QUICKTOPIA: Iteration-Level GPU Frequency Control for Energy–Latency Co-Optimization in LLM Inference

Soyang Baek[1,2*], Bodon Jeong[1*], Hongsu Byun[1], Sungyong Park[1†]
[1]*Sogang University*, Seoul, Republic of Korea [2]*LG Electronics Inc*
{bd91jeong, byhs, parksy}@sogang.ac.kr, soyang.baek@lge.com

***Abstract*—The expansion of LLM inference into resource-constrained environments has made energy efficiency a critical bottleneck for operational sustainability. While existing GPU DVFS approaches attempt to optimize either latency or energy, they often suffer from suboptimal global efficiency due to their inability to account for the dynamic fluctuations in computational intensity across iterations. This results in an unstable balance where focusing solely on a single metric fails to achieve meaningful operational efficiency in terms of the Energy-Delay Product (EDP). To address this, we propose QUICKTOPIA, an iteration-level GPU frequency control framework that identifies the optimal balance between energy and latency in real-time. QUICKTOPIA dynamically adapts to varying computational scales by leveraging iteration-level execution characteristics without requiring model modifications or offline profiling. Furthermore, it employs a Pareto-based decision mechanism and a tree-structured segment manager to minimize control overhead while ensuring stable policy convergence even in noisy environments. Our evaluation shows that QUICKTOPIA achieves the lowest EDP across diverse workloads, reducing EDP by up to 22% compared to state-of-the-art methods while maintaining latency increases within a marginal 1–5% range. This demonstrates that QUICKTOPIA is a practical and effective solution for optimizing LLM serving efficiency on single-GPU platforms.**

***Index Terms*—LLM Inference System, Energy Efficiency, GPU DVFS, Pareto Optimality**

I. INTRODUCTION

While the initial success of Large Language Models (LLMs) was driven by hyperscale models relying on massive GPU clusters, recent deployment environments are rapidly expanding to resource-constrained settings, such as single-GPU systems, edge devices, on-premises servers, and embedded AI platforms [1]–[3]. In particular, as demand for practical applications such as internal knowledge retrieval, automated customer support, and code generation assistance continues to surge, the need to efficiently run LLMs on such small-scale infrastructure has become more critical than ever. This democratization signifies a paradigm shift, where LLM inference evolves beyond a specialized data-center technology into a universal computing platform across diverse infrastructures.

However, alongside the widespread adoption of LLM serving, massive power consumption has emerged as a critical bottleneck that exacerbates operational costs [4]. The rapid power variability of GPUs accumulates costs linearly during continuous operation, imposing a non-negligible economic burden even on single-GPU systems [5]. Consequently, in modern inference environments, it is essential to adopt management strategies that reconcile the trade-off between performance and energy cost by maximizing energy efficiency while maintaining high responsiveness [4], [6]–[8].

GPU DVFS in LLM inference. In LLM inference, Dynamic Voltage and Frequency Scaling (DVFS) is a key software-based method for enhancing efficiency without hardware modifications [8]. By dynamically adjusting GPU frequencies at runtime, DVFS provides a practical mechanism to manage the trade-off between performance and power consumption. Empirical studies confirm that latency and energy vary significantly with DVFS settings, establishing frequency control as a critical lever for optimizing LLM serving efficiency [4]–[9].

State-of-the-art in GPU DVFS. In this context, throttLL'eM [6] is a particularly prominent recent work that leverages GPU DVFS for LLM serving. By dynamically adjusting GPU frequency based on request latency, it consumes only as much energy as needed to satisfy explicit service level objectives (SLOs), establishing itself as the current state-of-the-art in DVFS-based LLM serving research.

Energy-Delay Product for LLM inference. However, evaluating LLM inference efficiency in real-world deployments using a single metric, such as latency or energy consumption, is insufficient. Minimizing latency may increase power consumption, while focusing solely on energy reduction can degrade user-perceived performance [10], [11]. To capture both aspects, this paper adopts the Energy-Delay Product (EDP) [12], defined as the product of latency and energy consumption, as the primary metric. EDP captures the trade-off between performance and energy, making it suitable for assessing overall system efficiency in practical environments.

Motivation. throttLL'eM [6], a state-of-the-art study based on GPU DVFS, reduces energy consumption by lowering the frequency while satisfying SLOs. However, this approach exhibits two critical limitations from the perspective of global efficiency. First, in continuous batching environments, computational characteristics change dynamically at every LLM iteration. Consequently, the minimum frequency that satisfies the SLO is not guaranteed to be the optimal choice for

*These authors are first co-authors and have contributed equally.
†S. Park is the corresponding author.

2993-2114/26/$31.00 ©2026 IEEE
DOI 10.1109/CCGrid68966.2026.00014

maximizing energy efficiency. Second, it treats latency merely as a hard constraint without considering EDP optimization. By focusing solely on power reduction within the SLO bound while neglecting the balance with latency, it fails to achieve practical operational efficiency in terms of EDP (§III-A).

Modern LLM serving engines adopt continuous batching to maximize resource utilization by dynamically batching heterogeneous requests. This leads to an irregular combination of query counts and computational phases (prefill and decode) within each batch. As a result, the aggregate GPU workload fluctuates rapidly on an iteration-by-iteration basis. Therefore, maximizing overall operational efficiency requires capturing this rapidly varying workload in real time and dynamically determining the GPU frequency that achieves an optimal balance between latency and energy (§III-B).

Key Design and Contributions. This paper proposes QUICKTOPIA, a GPU DVFS control framework that dynamically identifies the optimal balance between energy and latency in real-time by leveraging the iteration-level execution characteristics of LLM inference.

(1) Iteration-Level Frequency Selection. QUICKTOPIA identifies the rapidly fluctuating computational intensity at every iteration within continuous batching environments in real-time. Building on this, it precisely and continuously adapts to variations in the execution flow by dynamically determining and applying the frequency that corresponds to the optimal balance between latency and energy for each computational scale.

(2) Pareto-based Frequency Decision. To determine the optimal frequency aligned with the identified LLM computational workload, QUICKTOPIA adopts Pareto optimality as a core criterion. By quantitatively evaluating the trade-off between two conflicting objective functions, latency and energy, it selects the frequency that achieves the optimal balance between these metrics. Consequently, QUICKTOPIA realizes multi-objective optimization without being biased toward a specific metric, which ultimately leads to improvements in the EDP.

(3) Per-Iteration DVFS Control. QUICKTOPIA is designed to ensure extremely rapid frequency exploration and decision-making, accommodating the short cycles of iteration-level control. To this end, it dynamically manages Pareto data structures to minimize the learning and search overhead for the Pareto frontier corresponding to each computational scale, without introducing noticeable overhead on inference execution.

