Safety-Constrained Contextual Bandit for Dynamic Power Management

Yasin Esfandiari[1,2], Karan Rajshekar[1,2,*], Cicy Agnes[2], Hannah Stein[1,2], Sabine Janzen[2] and Wolfgang Maass[1,2]

[1]Saarland University, Saarbrücken, Germany
[2]German Research Center for Artificial Intelligence (DFKI), Saarbrücken, Germany

Abstract

Real time AI model serving workloads are characterized by bursty and variable request patterns, yet production GPUs commonly operate at their maximum frequency, wasting substantial energy whenever latency headroom exists. Reducing energy consumption without violating tail-latency SLOs is challenging because frequency scaling affects latency in highly nonlinear, workload-dependent ways. This paper presents *HALO* (Hierarchical Adaptive Latency-Oriented DVFS), a fully black-box power-management controller that uses probe p95 latency to classify the system into coarse safety zones, gates the set of admissible frequency actions per zone, and runs a contextual bandit within those constraints to adapt online, requiring no offline profiling, model instrumentation, or serving-framework modifications. For LLM inference on the Azure LLM Inference Trace (Coding workload), HALO reduces total GPU energy by 39.4% on the primary model and up to 50.0% across model scales, while maintaining tail latency under the SLO. Applied to LLM pretraining, the same controller reduces energy with convergence unaffected, demonstrating that the approach generalizes across both inference and training workloads.

Keywords

GPU DVFS, Energy Efficiency, LLM Inference, LLM Training, Contextual Bandit, Safety Constraints, Service-Level Objectives, Online Learning

1. Introduction

Foundation model based AI services are becoming a central component of modern systems, and training and serving them in real time places a heavy and continuous demand on GPUs [1]. Large language and multimodal models now consume a meaningful share of data-center power. A single LLM inference query can draw tens of watt-hours, the largest training runs sit in the megawatt range, and inference is on track to dominate the lifetime energy footprint of deployed models [2, 3, 4]. In practice, production GPUs are still run with their streaming multiprocessor (SM) clocks at or near the maximum frequency [2]. A lot of available energy savings is left unused.

The most direct way to recover those savings is GPU dynamic voltage and frequency scaling (DVFS), but the energy-frequency relationship makes this harder than it looks. For LLM inference, decode tokens are insensitive to SM clock at moderate load, so the controller can cut frequency aggressively and pay almost nothing in latency; at higher load, the same cuts produce nonlinear blow-ups in tail latency, and the policy has to back off quickly. For training, the situation is different but shares the same structural property: total energy as a function of SM frequency has a valley, dropping as the clock comes down from maximum until the slowdown begins to dominate, after which energy rises again [5]. In both regimes the controller's job is to find the efficient operating region online and stay inside it as conditions change.

Two more constraints shape the design space. The first is observability: production GPU stacks expose only coarse telemetry (power, utilization, end-to-end latency) through NVML. Kernel timelines, KV-cache statistics, and serving-framework internals are usually unavailable [3], so anything that relies on them is hard to deploy. The second is non-stationarity: serving traces are bursty [6] and training

SuRE'26: Workshop on Sustainability and Resource-Efficiency of Artificial Intelligence, August 17, 2026, Bremen, Germany
*Corresponding author.
y.esfandiari@iss.uni-saarland.de (Y. Esfandiari); karan.rajshekar@uni-saarland.de (K. Rajshekar); cicy.agnes@dfki.de (C. Agnes); hannah.stein@dfki.de (H. Stein); sabine.janzen@dfki.de (S. Janzen); wolfgang.maass@dfki.de (W. Maass)

jobs pass through warmup, steady-state, and evaluation phases over long horizons. Prior controllers handle some subset of these requirements, but none of them all; Section 2 covers the comparison in detail.

For an energy controller to be deployable, the savings number is not the only thing that matters. The controller also has to keep the workload's performance constraint from slipping. A DVFS policy that occasionally violates a tail-latency SLO is not really an efficiency tool. We therefore enforce safety as a hard constraint on the action set, not as a penalty term in a learned objective: at every decision, the controller picks from a set of actions that have already been declared safe by an external signal.

We present HALO, a black-box DVFS controller built around this idea. The core is a LinUCB contextual bandit [7]; everything else is a thin wrapper that defines which actions the bandit is allowed to consider. For LLM inference, the wrapper is a three-zone classifier (GREEN, AMBER, RED) on probe p95 latency relative to the SLO, with each zone permitting a different subset of SM frequency adjustments. LinUCB chooses among the permitted actions to minimize an energy-squared-delay objective. For LLM training, the wrapper is replaced by a short pre-training calibration sweep that locates the energy-efficient range $[f_{\text{floor}}, f_{\max}]$ around the energy valley. The same LinUCB bandit then runs online inside that range, minimizing energy per step, since training has no per-request latency signal to defend. Optimizing energy per step naturally balances energy savings against training time. This metric naturally limits training time overhead, since frequencies that save little energy but add significant time are penalized by a higher energy-per-step cost. In both cases, the controller reads NVML telemetry and an optional end-to-end probe, runs as a separate process next to the workload, and needs no changes to the serving or training framework.

The contributions of this paper are the following.

