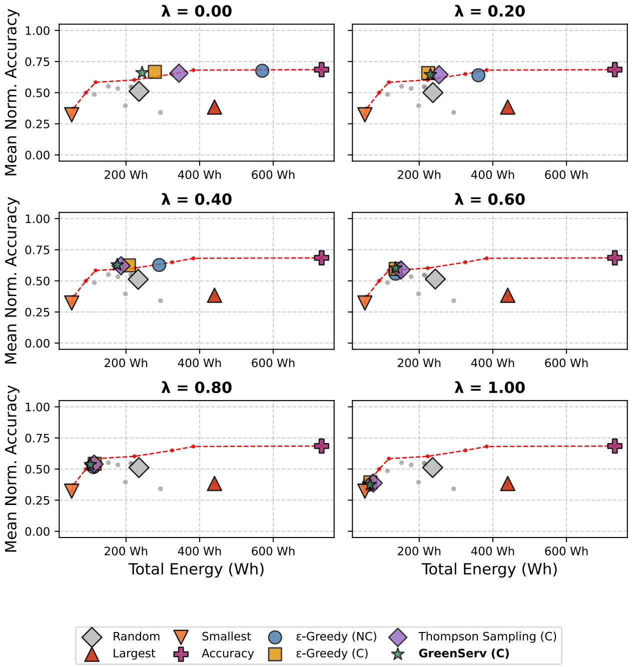


Figure 4: Mean Normalized Accuracy vs. Total Energy Consumption (Wh) for different strategies at varying λ values (0.0 to 1.0, increments of 0.2). The static Pareto front is shown for reference.

6.3.3 Impact of Contextual Features. Figure 5 presents the final cumulative regret distribution across 50 independent runs for each feature configuration: *None* (context-free baseline), *Task*, *Cluster*, *Complexity*, *Task + Cluster*, *Task + Complexity*, *Cluster + Complexity*, and *Full* (all features).

On average, Cluster reduces regret (-17) while Complexity increases it slightly (+7) compared to the implementation without features. However, the most substantial reduction in regret appears to be linked to the inclusion of the *Task* feature, dropping median cumulative regret to $\approx$ 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 informative feature interactions compared to the strong signal provided by the task type alone. We report in Appendix A.3 additional results that present how context influences model selection behavior by showing the selection frequency of each model.

6.3.4 Adaptability: Model Addition. Figure 6 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) correctly shows zero selection frequency as it was not yet part of the pool. Immediately after

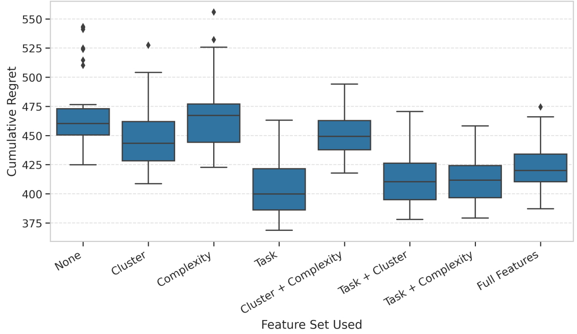


Figure 5: Distribution of final cumulative regret across multiple runs (n=50) for different contextual feature configurations at $\lambda = 0.4$.

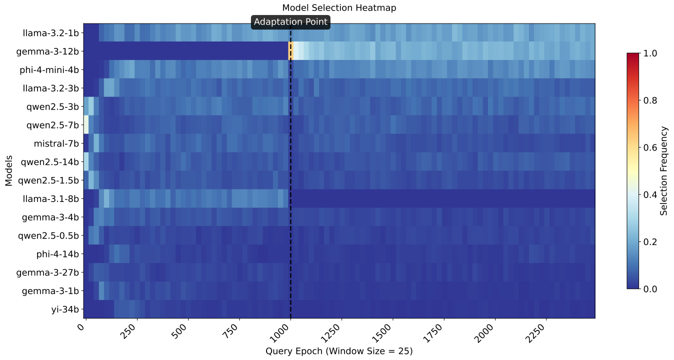


Figure 6: Mean selection frequency over time using GreenServ ($\lambda = 0.2$) for a single run. Gemma-3-12b is added at Query Step 1000 (black line).

its introduction, the algorithm begins to explore it, and its selection frequency rapidly increases. After around 100 queries, the selection frequency stabilizes at around 20%-25%. This shift in selection frequency visibly comes at the cost of previously frequently selected models (LLama-3.1-8b, Gemma-3-27b, Gemma-3-1b), which indicates the successful adaptation of the system to the pool of models by integrating the model into its routing strategy.