In this study, we implemented QUICKTOPIA and evaluated its effectiveness by integrating it with vLLM [13], a state-of-the-art LLM inference framework. Experimental results demonstrate that QUICKTOPIA reduces the EDP by up to 22% compared to throttLL'eM. Furthermore, we provide an in-depth evaluation of our Pareto-based design, showing that iterative learning progressively refines the Pareto frontier and that noise reduction is essential for stable optimization.

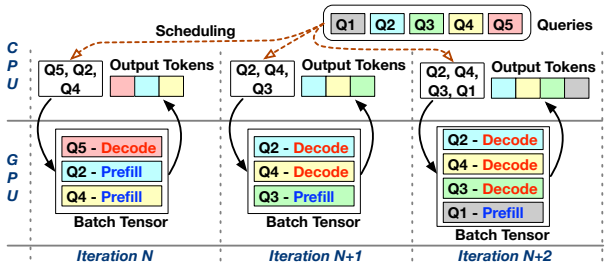


Fig. 1: Iterative inference sequence in LLM systems [13].

II. BACKGROUND

A. LLM Inference System: Serving Framework and Metrics

Iterative Process. Fig.1 illustrates the iteration sequence of a modern LLM inference system [13], [14]. For clarity, this section describes the mechanism based on vLLM [13], a representative framework. At the beginning of each iteration, the inference engine schedules multiple pending queries into a single batch per iteration and dispatches it to the GPU. The GPU performs a single forward pass on the batch, and the generated output tokens are returned to the CPU. The inference engine then updates the status of each request and immediately prepares the next batch for the following iteration. Throughout this process, the number of queries scheduled per iteration and the computational characteristics of the batch (i.e., whether in the prefill or decode phase) change dynamically. As a result, the execution time of each iteration naturally varies.

From the perspective of GPU kernels, these heterogeneous individual requests are abstracted as a unified batch tensor. In other words, the GPU hardware does not recognize or process them as a collection of separate workloads, but rather handles them collectively as a single monolithic computational block.

Metrics. When evaluating the efficiency of LLM inference systems, focusing on only one aspect such as latency or energy should be avoided. Reducing latency alone often results in excessive power consumption, while optimizing solely for energy can lead to unacceptable performance degradation. To capture this trade-off in a unified manner, we adopt the *Energy-Delay Product (EDP)* [12] as a primary evaluation metric. EDP is defined as the product of execution time and energy consumption (EDP $=$ Delay $\times$ Energy). A lower EDP value indicates that the system achieves high performance while maintaining energy efficiency.

In addition to EDP, we define two iteration-level metrics for analyzing the token generation process. *Iteration Time* is the actual time required to complete a single inference loop. In interactive services that stream tokens this value directly determines the *Time between Tokens (TBT)* perceived by users and thus represents service responsiveness. *Joules per Token (JPT)* denotes the total energy consumed to generate one token. It is computed as the product of the average power consumption in watts and the iteration time for that iteration and serves as an energy efficiency metric that is directly related to the operational cost of the service.

B. Pareto Optimality in Multi-Objective Optimization

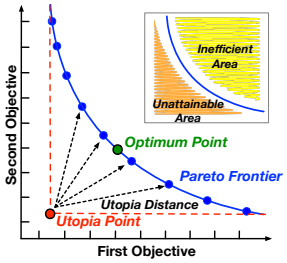


Fig. 2: Pareto optimality.

Multi-objective optimization problems may not have a single solution that simultaneously satisfies all objectives because the objectives often conflict with each other. In such cases, improving one objective usually requires sacrificing another, and the concept of *Pareto optimality* is used to define reasonable optimal solutions under this trade-off. Fig. 2 illustrates Pareto optimality in a bi-objective optimization problem, where both objective functions are minimized, and detailed explanations of each term are provided in Table I.

Simply identifying the *Pareto frontier* is not sufficient for practical decision making. The final goal is to select a single effective operating point known as the *Pareto optimum*. Since the *Utopia point* is theoretically unreachable, the solution on the Pareto frontier that minimizes the *Utopia distance* is commonly regarded as the *Pareto optimum*. By choosing this point the system can achieve the most balanced trade-off among conflicting objectives without being overly biased toward any single metric.

C. Related Work

Model and Algorithmic Optimization. Prior work has introduced model- and algorithm-centric techniques to reduce computation and memory costs in LLM inference. LLM.int8() [15] enables stable INT8 inference via outlier handling, while SmoothQuant [16] enhances post-training quantization accuracy by adjusting scales. AWQ [17] improves efficiency through activation-aware weight-only quantization without kernel changes. Speculative decoding [18] reduces decoding cost using a smaller auxiliary model. These methods, however, require modifications to the model or pipeline and differ from runtime-level control over the energy–latency trade-off.

GPU Power and Frequency Management. Several recent studies have explored improving energy efficiency in LLM serving by controlling GPU power and frequency at runtime without modifying the model or execution pipeline.

TABLE I: Key terminology in Pareto optimality

Term	Description
Pareto frontier	Set of optimal solutions achieving best efficiency under constraints.
Utopia point	Hypothetical point where all objectives are simultaneously minimized (ideal but unreachable).
Utopia distance	Distance between a specific solution and the Utopia point, used as a metric for optimality.
Optimum point	Best compromise solution selected based on user preference (e.g., minimum distance).
Inefficient area	Region dominated by the Pareto frontier, containing sub-optimal solutions.
Unattainable area	Region that cannot be reached due to physical constraints or system limits.

DynamoLLM [4] analyzes the trade-off between inference latency and energy consumption via GPU frequency scaling and shows that energy efficiency can be improved while satisfying latency constraints. GreenLLM [5] quantitatively studies the impact of GPU power management on performance and energy consumption during LLM inference and demonstrates that power state control plays an important role in energy-efficient LLM serving. Building on this line of work, throttLL'eM [6] proposes a GPU frequency scaling framework that explicitly considers service-level objectives (SLOs) in LLM serving environments and is regarded as the *state-of-the-art* in GPU DVFS-based energy optimization for LLM inference.

EDP Optimization in Other Environments. Recent work has begun adopting the EDP to jointly evaluate energy and performance in LLM inference. However, these studies differ from our work in both optimization targets and serving models: ELLIE [19] targets heterogeneous edge execution, while Camel [20] optimizes GPU frequency and batch size under sequential batching. Thus, we cite them as evidence of EDP's relevance, but do not include them as direct baselines.