- A unified formulation of runtime GPU DVFS for LLM inference and training as a contextual-bandit problem under partial observability, with safety enforced structurally on the action set rather than through reward shaping.
- HALO, a concrete controller built from this formulation: a three-zone safety wrapper plus LinUCB for inference, and a calibration-guided operating range plus the same LinUCB for training.
- An SLO-aware E^2D objective for inference and an energy-only objective for training, both shaped to give LinUCB a stable learning signal.
- An evaluation on a 12-hour replay of the Azure LLM Inference Trace across three model scales (Mistral-7B, Mistral-24B, Qwen-32B), four offered-load levels, three SLOs, and a data-parallel two-GPU setup, with total GPU energy reduced by 39.4% on the primary Mistral-24B model and up to 50.0% across model scales, at throughput within 0.3% of the always-max baseline. For LLM training, we evaluate the same controller on Pythia-6.9B pretraining and report 19.4% energy reduction relative to the always-max baseline, with convergence unaffected.

2. Related Work

GPU DVFS for LLM inference. DynamoLLM [6] jointly tunes instance count, parallelism, and GPU frequency at cluster scale using offline energy profiles. throttLL'eM [8] predicts per-iteration KV-cache utilization and batch size to set GPU frequency via a learned regression model. AGFT [9] trains a deep-RL agent for fine-grained SM frequency control. GreenLLM [10] applies phase-aware DVFS separating prefill and decode via fitted latency-power models. DualScale [11] extends DVFS-under-SLO to disaggregated serving with MPC-based fine-grained control. All five depend on either offline profiling, workload-specific fitted models, or deep RL whose safety is encoded through reward shaping rather than as a structural constraint on the action set. HALO instead uses a model-free contextual bandit with a probe-based three-zone safety wrapper that makes unsafe actions inadmissible rather than merely costly, providing structural SLO protection with no framework instrumentation.

GPU energy for DNN training. McDonald et al. [12] show that a static GPU power cap yields around 15% energy reduction when training language models. Zeus [13] applies a multi-armed bandit over batch

size and power limit, but operates across job recurrences rather than within a single run. Perseus [14] pre-characterizes the full pipeline-parallel time-energy Pareto frontier via graph-cut and produces a one-shot schedule, with no online learner active during training. GPOEO [15] and DRLCap [16] adapt GPU frequency online but without any pre-run characterization of where the energy-efficient operating range lies. Existing training-side controllers thus fall into two camps: *calibration-only* (Perseus) and *online-only* (Zeus, GPOEO, DRLCap), with static power capping at one end. HALO occupies the missing fourth point, combining a short calibration sweep that identifies $[f_{\text{floor}}, f_{\max}]$ around the energy valley with the same LinUCB bandit [7] used for inference, which to our knowledge is the first controller to unify both regimes under a single online learner.

Safe online learning for systems control. LinUCB has been applied to cloud resource management in Erlang [17], which uses a contextual bandit to autoscale microservice replicas and outperforms utilization- and supervised-ML baselines while generalizing to unseen workloads. For decisions whose safety must be guaranteed independently of learner confidence, conservative bandits [18] constrain exploration to actions that provably maintain a baseline performance level. HALO's three-zone wrapper (inference) and calibration-based range (training) instantiate this principle operationally: an external safety signal gates the bandit's action set (probe p95 latency versus SLO for inference; position on the energy valley curve for training), and LinUCB only explores within the remaining safe set.

3. Method

We present a black-box DVFS controller that adjusts the GPU SM frequency cap at runtime to reduce energy consumption. The controller combines a safety wrapper, which restricts which frequency actions are admissible based on a workload-specific performance signal, with a LinUCB contextual bandit that selects among admissible actions to minimize energy cost. We first describe the controller for LLM inference, where the performance constraint is a tail-latency SLO, then its adaptation for LLM training, where the operating range is determined by a pre-training calibration sweep.

3.1. Problem Formulation and Control Loop

Control proceeds in fixed windows of duration ΔT. At the end of each window t, the controller summarizes the window using probe latency statistics and GPU telemetry:

$$s_t = \big(\bar{L}_t, L_t^{95}, u_t, P_t, f_{\text{SM},t}\big),$$

where $\bar{L}_t$ and L_t^{95} are the probe mean and p95 end-to-end latencies, u_t is average SM utilization, P_t is average GPU power, and $f_{\text{SM},t}$, are the active SM and memory clock caps.

The controller actuates the SM frequency cap via five discrete actions on a 150 MHz ladder:

$$\mathcal{A}^{\text{SM}} = \{\text{SM}_{-2},\ \text{SM}_{-1},\ \text{HOLD},\ \text{SM}_{+1},\ \text{SM}_{+2}\},$$

corresponding to $\{-300, -150, 0, +150, +300\}$ MHz adjustments. Memory frequency is not controlled.

At the end of each window, the controller executes four steps:

1. **Sense.** Aggregate probe latencies and NVML telemetry to form summary s_t and context x_t.
2. **Zone & gate.** Classify the operating regime and derive the admissible action set $\mathcal{A}_{t+1}$.
3. **Score & update.** Compute energy E_t and reward r_t, then update the bandit with (x_{t-1}, a_t, r_t).
4. **Select & actuate.** Select $a_{t+1} \in \mathcal{A}_{t+1}$ and apply it as the SM clock cap for window $t{+}1$.