6.3.5 Overhead Analysis. The average overhead introduced by the system for each query consists of several distinct components. Task type classification requires 3.04 ms, semantic cluster identification takes 3.37 ms, and text complexity calculation adds 0.15 ms. For GreenServ, the LinUCB routing decision adds 0.86 ms. Summing these, the total pre-inference overhead per query is approximately 7.77 ms when processed sequentially.

When compared to the actual inference times of the evaluated models, this overhead is minor. For example, median inference latencies range from 36.1 ms for Llama-3.2-1B to 199.7 ms for Gemma-3-27B, with the relative overhead (assuming a 7.8 ms fixed cost) ranging from 21.6% for the fastest model down to just 3.9% for the slowest. We report in Appendix A.5 the detailed results for each of the models.

In summary, results indicate that the routing and feature extraction pipeline does not constitute a significant bottleneck relative to the overall inference time of state-of-the-art LLMs. In fact, the measured overhead is negligible for batch processing scenarios or applications where latency is not critical. However, in latency-sensitive deployments, further optimization of the feature extraction and routing steps may be warranted to minimize their impact.

6.3.6 External Benchmark Validation. We further validated our GreenServ using RouterBench [37] and evaluated on ≈ 36k queries spanning 9 tasks. Table 1 presents the results across three key metrics. GreenServ achieves the best peak and average accuracy at 75.7% and 71.7% respectively, which informed our algorithm selection. Contextual ϵ-Greedy achieves the highest AIQ of 0.637, where AIQ is RouterBench's primary metric for capturing the cost-performance trade-off frontier across different willingness-to-pay (WTP) parameters corresponding to GreenServ's λ.

Table 1: Performance of contextual routing algorithms on RouterBench. AIQ is averaged across all 9 tasks.

Algorithm	AIQ	Peak Acc.	Avg. Acc.
GreenServ	0.607	**75.7%**	**71.7%**
ϵ-Greedy	**0.637**	75.4%	69.2%
Thompson Sampling	0.624	69.5%	66.3%

Summary. Dynamic routing using contextual bandits consistently outperformed static and random baselines. At $\lambda = 0.4$, GreenServ exceeded the Pareto front, achieving accuracy-energy operating points unreachable by single-model deployments. Compared to baselines: Random (+22% accuracy, -31% energy), Smallest (+90-100% accuracy, +400% energy), and accuracy-optimized (-10-12% accuracy, -75% energy). External validation (RouterBench) showed GreenServ's superior accuracy. All contextual approaches achieved comparable performance, which confirms that feature engineering, model pool, and reward design are critical factors.

6.4 Discussion

Our study confirms that the accuracy–efficiency trade-off in LLM inference is significantly influenced by both inherent model characteristics (e.g., parameter count, architecture, training) and query-specific properties. As shown in the experimental results (see § 6.3), models demonstrate varying levels of accuracy and resource consumption depending on the task type, domain, and complexity of the input query. GreenServ addresses these trade-offs by inherently incorporating contextual information (i.e., query features) and adapting routing decisions based on the varying accuracy–efficiency performance across diverse queries and model repositories.

GreenServ includes the following limitations. *First*, MAB algorithms assume stationary rewards and adapt slowly to drifts. Periodic system calibration could address this. *Second*, our current evaluation focuses on tasks with objective ground truth (EM, ROUGE scores) to enable deterministic accuracy measurement. Many production LLM deployments involve structured tasks (classification, extraction, QA) where ground truth is available. Our framework can be extended to use alternative quality signals such as user feedback or LLM-as-judge evaluations [22] for open-ended generation tasks. *Third*, generalization across hardware requires latency profiles for various GPU architectures. *Fourth*, while we evaluated the impact of contextual characteristics, the sensitivity to specific features engineering choices, such as the number of clusters K or the number of complexity bins N, could be further explored. *Finally*, the empirical results obtained in this study are based on controlled environments for LLM deployments. In contrast, operational conditions should account for factors such as request concurrency, batch processing, queuing delays and runtime model switching.

7 Conclusions

We present GreenServ, a dynamic LLM inference routing framework that employs multi-armed bandits (MABs) to balance accuracy and energy consumption. It extracts a lightweight and multi-feature query context and leverages online MAB algorithms that adapt routing policies using partial feedback, eliminating the need for costly offline calibration. Formalizing routing as contextual multi-objective optimization with direct GPU energy measurements addresses the limitations of prior approaches that rely on proxy cost metrics.