III. Motivation

A. Limitations of throttLL'eM

ThrottLL'eM reduces energy consumption by lowering GPU frequency within the bounds of meeting service level objectives (SLOs). This approach is reasonable in that it reliably satisfies latency constraints, and it is currently regarded as a representative state-of-the-art method for GPU DVFS-based LLM serving. However, its frequency selection is determined solely based on whether the incoming query meets the SLO, without considering the execution characteristics of individual iterations, which limits its effectiveness from a global energy optimization perspective. In particular, in continuous batching-based LLM serving, the batch size and computational characteristics vary across iterations, so the minimum frequency that satisfies the SLO may not align with the frequency that maximizes energy efficiency for a given iteration.

throttLL'eM also treats latency solely as a constraint in terms of the SLO, and does not explicitly optimize for any joint metric that considers both energy and latency. As described in §II-A, the EDP is an important metric that evaluates the trade-off between energy and latency, highlighting that energy efficiency can degrade significantly as latency increases. However, selecting the minimum frequency that satisfies the SLO often leads to longer execution times, which can result in suboptimal outcomes from the EDP perspective. In other words, while throttLL'eM is effective in meeting the SLO for each iteration, it falls short of achieving holistic optimization that balances both energy and latency.

B. Aggregate Dynamics: Iteration-level Compute Scaling

As discussed in §II-A, in a multi-workload inference environment, multiple individual requests are packed into a single batch tensor and processed together. As a result, the actual load on the GPU hardware and GPU kernels is determined not by

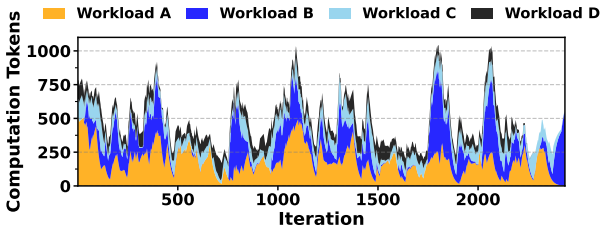


Fig. 3: Aggregate computation tokens (C_{token}) across workloads.

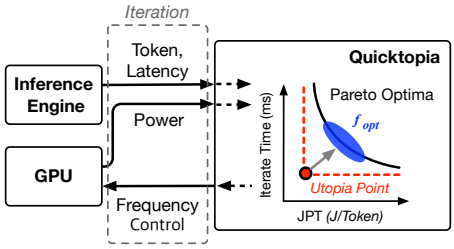


Fig. 4: An overview of QUICKTOPIA.

individual requests, but by the total amount of computation required by the entire batch executed in that iteration.

In this work, we use the number of processed tokens, which is proportional to the actual computational cost (FLOPs), to quantitatively represent the computation load of each iteration. Let $\mathcal{B}$ denote the set of queries in the current iteration's batch, $|g_i|$ the input length of query i, and t_i the index of the token currently being generated during the decoding phase. Then, the token-based computational load of iteration, denoted by C_{token}, is computed as follows:

$$C_{\text{token}} = \sum_{i \in \mathcal{B}} \begin{cases} |g_i|^2, & \text{if } i \in \text{prefill phase} \\ |g_i| + t_i, & \text{if } i \in \text{decode phase} \end{cases} \quad (1)$$

This computation directly reflects the computational characteristics of LLM inference. In the prefill phase, the self-attention operation incurs a quadratic computational cost with respect to the input length ($O(N^2)$), while in the decode phase, the computation scales linearly with the input length due to the use of the KV cache ($O(N)$).

Figure 3 shows the per-iteration C_{token} values of four heterogeneous workloads when they are executed simultaneously, computed using the method described above. We use an NVIDIA RTX 3090 GPU and the vLLM engine for this analysis, with detailed experimental settings provided in §VI. As observed in the graph, each workload exhibits different levels of computational variability across iterations, depending on request length and inference phase (prefill/decode). Consequently, the total computational load (C_{token}) also fluctuates sharply at the iteration level.

Therefore, to maximize energy efficiency through GPU DVFS, it is inadequate to determine the frequency based on individual workloads in isolation or solely on the characteristics of a single workload (e.g., the longest query). Such approaches can result in either over-provisioning, causing idle resources, or under-provisioning, leading to performance degradation. A more appropriate strategy is to determine the GPU frequency from a system-wide perspective based on the aggregated computational demand of the entire batch at each iteration (C_{token}).

C. Proposed Approach

To address the goals and challenges discussed earlier, this work proposes QUICKTOPIA, a new GPU DVFS control scheme that incorporates iteration-level execution characteristics of LLM inference. Figure 4 presents an overview of QUICKTOPIA. The core idea is to evaluate the trade-off between latency and energy for each iteration from a Pareto perspective, and to select the GPU frequency that yields better energy efficiency. To achieve this, QUICKTOPIA employs an online decision-making mechanism that uses observed latency and energy at each iteration to perform Pareto evaluation. In addition, to make the frequency decisions applicable to real LLM serving paths, QUICKTOPIA integrates lightweight per-iteration DVFS control that operates with low overhead. Through this approach, the proposed method goes beyond conservative, SLO-based frequency selection and continuously improves energy efficiency by adapting to the evolving inference characteristics of continuous batching environments.

IV. DESIGN GOALS AND CHALLENGES

This section outlines the design goals (**G**) of QUICKTOPIA and the associated challenges (**Ch**). For each goal, we highlight the key design elements introduced to address it, with reference to the relevant Sections (§) for detailed discussion.

A. Frequency Optimization under Iteration-Level Variability

G1. Iteration-Level Frequency Selection. (§V-B) The first goal is to determine the most efficient frequency for each iteration in LLM inference while jointly considering both latency and energy. Since computational demand varies significantly across iterations, as observed earlier, the frequency should be selected adaptively to accommodate this variability. Furthermore, this decision must be made during the CPU scheduling stage so that it can be applied before GPU execution.

Ch1. Unknown Optimal Frequency. However, determining the optimal frequency for the current iteration in advance is challenging at the CPU scheduling stage. The only information available beforehand is the number of computation tokens. Therefore, a decision mechanism is needed to accumulate and process historical data, such as latency and energy, for different computation levels, so that the appropriate frequency can be inferred from the token count of the current iteration.

B. Pareto Frequency Optimization

G2. Pareto-based Frequency Decision. (§V-C) The second goal is to select the GPU frequency for each iteration by jointly considering latency and energy. Since these metrics are in an inherent trade-off, optimizing one may worsen the other. Thus, each iteration requires selecting a balanced solution from non-dominated candidates. To this end, we adopt Pareto optimality

as the criterion for selecting frequencies that satisfy SLOs while remaining energy-efficient.

Ch2. Robust Pareto Assessment. However, iteration-level observations of latency and energy can be affected by noise. Even under the same computation level, individual measurements may fluctuate due to GPU scheduling randomness or subtle variations in memory access timing. Therefore, an effective technique is required to mitigate noise and reliably identify Pareto-optimal candidates.

C. Low-Overhead Per-Iteration DVFS Control

G3. Per-Iteration DVFS Control. (§V-D) The third goal is to enable practical per-iteration GPU DVFS in LLM serving environments. Since each iteration typically lasts only a few tens of milliseconds, frequency selection must be lightweight enough not to interfere with the inference pipeline. At the same time, it must remain accurate over long-term execution to consistently achieve a favorable latency–energy trade-off.