The objective is to minimize expected cumulative cost over all windows:

$$\min_{\pi}\ \mathbb{E}\left[\sum_{t=1}^{T} \text{cost}(E_t, \bar{L}_t, L_t^{95}, \text{SLO})\right].$$

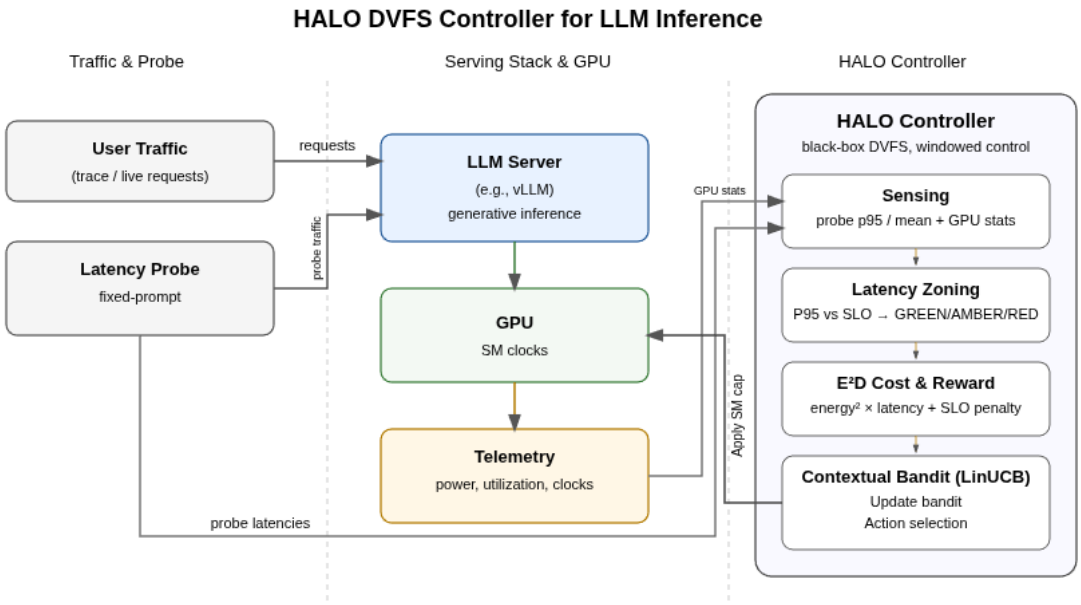

Figure 1: System architecture of HALO. The controller uses end-to-end probe latency and NVML telemetry only. Probe *p95* determines the latency zone and gates admissible actions; within that set, a contextual bandit selects an SM-frequency adjustment to reduce energy cost.

3.2. Latency Zoning and Action Gating

Each control window is classified into one of three zones based on probe p95 latency relative to the SLO:

$$z_t = \begin{cases} \textsc{Green}, & L_t^{95}/\mathrm{SLO} < 0.40, \\ \textsc{Amber}, & 0.40 \le L_t^{95}/\mathrm{SLO} \le 0.95, \\ \textsc{Red}, & L_t^{95}/\mathrm{SLO} > 0.95. \end{cases}$$

Each zone maps to a restricted action set. In GREEN, only downshifts are allowed; in AMBER, all actions are admissible; in RED, only upshifts are permitted. The thresholds are intentionally conservative: probe p95 carries 5–10% measurement noise, and tail-latency violations are operationally far more costly than modest energy inefficiency. On first entry into RED, the controller applies a deterministic emergency upshift before consulting the bandit, jumping several rungs up the frequency ladder to quickly restore headroom.

3.3. Contextual Bandit Policy

After zoning determines the admissible action set $\mathcal{A}_t$, the bandit selects which frequency adjustment to apply. Each window is treated as an independent contextual decision with immediate feedback: one action is held for a full window, and the resulting energy and latency statistics are observed. We use Linear Upper Confidence Bound (LinUCB) [7], which maintains a separate linear reward model per action and updates via closed-form regularized least squares.

Context vector. From window summary s_t, the controller builds a $d = 5$ context vector:

$$x_t = \left[1,\ \frac{\bar{L}_t^{\mathrm{probe}}}{\mathrm{SLO}},\ \frac{u_t}{100},\ \frac{P_t}{P_{\max}},\ \frac{f_{\mathrm{SM},t}}{f_{\mathrm{SM}}^{\max}}\right]^\top.$$

where 1 is a bias term, $\bar{L}_t^{\mathrm{probe}}/\mathrm{SLO}$ is normalized probe mean latency, $u_t/100$ is SM utilization, $P_t/P_{\max}$ is normalized GPU power, and $f_{\mathrm{SM},t}/f_{\mathrm{SM}}^{\max}$ is the normalized SM frequency cap. Probe mean latency is

clipped at $3\times$ after normalization. Probe p95 is reserved for zoning and the cost penalty; the bandit uses the smoother mean for learning.

Action selection. While the functional form of the reward is defined in Section 3.4, the reward realization r_t is unknown at action-selection time: E_t and L_t are determined by the stochastic workload and GPU response during window t, neither of which is knowable before the window executes. LinUCB learns the expected mapping $\mathbb{E}[r_t \mid x_t, a]$ under these non-stationary conditions. For each action $a \in \mathcal{A}^{\text{SM}}$, LinUCB models expected reward as $\mathrm{E}[r_t \mid x_t, a] = \theta_a^\top x_t$, where $\theta_a \in \mathbb{R}^d$ is learned online:

$$A_a = \lambda I + \sum_{\tau:a_\tau=a} x_\tau x_\tau^\top, \quad b_a = \sum_{\tau:a_\tau=a} x_\tau r_\tau, \quad \theta_a = A_a^{-1} b_a.$$

The selected action maximizes the UCB over the zone-filtered set:

$$a_t = \arg\max_{a\in\mathcal{A}_t} \left(\theta_a^\top x_t + \alpha(t)\sqrt{x_t^\top A_a^{-1} x_t} \right).$$

Exploration schedule. The exploration coefficient decays as:

$$\alpha(t) = \max\left(\alpha_{\min}, \frac{\alpha_0}{\sqrt{1 + t/H}} \right),$$

with $\alpha_0 = 1.0$, $\alpha_{\min} = 0.1$, $H = 40$ windows. Zone gating always takes precedence; $\alpha(t)$ only affects preferences among admissible actions.