Evaluation demonstrates superior performance over static baselines, achieving 22% higher accuracy and a 31% lower energy consumption under optimal configurations. Results validate the framework's ability to adapt policies to new models at runtime without requiring expensive offline recalibration.

Future work will extend the framework to support low-level hardware knob configurations, reducing the energy cost of LLM inference and scale our experiments to include larger models and multi-node cluster deployments.

References

[1] [n. d.]. Alibaba Cloud Qwen models. https://qwenlm.github.io/.

[2] [n. d.]. FastAPI. https://fastapi.tiangolo.com/.

[3] [n. d.]. Google Gemma models. https://ai.google.dev/gemma.

[4] [n. d.]. Huggingface. https://huggingface.co/docs/datasets/index.

[5] [n. d.]. HuggingFace evaluate documentation. https://huggingface.co/docs/evaluate.

[6] [n. d.]. Meta Llama models. https://www.llama.com/models/llama-3/.

[7] [n. d.]. Microsoft Phi models. https://azure.microsoft.com/en-us/products/ai-studio/phi.

[8] [n. d.]. Mistral AI models. https://mistral.ai/.

[9] [n. d.]. NumPy. https://numpy.org/.

[10] [n. d.]. PostgreSQL. https://www.postgresql.org.

[11] [n. d.]. PyTorch. https://pytorch.org/.

[12] [n. d.]. Redis. https://redis.io/.

[13] [n. d.]. SBERT. https://www.sbert.net/.

[14] [n. d.]. scikit-learn. https://scikit-learn.org/stable/.

[15] [n. d.]. textstat. https://pypi.org/project/textstat/.

[16] [n. d.]. Transformers documentation. https://huggingface.co/docs/transformers/index.

[17] [n. d.]. Zeus: Energy and Power Profiling for ML. https://github.com/ml-energy/zeus.

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

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

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

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

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

[23] Lingjiao Chen, Matei Zaharia, and James Zou. 2023. FrugalGPT: How to Use Large Language Models While Reducing Cost and Improving Performance. *arXiv:2305.05176* (2023). doi:10.48550/ARXIV.2305.05176

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

[25] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training Verifiers to Solve Math Word Problems. *arXiv:2110.14168* (2021). https://arxiv.org/abs/2110.14168

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

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

[28] David Dohan, Winnie Xu, Aitor Lewkowycz, Jacob Austin, David Bieber, Raphael Gontijo Lopes, Yuhuai Wu, Henryk Michalewski, Rif A. Saurous, Jascha Sohl-dickstein, Kevin Murphy, and Charles Sutton. 2022. Language Model Cascades. arXiv:2207.10342 [cs.CL] https://arxiv.org/abs/2207.10342

[29] Tao Feng, Yanzhen Shen, and Jiaxuan You. 2025. GraphRouter: A Graph-Based Router for LLM Selections. In *Proceedings of the International Conference on Learning Representations (ICLR)*. https://openreview.net/forum?id=eU39PDsZtT

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

[31] Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. 2023. OPTQ: Accurate Quantization for Generative Pre-Trained Transformers. In *Proceedings of International Conference on Learning Representations (ICLR)*. https://openreview.net/forum?id=tcbBPnfwxS

[32] Xue-Yong Fu, Md Tahmid Rahman Laskar, Elena Khasanova, Cheng Chen, and Shashi Tn. 2024. Tiny Titans: Can Smaller Large Language Models Punch Above Their Weight in the Real World for Meeting Summarization?. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 6: Industry Track)*. Association for Computational Linguistics, Mexico City, Mexico, 387–394. doi:10.18653/v1/2024.naacl-industry.33

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

[34] Surya Narayanan Hari and Matt Thomson. 2023. Tryage: Real-time, intelligent Routing of User Prompts to Large Language Models. *arXiv:2308.11601* (2023). https://arxiv.org/abs/2308.11601

[35] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2021. Measuring Massive Multitask Language Understanding. In *Proceedings of International Conference on Learning Representations (ICLR)*. https://openreview.net/forum?id=d7KBjmI3GmQ

[36] Karl Moritz Hermann, Tomas Kocisky, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. 2015. Teaching Machines to Read and Comprehend. In *Advances in Neural Information Processing Systems (NeurIPS)*, Vol. 28.