Ch3. Low-Overhead Runtime Decision. The main challenge is deciding how broadly or finely computation levels should be examined to identify the optimal frequency. An overly fine-grained range increases search overhead, whereas an overly coarse-grained range can lead to suboptimal frequency decisions. Therefore, the system needs a way to adapt the search space granularity according to runtime behavior, preserving efficiency without sacrificing frequency-selection accuracy.

V. Design and Implementation

A. Overview of Quicktopia

As illustrated in Fig. 5(a), the operational workflow of QUICKTOPIA consists of three distinct phases: *analyze*, *apply & exec*, and *update*. QUICKTOPIA maintains a *segment* data structure to store historical execution data for identifying the optimal GPU frequency, and each segment is classified into one of two states: *complete* or *incomplete*.

The roles of the three stages are as follows. In the *analyze* stage, QUICKTOPIA receives from the LLM inference engine the set of queries assigned to the current iteration, the input length of each query, and the current step index. Using this information, it determines for each query whether the current step corresponds to prefill or decode, and then computes the total computational demand of the iteration, C_{token}, using Eq. 1. Based on this demand, QUICKTOPIA identifies the segment corresponding to the current iteration. In the *apply & exec* stage, QUICKTOPIA consults the selected segment to decide the GPU frequency, applies the chosen frequency, and executes the iteration. Finally, in the *update* stage, QUICKTOPIA updates the statistics stored in the segment based on the observed execution results.

B. Segment Management

As illustrated in Fig. 5(b), QUICKTOPIA manages the range of computational demand using a tree-structured segment manager to enable efficient frequency control. Each segment corresponds to a leaf node in the tree, ensuring $O(\log n)$ lookup time and keeping the online control overhead low even when managing a large number of segments. Each segment maintains a data table that is in one-to-one correspondence with the list of configurable GPU frequencies exposed by the vendor. For each frequency, the table stores statistical metrics such as JPT, latency, reference count, and Utopia distance, and QUICKTOPIA uses these statistics to dynamically maintain a single optimal frequency for each segment.

To ensure statistical reliability in frequency selection, each segment is classified as either *incomplete* or *complete*, based on whether the minimum exploration requirement is met. We define a hyperparameter Ref_{min}, which specifies the minimum number of observations required per frequency. A segment is considered *incomplete* if any candidate frequency has been explored fewer than Ref_{min}, indicating that the corresponding performance statistics remain unreliable. In this state, QUICKTOPIA continues to profile all underexplored frequencies. Once every frequency in the segment has met the Ref_{min} threshold, the segment transitions to the *complete* state, in which reliable optimal frequency selection becomes possible based on accumulated observations.

Incomplete Segment. The upper portion of Fig. 5(c) illustrates the update procedure when a segment is in the *incomplete* state. ① An incomplete segment represents an early stage in which the statistical reliability of its data is insufficient. During this profiling phase, QUICKTOPIA collects performance statistics for all candidate frequencies. Frequencies are applied sequentially from f_{max} to f_{min}, and this process continues until the number of samples for each frequency reaches the threshold defined by Ref_{min}. ② For each execution, QUICKTOPIA computes the JPT from the observed latency and energy consumption, and updates the segment's statistics using an incremental average scheme that aggregates these values into the segment's internal table. ③a If some frequencies still fall short of the Ref_{min} requirement, the segment remains incomplete, and the system continues profiling those remaining frequencies during subsequent iterations. ③b Once all candidate frequencies have been sampled at least Ref_{min}, the segment transitions to the *complete* state. At this point, QUICKTOPIA normalizes the latency and JPT metrics across all frequencies to compute their Utopia distances. Rather than using these raw distances directly, the system converts them into a scalar reward value via a scoring function, which is then used to select the optimal frequency. The detailed method for computing and updating this selection is described in §V-C.

Complete Segment. Once a segment transitions to the *complete* state, the system performs real-time optimization based on the accumulated statistics, as shown in the lower part of Fig.5(c). As in the incomplete case, QUICKTOPIA receives iteration-level information from the LLM inference engine and computes the total computational demand, C_{token}, using Eq.1. This value is used as a key to identify the corresponding segment range and retrieve the segment responsible for frequency selection for the current iteration. ❶ Unlike in the incomplete state, each complete segment maintains a pre-determined optimal frequency. QUICKTOPIA applies the

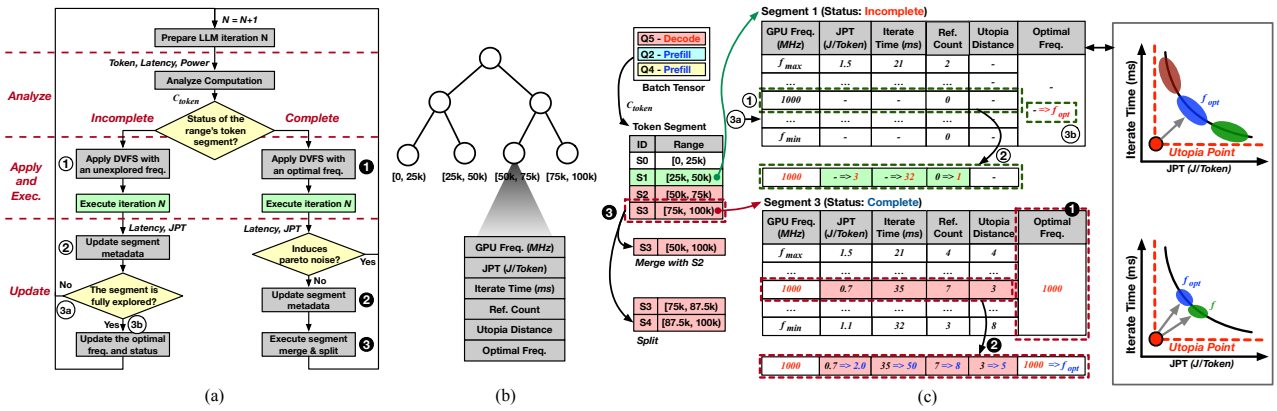


Fig. 5: Detailed design of QUICKTOPIA. (a) Overall workflow of QUICKTOPIA, (b) Segment data structure, (c) Segment update policy for different segment states.

optimal frequency before execution begins, ensuring that the GPU operates at the selected frequency throughout the iteration. ❷ After GPU execution completes, the system does not immediately update the segment's latency and JPT statistics. Instead, it first evaluates whether the newly observed data point is affected by noise. If the measurement is deemed reliable, it is integrated into the segment's existing statistics using the same incremental averaging method employed during the incomplete phase. The procedure for detecting and filtering out noisy samples is detailed in §V-C. ❸ If the newly observed data point is classified as non-noisy and successfully incorporated into the segment's statistics, QUICKTOPIA then evaluates whether the current segment satisfies the trigger conditions for a *merge* or *split* operation. If any condition is met, the corresponding structural update is performed on the segment tree to better reflect the distribution of compute demand. The specific criteria and procedures for segment merging and splitting are detailed in §V-D.