3.4. Energy-Aware Cost and Reward Design

The bandit requires a scalar signal that reflects the energy-latency tradeoff while making operation near the SLO unattractive. Zoning provides the hard safety guardrails; the cost below shapes learning inside the region where exploration is allowed.

Energy-Squared–Delay ($\mathbf{E^2D}$). Window energy is computed by integrating NVML power samples:

$$E_t = \sum_{i=1}^{N_t} P_i \Delta t_i,$$

where P_i is instantaneous power and $\Delta t_i = 0.05$ s is the sampling interval. The baseline cost combines energy and mean probe latency:

$$\text{cost}_{\text{base}}(E_t, \bar{L}_t) = \left(\frac{E_t}{1000}\right)^2 \cdot \frac{\bar{L}_t}{1000}.$$

Squaring energy emphasizes high-power operating points and encourages downclocking when latency headroom is available. Tail safety is handled separately via zoning.

Soft SLO penalty. Let $L_0 = 0.65 \times$ SLO mark the start of the penalty region. For $L_t^{95} > L_0$:

$$\text{cost}(E_t, \bar{L}_t, L_t^{95}) = \text{cost}_{\text{base}} \cdot \left[1 + \lambda \left(\left(\frac{L_t^{95}}{L_0} \right)^\gamma - 1 \right) \right],$$

with $\lambda = 1.0$ and $\gamma = 2.0$. The penalty grows superlinearly as p95 approaches the SLO, steering the bandit away from low-headroom operating points before the controller enters the emergency regime.

Reward. The bandit receives the negated scaled cost:

$$r_t = -\frac{\text{cost}(E_t, \bar{L}_t, L_t^{95})}{\beta},$$

where β is a constant that keeps rewards numerically stable for LinUCB without changing the ordering induced by the cost.

3.5. Sensing and Measurement

HALO relies only on standard GPU telemetry and end-to-end latency from a lightweight probe stream, requiring no kernel timelines, KV-cache statistics, or serving-framework internals.

Latency. A continuous probe client sends fixed-length requests to the LLM endpoint independently of the main workload. Because the probe shares the same queue and GPU resources as regular requests, its end-to-end latency reflects execution time, batching, queueing, and interference effects. Per-window probe mean and p95 are aggregated and used as described in Sections 3.3 and 3.4.

Power and telemetry. GPU power is sampled via `nvmlDeviceGetPowerUsage` at 50 ms intervals. SM utilization u_t and the active SM clock cap $f_{\text{SM},t}$ are also read from NVML. All signals are aggregated per window to form the context vector x_t.

3.6. Extension to LLM Training

The inference controller assumes a continuously running serving stack with a latency SLO as the performance constraint. Training workloads have a different structure: the job runs for a fixed number of steps, where each step is one forward-backward pass and optimizer update over a batch of tokens, there is no per-request latency signal, and the optimization target is energy per training step. We adapt the same LinUCB framework to this setting by replacing the probe-based safety wrapper with a calibration sweep that identifies the energy-efficient operating range before training begins.

The energy valley. GPU energy consumption during training follows a curve as a function of SM frequency, shown in Figure 2. Since the job runs for a fixed number of steps, total energy can be extrapolated from per-step measurements at each frequency. At maximum frequency, power is high and the job completes quickly, but total energy is large. As frequency decreases, energy drops. Below a certain point, the job slows down enough that total energy rises again despite lower power. The efficient operating range sits on the descending portion of this curve, before the valley.

Calibration. At the start of training, the controller sweeps SM frequencies downward from maximum in fixed decrements. At each frequency, it runs the training job for a fixed number of steps and measures the average energy per step. It then extrapolates total training energy as:

$$\hat{E}(f) = \bar{e}(f) \times N,$$

where $\bar{e}(f)$ is the measured energy per step at frequency f and N is the total number of training steps. The sweep continues downward until $\hat{E}(f)$ first increases, indicating that the slowdown from lower frequency now outweighs the power reduction. This identifies the energy valley and sets the floor of the operating range. Since calibration runs during the first steps of training rather than as a separate phase, it adds no extra energy overhead; LinUCB takes over seamlessly once the sweep completes. LinUCB then operates within $[f_{\text{floor}}, f_{\max}]$ on a 45 MHz ladder, applying an analogous five-action set $\{-90, -45, 0, +45, +90\}$ MHz.