[37] Qitian Jason Hu, Jacob Bieker, Xiuyu Li, Nan Jiang, Benjamin Keigwin, Gaurav Ranganath, Kurt Keutzer, and Shriyash Kaustubh Upadhyay. 2024. RouterBench: A Benchmark for Multi-LLM Routing System. In *Agentic Markets Workshop at ICML*. https://openreview.net/forum?id=IVXmV8Uxwh

[38] International Energy Agency. 2024. Electricity 2024: Analysis and Forecast to 2026. https://iea.blob.core.windows.net/assets/18f3ed24-4b26-4c83-a3d2-8a1be51c8cc8/Electricity2024-Analysisandforecastto2026.pdf.

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

[40] Wittawat Jitkrittum, Harikrishna Narasimhan, Ankit Singh Rawat, Jeevesh Juneja, Zifeng Wang, Chen-Yu Lee, Pradeep Shenoy, Rina Panigrahy, Aditya Krishna Menon, and Sanjiv Kumar. 2025. Universal LLM Routing with Correctness-Based Representation. In *First Workshop on Scalable Optimization for Efficient and Adaptive Foundation Models*. https://openreview.net/forum?id=QpOCijgaBE

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

[42] John Langford and Tong Zhang. 2007. The Epoch-Greedy Algorithm for Multi-armed Bandits with Side Information. In *Advances in Neural Information Processing Systems 20, Proceedings of the 21st Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, Dec 3-6, 2007*, John C. Platt, Daphne Koller, Yoram Singer, and Sam T. Roweis (Eds.). Curran Associates, Inc. https://proceedings.neurips.cc/paper/2007/hash/4b04a686b0ad13dce35fa99fa4161c65-Abstract.html

[43] Tor Lattimore and Csaba Szepesvári. 2020. *Bandit algorithms.* Cambridge University Press.

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

[45] Yang Li. 2025. LLM Bandit: Cost-Efficient LLM Generation via Preference-Conditioned Dynamic Routing. *arXiv:2502.02743* (2025). https://arxiv.org/abs/2502.02743

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

[47] R.T. Marler and J.S. Arora. 2004. Survey of Multi-Objective Optimization Methods for Engineering. *Structural and Multidisciplinary Optimization* 26, 6 (2004), 369–395. doi:10.1007/s00158-003-0368-6

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

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

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

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

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

[53] Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence Embeddings Using Siamese BERT-Networks. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing and the International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan (Eds.). Association for Computational Linguistics, Hong Kong, China. doi:10.18653/v1/D19-1410

[54] Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2021. WinoGrande: An Adversarial Winograd Schema Challenge at Scale. *CACM* 64, 9 (2021), 99–106. doi:10.1145/3474381

[55] Aarohi Srivastava, Abhinav Rastogi, Abhishek Rao, Abu Awal Md Shoeb, Abubakar Abid, Adam Fisch, Adam R Brown, Adam Santoro, Aditya Gupta, Adrià Garriga-Alonso, et al. 2022. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models. *arXiv preprint arXiv:2206.04615* (2022).

[56] Dimitris Stripelis, Zhaozhuo Xu, Zijian Hu, Alay Dilipbhai Shah, Han Jin, Yuhang Yao, Jipeng Zhang, Tong Zhang, Salman Avestimehr, and Chaoyang He. 2024. TensorOpera Router: A Multi-Model Router for Efficient LLM Inference. In *Proceedings of EMNLP, Industry Track*, Franck Dernoncourt, Daniel Preoţiuc-Pietro, and Anastasia Shimorina (Eds.). Association for Computational Linguistics, Miami, Florida, US, 452–462. doi:10.18653/v1/2024.emnlp-industry.34

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

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

[59] Yiding Wang, Kai Chen, Haisheng Tan, and Kun Guo. 2023. Tabi: An Efficient Multi-Level Inference System for Large Language Models. In *Proceedings of the Eighteenth European Conference on Computer Systems*. ACM, Rome Italy, 233–248. doi:10.1145/3552326.3587438

[60] Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830* (2019).

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

[62] Zesen Zhao, Shuowei Jin, and Z. Morley Mao. 2024. Eagle: Efficient Training-Free Router for Multi-LLM Inference. arXiv:2409.15518 [cs.LG] https://arxiv.org/abs/2409.15518

[63] Richard Zhuang, Tianhao Wu, Zhaojin Wen, Andrew Li, Jiantao Jiao, and Kannan Ramchandran. 2025. EmbedLLM: Learning Compact Representations of Large Language Models. In *Proceedings of the International Conference on Learning Representations (ICLR)*. https://openreview.net/forum?id=Fs9EabmQrJ

A Appendix: Experiment Details and Results

A.1 Model Pool

Table 2: Model Pool

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

Table 2 lists the LLMs used in the experiments, grouped by model family, along with their parameter counts and Hugging Face identifiers (HF Handles).