C. Detail of Pareto Optima

This method is applied both when a segment transitions to the *complete* state during the update phase, and whenever a complete segment is further updated. Its primary role is to filter out noisy observations while simultaneously improving the accuracy of Pareto frontier estimation. The goal of this technique is to select a GPU DVFS configuration that best balances the trade-off between two competing objectives: latency and JPT. The method proceeds through several stages, first detecting and excluding noisy samples, and then normalizing iteration time and JPT across all candidate frequencies. Using these normalized values, it computes the Utopia distance for each frequency and converts the distance into a scalar reward score, which is finally used to determine the selected frequency for the current segment.

Noise Reduction. The latency and JPT data collected at runtime are prone to outliers caused by transient system disturbances such as background load fluctuations or GPU driver scheduling noise. These outliers can distort the data distribution during normalization, degrading the accuracy of subsequent optimization steps. To address this, QUICKTOPIA employs the Welford algorithm [21] to assess the statistical validity of each incoming data point in constant time, with $O(1)$ complexity. Welford's method incrementally updates the running mean and variance without requiring storage of the entire history, ensuring minimal overhead. Based on the updated statistics, QUICKTOPIA maintains a confidence interval for expected values. Any new observation that falls outside this interval is classified as noise and excluded from further analysis. This filtering mechanism ensures that the training data remains robust and reliable, preserving the integrity of the optimization process.

Normalization. If a measurement is determined to be non-noisy, it is incorporated into the segment's statistical records. Since the two optimization objectives, latency and JPT, exist on different numerical scales, directly comparing or combining their raw values can result in one metric disproportionately dominating the decision process. To prevent this imbalance, QUICKTOPIA applies min-max normalization to both metrics using the observed value ranges within the current segment. As shown in Eq. 2, the normalized latency ($\hat{L}_i$) and normalized JPT ($\hat{J}_i$) are mapped to a common scale in the range $[0, 1]$, enabling fair comparison and balanced decision-making across the two objectives.

$$\hat{L}_i = \frac{L_i - L_{\min}}{L_{\max} - L_{\min}}, \qquad \hat{J}_i = \frac{J_i - J_{\min}}{J_{\max} - J_{\min}} \tag{2}$$

Reward Function Setting. Since a single outlier can skew the statistical average and undermine stability when using simple distance, we adopt a more robust alternative: a Utopia-distance-based reward function that combines normalized latency and JPT into a single scalar value for each iteration, which is used in place of the segment's raw Utopia distance. In Eq. 3, r_i is defined as the negative exponential of the Euclidean distance to the Utopia point in the normalized space. This reward increases as both latency and JPT decrease (i.e., as the point approaches the Utopia point) and drops sharply when either metric deteriorates, naturally discouraging unbalanced configurations that over-optimize only one metric. By mapping the Utopia distance to a bounded scalar in the range $(0, 1]$ through the exponential function, the influence

of outliers on cumulative averages and cross-frequency comparisons is mitigated, enabling statistically stable decision-making. Consequently, the reward function provides a robust scalar representation of the latency–energy trade-off.

$$r_i = \exp\left(-\sqrt{\hat{L}_i^2 + \hat{J}_i^2}\right) \tag{3}$$

Statistical Aggregation. In an iteration-level environment, performance observations can vary even under the same frequency, depending on the execution conditions. Therefore, making decisions based on the reward from a single iteration is unstable and a certain degree of statistical aggregation is required. To this end, as shown in Eq. 4, we maintain the cumulative average reward $\hat{\mu}_{s,f}(t)$ for each combination of segment s and frequency f over time. This cumulative average reflects the long-term performance trend of each frequency and improves the accuracy of the Pareto frontier.

$$\hat{\mu}_{s,f}(t) = \frac{1}{n_{s,f}(t)} \sum_{k=1}^{n_{s,f}(t)} r_{s,f}^{(k)} \tag{4}$$

Optimal Frequency. Finally, for each segment s, we define the GPU frequency with the highest cumulative average reward as the optimal frequency of that segment. As shown in Eq. 5, the optimal frequency f_s is selected as the frequency that maximizes the cumulative average reward. This definition implies that, within the computational range represented by the segment, the chosen frequency best achieves the trade-off between latency and JPT, and since it is based on long-term observations, it enables stable policy decisions.

$$f_s = \arg\max_{f \in \mathcal{F}} \hat{\mu}_{s,f} \tag{5}$$

D. Segment Split and Merge

QUICKTOPIA divides and manages the range along the computational axis C_{token} into segments in order to reduce system overhead, configure DVFS quickly at the iteration level, and maintain iteration-level DVFS stably over long periods. In real-world serving environments, workload distribution changes over time, and the optimal frequency may differ even under similar computational demand. Therefore, if the segment boundaries are kept fixed, multiple distinct optimal policies may coexist within a single segment, reducing optimization accuracy; conversely, segments may become unnecessarily fine-grained, increasing the overhead of state management and exploration. To address this, QUICKTOPIA supports *split* and *merge* operations that reconfigure segment boundaries at runtime. Since segment ranges do not overlap, the segment manager can be implemented as a simple binary tree, and by using this tree-based segment manager, lookup as well as split/merge operations can all be performed in $O(\log n)$ time.

Split. Segment splitting is a policy for improving the accuracy of iteration-level DVFS and is performed when a single segment is too wide to be represented by a single DVFS policy. As illustrated by the Pareto frontier in the lower right of Fig. 5(c), if different optimal frequencies are required within a single segment, it is more accurate to split that segment and learn the optimal frequency independently for each sub-range. To this end, QUICKTOPIA allows splitting only after a segment has sufficiently converged, that is, only after statistical reliability has been secured. Specifically, the split condition is evaluated only when the cumulative number of samples for segment s is greater than or equal to the minimum threshold Ref_{min}^{total}, and in our implementation we empirically set $Ref_{min}^{total} = 2000$. Due to iteration-level measurement noise, the optimal frequency can fluctuate in the early stage. Therefore, we reconfigure the boundaries only after sufficient observations have been accumulated so that the optimal frequency selection has converged.

The condition that triggers a split is defined as the case where two or more candidate frequencies within a segment are selected as the optimal frequency with nearly equal quality. Specifically, for the current segment's optimal frequency f_s, if another frequency f does not differ from f_s by more than a certain ratio in cumulative average reward, f is regarded as a competing equivalent policy, and a split is triggered. Specifically, a split is triggered when Eq. 6 is satisfied.

$$\hat{\mu}_{s,f} \geq (1-\delta)\,\hat{\mu}_{s,f_s} \tag{6}$$

Here, $\hat{\mu}_{s,f}$ is the cumulative average reward of frequency f in segment s, and $\delta \in (0,1)$ is a hyperparameter for determining whether two frequencies are nearly equivalent.