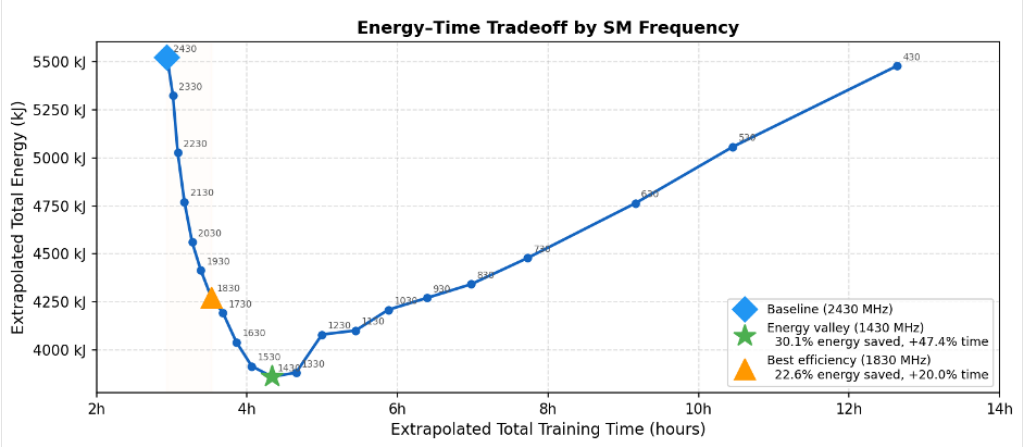

Figure 2: Energy-time tradeoff across SM frequencies for a representative training workload. Each point is a frequency setting, with the baseline at maximum frequency (top left). Moving left along the curve reduces energy until the valley, after which further frequency reduction adds training time faster than it saves energy. The calibration sweep identifies this valley as the floor of the LinUCB operating range.

Context vector. The training context replaces the latency signal with frequency and power efficiency features that help the bandit learn the shape of the energy valley:

$$x_t = \left[1,\ \tilde{f}_t,\ \tilde{f}_t^2,\ \frac{\tau_t}{\tau_{\max}},\ \frac{P_t}{P_{\max}}\right]^\top,$$

where $\tilde{f}_t = f_{\mathrm{SM},t}/f_{\mathrm{SM}}^{\max}$ is the normalized SM frequency, τ_t is throughput in steps per second normalized by the maximum observed throughput $\tau_{\max}$, and P_t is average GPU power normalized by the device power limit $P_{\max}$. The bias term enables the bandit to learn a frequency-independent baseline cost. The quadratic frequency feature lets LinUCB fit the curved energy-frequency relationship without a nonlinear model. Throughput and power together give the bandit the information needed to estimate energy per step at each operating point.

Cost and reward. Each training step corresponds to one forward-backward pass and optimizer update over a batch of tokens. The cost is energy per training step:

$$\mathrm{cost}_{\mathrm{train}}(E_t, n_t) = \frac{E_t}{n_t},$$

where E_t is the GPU energy consumed in window t and n_t is the number of training steps completed in that window. Normalizing by steps accounts for varying window durations and naturally balances energy savings against training time: a frequency that reduces power but slows training proportionally yields no improvement in the metric. The reward is $r_t = -\mathrm{cost}_{\mathrm{train}}/\beta$. The bandit learns which frequency minimizes energy per step within the calibrated range, converging online to the efficient operating point identified during calibration.

Integration. The only change required to a training script is adding a lightweight `ProgressCallback` that writes progress and step count to a shared file at each step. The controller reads this file to track training progress and runs as a separate process alongside the training job, requiring no modifications to the training framework itself.

4. Experimental Evaluation

4.1. Setup and Metrics

Setup. All experiments run on a single NVIDIA RTX PRO 6000 Blackwell GPU (96 GB VRAM, SM clock 180–2430 MHz), serving requests through vLLM [19]. We evaluate three models, Mistral-7B

Instruct v0.3, Mistral-Small-24B as the primary, and Qwen-2.5-32B-Instruct, to assess portability across model scales.

The workload replays the Azure LLM Inference Trace for the Coding workload [6]. The original trace reflects a large shared cluster, so we apply Bernoulli subsampling at $p{=}1/3$ to scale average throughput to single-GPU capacity; because each request is sampled independently, the temporal burstiness of the original is preserved. We extract the 12-hour daytime slice from 09:00 to 21:00, yielding 1,438 control windows of 30 s each. The *baseline* does no DVFS and operates at default SM Clock values (Baseline, no controller). HALO's primary configuration uses SLO = 10 s and offered-load speedup $1.0\times$. For Mistral-24B, we additionally ablate over speedup $\in \{0.75, 1.0, 1.5, 2.0\}\times$ and SLO $\in \{5, 7, 10\}$ s; the other models run at the primary configuration only.

Metrics. HALO compares probe p95 latency against the SLO threshold for zone gating, and uses probe mean latency for the cost term; all reported latency and throughput figures are computed from vLLM logs over user workload requests only, excluding probes. Total energy includes probe overhead, making reported energy savings conservative; we report total energy (kJ), average power (W), energy per request (J/req), mean latency (ms), throughput (req/s), and the offline E^2D objective. The controller and probe together consume less than 0.02% of total GPU energy, confirming negligible overhead. For the training evaluation, we additionally report energy per step (J/step), total training time, and average SM clock.

4.2. Single-GPU LLM Inference

Main result. Table 1 summarizes the primary result for Mistral-24B at $1.0\times$ offered load and SLO = 10 s. HALO reduces total GPU energy from 18,312 kJ to 11,096 kJ, a reduction of 39.4%, with average power falling proportionally from 423.8 W to 256.6 W and energy per request from 32.52 J to 19.73 J. Throughput is virtually unchanged at 13.01 versus 13.03 req/s.

Mean latency rises by 54.8%, from 1,006 ms to 1,556 ms, as HALO deliberately exploits the headroom below the 10 s probe SLO. Over the 12-hour run, the controller encounters only 7 brief RED-zone excursions and 4 probe-SLO violations. The mean E^2D objective improves by 43.2% from 164 to 93, confirming that HALO reaches a more energy-efficient operating point without exhausting the latency budget (Figure 3).