A.2 Model Selection Patterns

Figure 7 illustrates model selection patterns across the MAB algorithms. Models like Llama-3.1-8B and Phi-4-Mini-4B show high selection frequency across algorithms. Contextual algorithms (C) exhibit more distributed patterns than non-contextual (NC) variants, indicating finer-grained performance distinctions, particularly evident in middle-tier models. This suggests algorithms identify

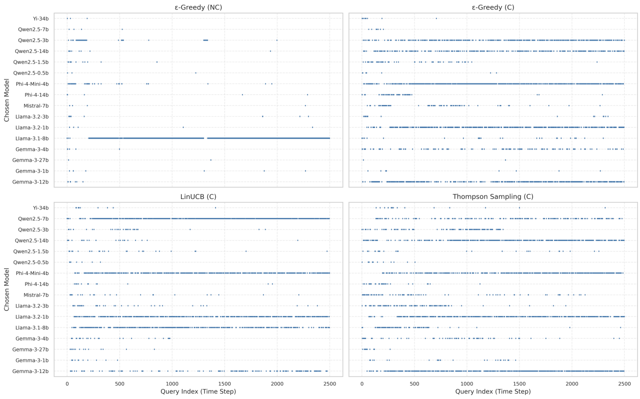


Figure 7: Sequence of models chosen by the MAB algorithms during a single run (λ = 0.4).

niches where certain models excel despite not being globally optimal.

A.3 Contextual Features

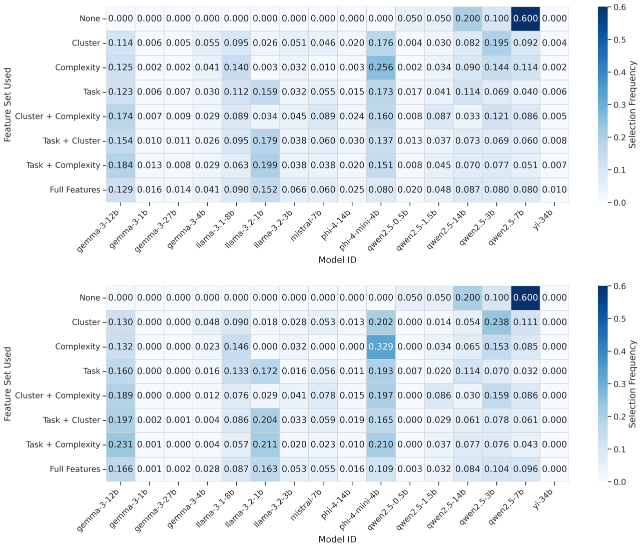


Figure 8: The heatmaps show the selection frequency of each model for different feature configurations across runs (n=10). The top heatmap shows selections during the first half of the experiments (1-1250), while the bottom shows the second half (1251-2500). Darker blue indicates higher selection frequency. 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.

Figure 8 shows how context influences model selection frequency. The top heatmap displays patterns for the first half (1-1250), while the bottom shows the second half (T=1251 to T=2500), averaged over ten runs. Initially, selections distribute widely as algorithms explore. Adding features increases exploration, further distributing loads across models. In contrast, the second half shows more concentrated selection patterns as policies stabilize. The baseline without features consistently favors Qwen2.5-7B ($\approx$ 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 ($\approx$ 0.17) and Phi-4-mini-4B ($\approx$ 0.19) among others. The *Full Features* configuration demonstrates the most spread-out policies as it tries to match query requirements precisely.

A.4 Trade-off Analysis (λ Sweep)

Figure 9 presents the distribution of mean normalized accuracy and total energy consumption across 20 runs for GreenServ and baseline algorithms as λ changes in between 0 and 1. Both accuracy and energy consumption decrease as λ increases, which demonstrates the system's ability to prioritize either objective when instructed. GreenServ and the contextual baselines show similar trends, maintaining slightly higher accuracy, lower energy consumption and greater robustness compared to the non-contextual ϵ-Greedy across most λ values.