Merge. Conversely, merging is an operation aimed at reducing system overhead. If two adjacent segments s and s' share the same optimal frequency, their ranges can be merged under a single policy with negligible Pareto loss. Therefore, as in Eq. 7, when the optimal frequencies of adjacent segments are identical, QUICKTOPIA merges the two segments into one, reducing the number of states and alleviating system overhead.

$$f_s^* = f_{s+1}^* \;\Rightarrow\; \text{merge}(s, s+1). \tag{7}$$

After merging, the resulting segment updates its cumulative averages by combining the statistics of the two original segments, and from the next iteration onward, profiling continues at the granularity of this new segment. Consequently, the merge operation automatically groups ranges that can be represented by the same policy, and it works complementarily with splitting to maintain a balance between optimization accuracy and runtime overhead.

VI. EVALUATION

A. Experimental Setup

Platform and LLM Models. All experiments in this study were conducted on a single server equipped with an Intel Core i7-13700F CPU, 64GB RAM, and a single NVIDIA RTX 3090 GPU with 24GB of VRAM. The LLM models used in the experiments are Qwen2-7B [22] and Qwen2.5-32B-AWQ [23].

Compared Systems. To evaluate the performance of QUICKTOPIA, we define the following three system environments as baselines for comparison. Here, both throttLL'eM and

QUICKTOPIA are not standalone serving engines, but GPU frequency control layers integrated into the vLLM serving engine. Thus, all comparisons are conducted on the same vLLM execution environment, differing only in the frequency control mechanism.

- **vLLM** [13]: A representative LLM serving engine that achieves high throughput and low latency via continuous batching and efficient KV-cache management. We use vLLM as the baseline system without any GPU power or frequency control.
- **throttLL'eM** [6]: A state-of-the-art DVFS system that dynamically scales GPU frequency based on request latency to meet SLOs and reduce energy consumption during LLM serving.
- **QUICKTOPIA**: The system proposed in this paper, which dynamically adjusts GPU frequency at the iteration level by jointly considering latency and energy, improving both response time and energy efficiency in LLM serving.

Methodology. All frameworks were executed with the same inputs, output sizes, and a temperature of 0.0 to eliminate nondeterminism in generation, ensuring fair comparison of latency and JPT. For iteration-level power measurement, we used workloads derived from ShareGPT-small [24] and ShareGPT-4 [25], which reflect realistic user interaction patterns. Request arrivals followed a Poisson process, and the RPS (requests per second) was varied from 1 to 5 to analyze power behavior under different load levels. Power was measured using the NVIDIA Management Library (NVML), sampling total GPU power every 5 ms, and iteration-level energy consumption was computed by integrating the sampled power over each iteration's execution window.

throttLL'eM Configurations. Since throttLL'eM is not publicly available, we implemented an oracle version by fixing the temperature to 0.0 so that the output length is known in advance. The IPS (Iterations per Second) prediction model, based on Gradient Boosted Trees (GBT), was trained separately for each model size. For training, we used all selectable frequencies from the GPU vendor's supported frequency list and, for each frequency, pre-ran inference on the two datasets used in our experiments.

QUICKTOPIA Configurations. In our experiments, the minimum number of exploration trials per frequency within a segment, Ref_{min}, was set to 2 to ensure basic statistical confidence. The threshold for segment splitting, defined as the relative difference in cumulative average rewards δ, was set to 0.5 so that splitting occurs only when statistically similar frequency candidates coexist. QUICKTOPIA performs its initial profiling using a single pass over the dataset, without any large-scale offline pretraining.

B. Overall Performance

Fig. 6 presents the results of total energy consumption, total latency, and EDP measured during a single benchmark run for all models and datasets evaluated in this study. Each metric is normalized against the baseline vLLM. For all metrics, lower values indicate better performance.

As illustrated in Fig. 6(a), throttLL'eM shows the lowest energy usage across all models. This is because throttLL'eM adopts an energy-centric policy that minimizes GPU frequency as long as latency SLOs are satisfied. In contrast, while QUICKTOPIA consumes slightly more energy than throttLL'eM, it still achieves up to a 22% reduction in energy consumption compared to the baseline.

Next, the latency results in Fig. 6(b) clearly reveal the limitations of optimizing for energy alone. throttLL'eM shows up to a 36% increase in latency compared to vLLM. In contrast, QUICKTOPIA keeps latency increases within 1–6% across all experimental settings, maintaining responsiveness close to the baseline.

This difference becomes even more apparent in the EDP results shown in Fig. 6(c). Although throttLL'eM significantly reduces energy consumption, the increased latency offsets the gain, resulting in negligible improvement in EDP—and in some cases, even degradation. In contrast, QUICKTOPIA achieves the lowest EDP across all models by selecting pareto-optimal frequencies that balance both latency and energy. As a result, it delivers up to a 21% improvement in end-to-end efficiency compared to the baseline.

C. Runtime Analysis

Fig. 7 illustrates the runtime variations of JPT, iteration time, and EDP observed at the iteration level during inference on the gpt-small dataset using the Qwen2-7B model. Since each iteration has a different batch composition and computational load, performance and energy metrics exhibit significant fluctuations over time in real-world serving environments.

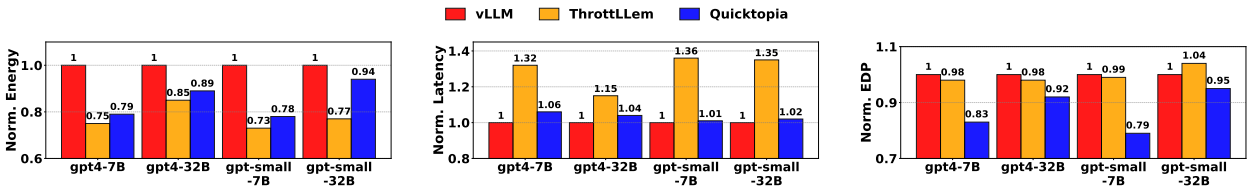
As shown in Fig. 7(a), vLLM exhibits high energy consumption with large fluctuations, while throttLL'eM reduces the average energy but suffers from inefficient energy spikes depending on the iteration. In contrast, QUICKTOPIA maintains consistently low JPT by making DVFS decisions that reflect the computational load of each iteration.

As shown in Fig. 7(b), throttLL'eM exhibits generally higher iteration time from a latency perspective, with elevated latency persisting for extended periods. In contrast, QUICKTOPIA consistently maintains a latency range similar to that of vLLM, while also reducing the frequency of iteration-level latency spikes.