Table 1
Mistral-24B, sp=1.0×, SLO=10 s (1,438 windows, ≈12 h). Parentheses show change relative to Baseline.

	Energy			*Latency & throughput*		*Eff.*
	Total (kJ)	**Power (W)**	**E/req (J)**	**Lat (ms)**	**TP (req/s)**	E^2**D**
Baseline	18,312	423.8	32.52	1,006	13.03	164
HALO	11,096	256.6	19.73	1,556	13.01	93
Δ	−39.4%	−39.4%	−39.3%	+54.8%	−0.2%	−43.2%

Controller behaviour. Figure 4 shows HALO's internal signals over the 12-hour run. In the first dozen windows, the controller downshifts aggressively from the initial SM cap, taking advantage of the GREEN zone's full downshift allowance. It then settles into predominantly HOLD, which accounts for 76% of all windows, with the probe-based E^2D cost remaining stable throughout. Each RED-zone entry triggers an emergency ladder-up that briefly raises the SM cap before the controller returns to its operating point.

Sensitivity to offered load. Table 2 shows results as offered-load speedup rises from $0.75\times$ to $2.0\times$, with SLO fixed at 10 s. Energy savings narrow from 41.2% to 35.7% over this range. At higher loads, the latency margin shrinks and the controller needs to hold higher SM caps more often, leaving less

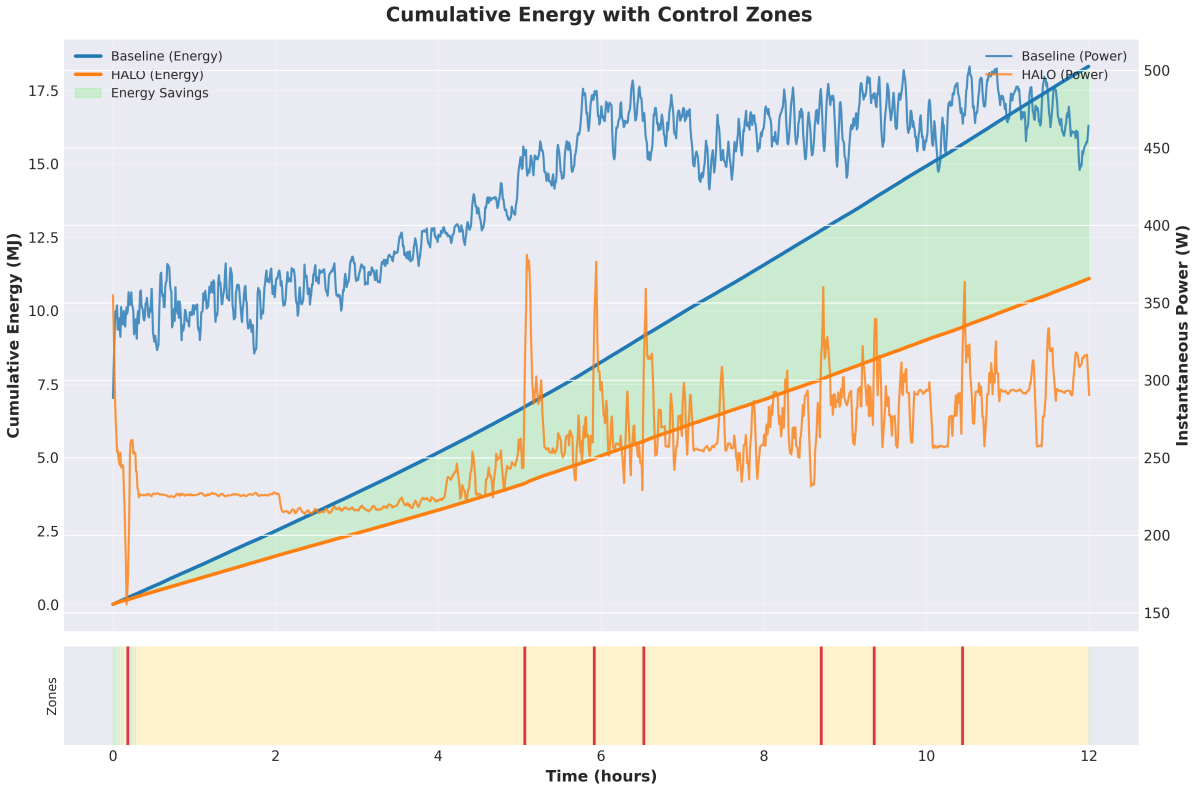

Figure 3: Cumulative GPU energy (solid lines) and instantaneous power (dashed lines) over 1,438 windows (≈12 h), Mistral-24B, sp=1.0×, SLO=10 s. Colored bands show the zone classification per window. The shaded region highlights the 39.4% energy gap between HALO and Baseline.

room for frequency reduction. Latency inflation follows the same trend, falling from +64.5% at 0.75× to +36.2% at 2.0× as the controller grows more conservative. Throughput stays within 0.3% of Baseline at all speedups. The E^2D objective improves consistently across the range, from −38.7% at 2.0× to −43.2% at 1.0×, indicating that HALO finds a more energy-efficient operating point regardless of load level.

Table 2
Sensitivity to offered load: Mistral-24B, SLO=10 s. Energy and E^2D Δ are relative to Baseline. Lat Δ is mean latency change.