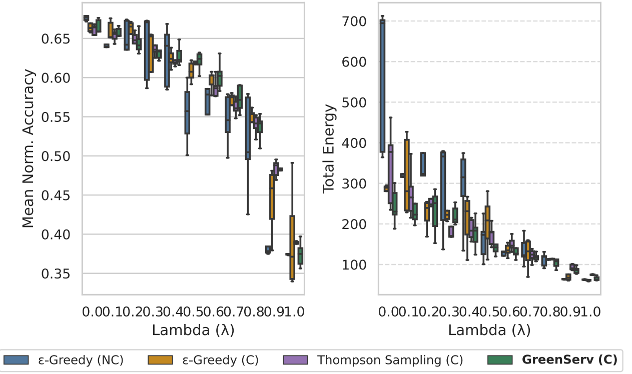


Figure 9: Distribution of Mean Normalized Accuracy (top) and Total Energy Consumption (bottom) for GreenServ and baseline MAB strategies across λ values.

Figure 4 provides another perspective on the trade-off by showing our strategy aggregates on accuracy-energy for different λ values (in increments of 0.2 for clarity). Each point represents the average accuracy-energy outcome of an algorithm at a specific λ value. As λ increases, MAB results follow the Pareto front from upper-right to lower-left. Remarkably, GreenServ and the contextual baselines consistently operate close to or beyond the static Pareto front (red dashed line).

A.5 Overhead Analysis

Table 4 lists the average elapsed time (ms) for each step involved in the feature extraction and routing decision process for a single query. 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

Table 3: Model Inference Latency Statistics

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

inference times, which varied significantly across our model pool, as detailed in Table 3.

Table 4: Average Overhead per Component

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**

B Appendix: Complexity Analysis

The computational complexity of GreenServ stems mainly from feature extraction and model selection during routing. We analyze how the system scales with key parameters: the number of queries T, model pool size $|M|$, context vector dimension d, and average query length l.

B.1 Time Complexity

For each incoming query, we perform feature extraction followed by routing. In the demonstrated implementation, feature extraction involves computing two transformer embeddings using all-MiniLM-L6-v2, which has a fixed maximum sequence length of 256 tokens. Any input exceeding this limit is truncated, bounding the self-attention computation to $O(256^2) = O(1)$ time regardless of query length. The Flesch Reading Ease calculation adds an $O(l)$ pass through the full text. Since the remaining operations (task classification, cluster assignment and one-hot encoding) require constant time, feature extraction totals $O(l)$ per query, though in practice the constant-time transformer operations dominate.

The routing complexity depends on the chosen algorithm. All variants first check feasibility constraints for each model in $O(|M|)$ time. Non-contextual ϵ-Greedy then requires at most $O(|M|)$ comparisons to find the best model. Contextual algorithms cause higher costs: contextual ϵ-Greedy computes $|M|$ dot products of dimension d, resulting in $O(|M|d)$ complexity. LinUCB and Thompson Sampling must invert $d \times d$ matrices for each model, resulting in $O(|M|d^3)$ complexity which is dominated by the matrix operations. Processing all T queries sequentially results in a time complexity $O(T \cdot (l + |M|d^3))$. With our experimental parameters ($|M| = 16$, $d = 12$), constant-time transformer embedding dominates feature extraction at approximately 6-7 milliseconds per query, and routing adds 0.02-1.21 milliseconds depending on the algorithm.

B.2 Space Complexity

Space complexity remains independent of T as the system maintains only derived statistics. Feature extraction requires $O(K \cdot d\text{emb})$ for semantic cluster centroids and $O(n\text{tasks} \cdot d_{\text{emb}})$ for classifier weights. MAB algorithms vary in memory usage. Non-contextual ϵ-Greedy requires $O(|M|)$ and its contextual variants $O(|M| \cdot d)$, while LinUCB and Thompson Sampling require $O(|M| \cdot d^2)$ to store the matrices and vectors per model. With $|M| = 16$, $d = 12$, and $K = 3$, total memory usage remains negligible compared to employed language model weights.

B.3 Practical Implications

Feature extraction is dominated by constant-time transformer embeddings, while Flesch complexity scales linearly but contributes minimally. For routing, the number of models $|M|$ and context vector dimension d affect execution time. The cubic scaling in d for contextual bandits appears concerning but remains manageable with $d = 12$ when applying modern libraries that optimize these matrix operations. Combined, our framework achieves linear time scaling with respect to the primary input size T while maintaining constant space complexity. The combination of predictable per-query cost and fixed memory usage makes the system suitable for long-running deployments processing millions of queries.