This difference becomes even more pronounced in the EDP results shown in Fig. 7(c). Despite reducing energy consumption, throttLL'eM frequently exhibits EDP spikes due to latency fluctuations. In contrast, QUICKTOPIA maintains the lowest EDP across most iterations, demonstrating that its end-to-end efficiency improvements are not merely transient, but consistently sustained throughout the runtime.

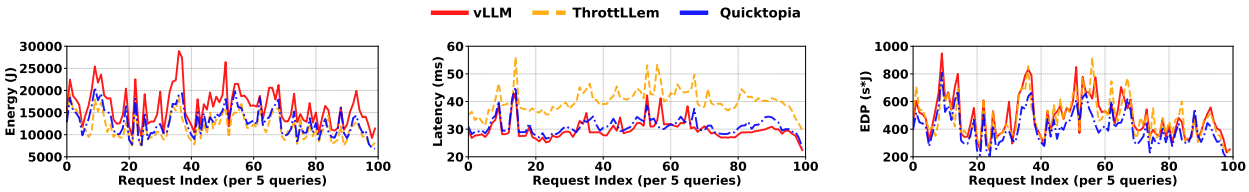
D. SLO-Constrained Energy Minimization

Similar to throttLL'eM, the Pareto-based optimization of QUICKTOPIA can be adapted to incorporate SLO constraints. To demonstrate this capability, we evaluated an additional



(a) Energy consumption (b) Latency (c) EDP (Energy-Delay Product)

Fig. 6: End-to-end results for various LLM inference workloads, normalized to vLLM. For all metrics, lower is better. QUICKTOPIA consistently achieves the lowest EDP across all models.



(a) JPT (Joule Per Token) (b) Iteration Time (c) EDP (Energy-Delay Product)

Fig. 7: Runtime trace of JPT, Iteration Time, and EDP (gpt-small-7B). For all metrics, lower is better. QUICKTOPIA consistently achieves lowest EDP with stable iteration times.

TABLE II: End-to-end performance under TBT SLO.

Dataset & Model	Config	TBT SLO	Lat. (s)	Energy (kJ)	EDP (MJ·s)
gpt4-7B	throttLL'eM	✓	884	197	174
	QUICKTOPIA (SLO)	✓	**732**	**184**	**135** *(0.78×)*
gpt4-32B	throttLL'eM	✓	559	171	96
	QUICKTOPIA (SLO)	✓	**559**	**153**	**85** *(0.89×)*
gpt-small-7B	throttLL'eM	✓	880	193	170
	QUICKTOPIA (SLO)	✓	**787**	**190**	**150** *(0.88×)*
gpt-small-32B	throttLL'eM	✓	761	180	137
	QUICKTOPIA (SLO)	✓	**681**	**179**	**122** *(0.89×)*

configuration using a modified reward function (Eq. 3). Specifically, instead of utilizing the two-dimensional Pareto distance based on both latency and energy, we employed the Euclidean distance of the single energy metric. Furthermore, to strictly prevent SLO violations, a zero reward was assigned to any observation points exceeding the SLO threshold.

Table II presents the end-to-end performance comparison under Time-Between-Tokens (TBT) SLO constraints. In all experimental settings, both QUICKTOPIA and throttLL'eM are configured to satisfy the identical TBT SLO, sharing the common objective of optimizing energy efficiency under latency constraints. Thus, this ensures a fair comparison within an SLO-constrained environment.

Consequently, QUICKTOPIA achieves a lower EDP compared to throttLL'eM across all model and dataset combinations. Specifically, in the gpt4-7B setting, it reduces latency by over 17% while simultaneously lowering energy consumption, thereby improving EDP by up to 22%. Furthermore, across other settings, QUICKTOPIA demonstrates consistent improvements ranging from 11% to 22% in terms of EDP.

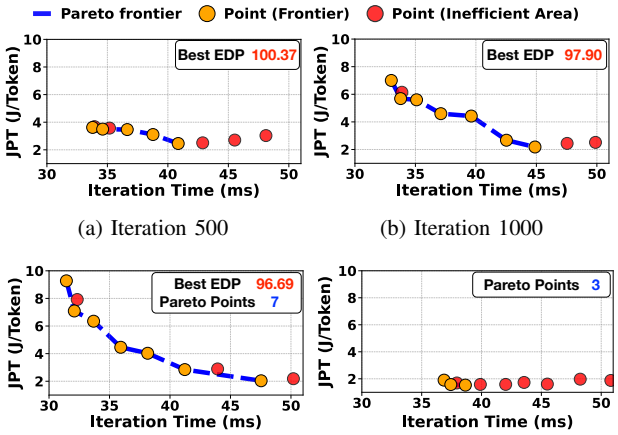
This performance improvement stems from the difference in the granularity of frequency selection, as discussed in §III-A, even though both systems satisfy the identical SLO constraints. While throttLL'eM primarily performs frequency selection at the arrival of new queries, QUICKTOPIA selects the most energy-efficient frequency more precisely at the iteration level by reflecting the energy–latency characteristics of each execution phase. Consequently, this enables the effective reduction of unnecessary power consumption without violating the latency SLO.

E. Pareto Frontier Accuracy

Fig. 8 illustrates how the Pareto frontier forms and converges as iterations accumulate for the same computation segment. Each point represents the observed iteration time and JPT at a specific GPU frequency, while the blue dashed line indicates the Pareto frontier identified at that iteration point.

Fig. 8(a)–(c) present the cumulative results after 500, 1,000, and 2,000 iterations, respectively. As the number of iterations increases, the Pareto frontier gradually stabilizes and converges toward more optimal points within the energy-latency trade-off space. To quantitatively verify this improvement, the best EDP observed at each stage is annotated in the upper-right corner of each graph. Consequently, the Best EDP continuously improves to 100.37, 97.9, and 96.69, confirming that the overall system efficiency is substantially enhanced as learning progresses.

Fig. 8(c) and (d) both represent results after 2,000 iterations. However, (d) depicts raw log data without the noise reduction filter. In this case, transient fluctuations in latency and JPT cause intermittent outliers to be misidentified as part of the Pareto frontier. Consequently, the frontier exhibits a discontinuous shape and contains only three Pareto points, failing to provide reliable operating points for control policy selection.



(a) Iteration 500 (b) Iteration 1000

(c) Iteration 2000 (d) Iteration 2000 without filter

Fig. 8: Evolution of Pareto frontier during design space exploration.

In contrast, (c), which applies Welford's algorithm-based filtering, displays a continuous frontier shape and identifies seven Pareto points. This indicates that the system offers a diverse set of stable and consistent frequency options. Ultimately, these results clearly demonstrate that a noise-robust Pareto assessment technique is essential for ensuring accurate and reliable control in iteration-level environments.