Speedup	Baseline (kJ)	HALO (kJ)	Energy Δ	Latency Δ	E^2D Δ
0.75×	22,692	13,345	−41.2%	+64.5%	−42.3%
1.0×	18,312	11,096	−39.4%	+54.8%	−43.2%
1.5×	13,020	8,194	−37.1%	+47.1%	−39.2%
2.0×	10,355	6,661	−35.7%	+36.2%	−38.7%

Model portability. Table 3 shows results across three model scales at sp=1.0× and SLO=10 s. HALO requires no modification across models, achieving energy savings of 50.0% on Mistral-7B, 39.4% on Mistral-24B, and 21.3% on Qwen-32B, with throughput preserved within 0.2% in all cases.

Mistral-7B shows the largest latency increase at +168% (from 314 ms to 842 ms), with probe p95 latency remaining well within the 10 s SLO throughout the run (no RED-zone excursions or probe-SLO violations). The 7B model runs well within GPU capacity at 1.0× load, giving the controller ample room to downclock aggressively. Qwen-32B shows smaller savings because it operates closer to saturation under the same workload, leaving less frequency headroom.

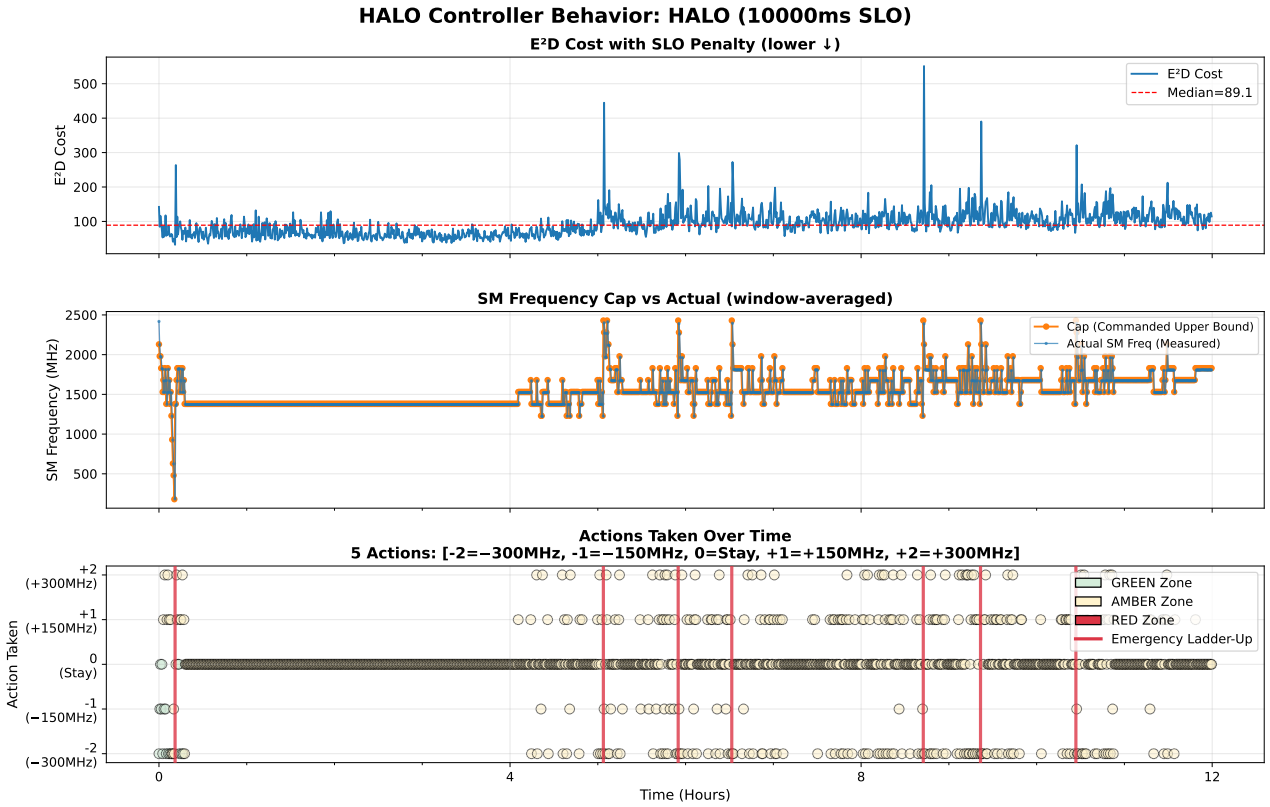

Figure 4: HALO controller behaviour (Mistral-24B, sp=1.0×, SLO=10 s, 12 h). Top to bottom: (i) probe-based E^2D cost per window; (ii) commanded SM-frequency cap; (iii) actions per window, colored by zone (GREEN/AMBER/RED).

Table 3
Model portability: sp=1.0×, SLO=10 s. All Δ values are HALO relative to Baseline. Lat Δ is mean latency change.

Model	Baseline (kJ)	HALO (kJ)	Energy Δ	Latency Δ	E^2D Δ
Mistral-7B	13,397	6,703	−50.0%	+168%	−33.1%
Mistral-24B	18,312	11,096	−39.4%	+54.8%	−43.2%
Qwen-32B	19,389	15,265	−21.3%	+19.0%	−18.7%

SLO sensitivity. Tightening the probe SLO from 10 s to 7 s to 5 s (Mistral-24B, sp=1.0×) reduces energy savings from 39.4% to 27.8% to 15.3%, with throughput staying within 0.3% throughout. Mean user latency falls accordingly, from 1,556 ms at the 10 s SLO to 1,273 ms at 7 s to 1,127 ms at 5 s. A tighter SLO leaves less latency margin, so the controller holds higher SM caps more often, making the SLO a direct knob for trading energy savings against latency conservatism.