VII. Conclusion

In this paper, we proposed QUICKTOPIA, an iteration-level GPU frequency control framework designed to address the computational variability and latency-energy trade-offs inherent in LLM inference. QUICKTOPIA effectively manages runtime variability by synergistically integrating tree-based segment search, noise-robust Utopia distance optimization, and dynamic segment splitting and merging mechanisms. Experimental results demonstrate that QUICKTOPIA reduces the EDP by up to 22% compared to the state-of-the-art method, throttLL'eM.

Acknowledgment

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korean government (MSIT) (RS-2024-00453929) (RS-2024-00416666).

References

[1] C. Tian, X. Qin, K. Tam, L. Li, Z. Wang, Y. Zhao, M. Zhang, and C. Xu, "Clone: customizing llms for efficient latency-aware inference at the edge," in *Proceedings of the 2025 USENIX Conference on Usenix Annual Technical Conference*, USENIX ATC '25, (USA), USENIX Association, 2025.

[2] Y. Sheng, L. Zheng, B. Yuan, Z. Li, M. Ryabinin, B. Chen, P. Liang, C. Ré, I. Stoica, and C. Zhang, "Flexgen: high-throughput generative inference of large language models with a single gpu," in *Proceedings of the 40th International Conference on Machine Learning*, ICML'23, JMLR.org, 2023.

[3] Y. Fu, L. Xue, Y. Huang, A.-O. Brabete, D. Ustiugov, Y. Patel, and L. Mai, "ServerlessLLM: Low-Latency serverless inference for large language models," in *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, (Santa Clara, CA), pp. 135–153, USENIX Association, July 2024.

[4] J. Stojkovic, C. Zhang, Goiri, J. Torrellas, and E. Choukse, "Dynamollm: Designing llm inference clusters for performance and energy efficiency," in *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, p. 1348–1362, IEEE, Mar. 2025.

[5] Q. Liu, D. Huang, M. Zapater, and D. Atienza, "Greenllm: Slo-aware dynamic frequency scaling for energy-efficient llm serving," 2025.

[6] A. K. Kakolyris, D. Masouros, P. Vavaroutsos, S. Xydis, and D. Soudris, "throttll'em: Predictive gpu throttling for energy efficient llm inference serving," in *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 1363–1378, 2025.

[7] Z. Tang, Y. Wang, Q. Wang, and X. Chu, "The impact of gpu dvfs on the energy and performance of deep learning: an empirical study," 2019.

[8] P. J. Maliakel, S. Ilager, and I. Brandic, "Investigating energy efficiency and performance trade-offs in llm inference across tasks and dvfs settings," 2025.

[9] Y. Zhao, C.-Y. Lin, K. Zhu, Z. Ye, L. Chen, S. Zheng, L. Ceze, A. Krishnamurthy, T. Chen, and B. Kasikci, "Atom: Low-bit quantization for efficient and accurate llm serving," 2024.

[10] C. Niu, W. Zhang, J. Li, Y. Zhao, T. Wang, X. Wang, and Y. Chen, "Tokenpowerbench: Benchmarking the power consumption of llm inference," 2025.

[11] S. P. Rachuri, N. Shaik, M. Choksi, and A. Gandhi, "Ecoedgeinfer: Dynamically optimizing latency and sustainability for inference on edge devices," in *2024 IEEE/ACM Symposium on Edge Computing (SEC)*, pp. 191–205, 2024.

[12] M. M. S. Aly, M. Gao, G. Hills, C.-S. Lee, G. Pitner, M. M. Shulaker, T. F. Wu, M. Asheghi, J. Bokor, F. Franchetti, *et al.*, "Energy-efficient abundant-data computing: The n3xt 1,000 x," *Computer*, vol. 48, no. 12, pp. 24–33, 2015.

[13] W. Kwon, Z. Li, S. Zhuang, Y. Sheng, L. Zheng, C. H. Yu, J. Gonzalez, H. Zhang, and I. Stoica, "Efficient memory management for large language model serving with pagedattention," in *Proceedings of the 29th symposium on operating systems principles*, pp. 611–626, 2023.

[14] G.-I. Yu, J. S. Jeong, G.-W. Kim, S. Kim, and B.-G. Chun, "Orca: A distributed serving system for Transformer-Based generative models," in *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, (Carlsbad, CA), pp. 521–538, USENIX Association, July 2022.

[15] T. Dettmers, M. Lewis, Y. Belkada, and L. Zettlemoyer, "Gpt3. int8 (): 8-bit matrix multiplication for transformers at scale," *Advances in neural information processing systems*, vol. 35, pp. 30318–30332, 2022.

[16] G. Xiao, J. Lin, M. Seznec, H. Wu, J. Demouth, and S. Han, "Smoothquant: Accurate and efficient post-training quantization for large language models," in *International conference on machine learning*, pp. 38087–38099, PMLR, 2023.

[17] J. Lin, J. Tang, H. Tang, S. Yang, W.-M. Chen, W.-C. Wang, G. Xiao, X. Dang, C. Gan, and S. Han, "Awq: Activation-aware weight quantization for on-device llm compression and acceleration," *Proceedings of machine learning and systems*, vol. 6, pp. 87–100, 2024.

[18] Y. Leviathan, M. Kalman, and Y. Matias, "Fast inference from transformers via speculative decoding," in *International Conference on Machine Learning*, pp. 19274–19286, PMLR, 2023.

[19] H. Fan, Y.-C. Lin, and V. Prasanna, "Ellie: Energy-efficient llm inference at the edge via prefill-decode splitting," in *2025 IEEE 36th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, pp. 139–146, IEEE, 2025.

[20] H. Xu, L. Peng, S. Song, X. Liu, M. Jun, S. Li, J. Yu, and X. Mao, "Camel: Energy-aware llm inference on resource-constrained devices," *arXiv preprint arXiv:2508.09173*, 2025.

[21] B. P. Welford, "Note on a method for calculating corrected sums of squares and products," *Technometrics*, vol. 4, no. 3, pp. 419–420, 1962.

[22] Q. Team *et al.*, "Qwen2 technical report," *arXiv preprint arXiv:2407.10671*, vol. 2, no. 3, 2024.

[23] A. Yang, B. Yang, B. Zhang, B. Hui, B. Zheng, B. Yu, C. Li, D. Liu, F. Huang, H. Wei, *et al.*, "Qwen2. 5 technical report," *arXiv preprint arXiv:2412.15115*, 2024.

[24] H. Larcher, "share_gpt_small dataset." https://huggingface.co/datasets/hlarcher/share_gpt_small, 2024. Accessed: 2025-12-15.

[25] G. Wang, S. Cheng, X. Zhan, X. Li, S. Song, and Y. Liu, "Openchat: Advancing open-source language models with mixed-quality data," 2024.