4.3. Data-Parallel Inference

We evaluate HALO in a data-parallel configuration to assess horizontal scalability. The full Azure Coding trace (Mistral-24B, SLO=10 s) is distributed across two RTX PRO 6000 GPUs via an external vLLM router using round-robin scheduling, with one independent HALO instance per GPU and no inter-GPU coordination. System-level energy and throughput are aggregated across both GPUs.

Table 4 shows that HALO achieves 31.9% system-level energy savings at 1.0× load, narrowing to 15.0% at 1.5× as both GPUs approach saturation. Mean latency increases by 40.0% at 1.0× and 33.2% at 1.5×, while probe p95 latency remains within the 10 s SLO throughout. Each GPU independently adjusts

its SM frequency based on its own latency headroom with no cross-GPU coordination. Throughput stays within 2.4% of Baseline at both load levels.

Table 4
Data-parallel inference: two RTX PRO 6000 GPUs, round-robin routing, Mistral-24B, SLO=10 s. Base and HALO columns show system-level energy in kJ.

Speedup	Baseline (kJ)	HALO (kJ)	Energy Δ	Latency Δ	E^2D Δ
1.0×	35,386	24,103	−31.9%	+40.0%	−46.0%
1.5×	20,906	17,768	−15.0%	+33.2%	−31.7%

4.4. Ablation

To isolate the contributions of zone-gating and the LinUCB bandit, we compare four configurations at the primary operating point (Mistral-24B, sp=1.0×, SLO=10 s): always-max, HTC (Heuristic Threshold Controller), bandit-only without zone gating, and full HALO. HTC applies a deterministic rule using the same zone thresholds as HALO: in the Green zone it scales down two frequency steps, in Amber it holds, and in Red it scales up one step. HTC serves as the additional non-argmax baseline; bandit-only isolates the safety contribution of zone gating. Results are in Table 5.

Without zone gating, the bandit-only variant accumulates 16 SLO violations against 0–4 for all zone-gated configurations, confirming that zone gating is the primary safety mechanism. At sp=1.0×, HTC and HALO reach similar E^2D reductions of −46.6% and −43.2%: the system spends most windows in the Green zone, where the fixed two-step downshift already captures most available savings. At higher load the picture changes. At sp=2.0×, HTC's energy savings drop to −10.9% while HALO sustains −35.7%, with HALO recording 24 SLO violations against zero for HTC. With most windows now in Amber, HTC holds frequency and saves almost nothing; the bandit adapts as load grows.[1]

Table 5
Ablation at sp=1.0×, SLO=10 s (Mistral-24B, 1,438 windows). All Δ values are relative to always-max. Baseline SLO violations are 0 by design (max frequency; probe omitted from baseline run).

Configuration	Energy Δ	E^2D Δ	SLO Violations
Always-max (baseline)	0%	0%	0
HTC (zones, no bandit)	−41.4%	−46.6%	0
Bandit-only (no zones)	−41.1%	−42.5%	16
Full HALO	−39.4%	−43.2%	4

4.5. LLM Training

Setup. We evaluate the controller on Pythia-6.9B trained from random initialization on SlimPajama-627B-DC [20] for 1,000 steps with a sequence length of 2,048 tokens and an effective batch size of 65,536 tokens per step. All training experiments run on a single NVIDIA RTX PRO 6000 Blackwell GPU (SM clock 180–2,430 MHz). The baseline runs without the controller, operating at the default SM clock. The controller runs the calibration sweep at the start of training, stepping down in 150 MHz decrements and measuring 10 steps per frequency point. Calibration consumes 90 of the 1,000 training steps and identifies an operating floor of 1,380 MHz, after which LinUCB takes over within the range $[1{,}380, 2{,}430]$ MHz using a 90 s control window.

[1] A less aggressive GREEN rule for HTC (SM_{-1} instead of SM_{-2}) produced nearly identical results at sp=1.0× (−41.1% vs. −41.4% energy), confirming the comparison is not sensitive to this design choice.

Main result. Table 6 shows that the controller reduces total GPU energy by 19.4% (5,463.1 kJ to 4,404.5 kJ), with average power dropping from 594.1 W to 401.9 W. The reported total includes a one-time calibration cost of 282.7 kJ. The DVFS control phase alone consumes 4,121.8 kJ, a 24.6% reduction over baseline, so the calibration overhead amortizes on longer runs. The average SM clock settles at 1,871 MHz, below the 2,430 MHz baseline. Training time increases by 18.2% (2.56 h to 3.02 h), reflecting the throughput cost of lower-frequency operation. For comparison, we also report a fixed frequency run at the calibrated energy-valley floor (f_{floor} static, 1,380 MHz): this reaches lower total energy (−29.1% vs. baseline) but at a 60.9% time penalty, more than triple the controller's. Both runs complete 1,000 steps with equivalent loss trajectories, confirming that DVFS does not affect training convergence.

Table 6
Pythia-6.9B pretraining on SlimPajama, 1,000 steps, single GPU. Controller and f_{floor} static totals each include a 282.7 kJ one-time calibration cost, and total time includes the calibration sweep. The f_{floor} static row is extrapolated from the calibration sweep at 1,380 MHz. Δ is change relative to the indicated reference.

	Energy					*Performance*	
	Total (kJ)	**Cal. (kJ)**	**DVFS (kJ)**	**Power (W)**	**E/step (J)**	**Time (h)**	**SM (MHz)**
Baseline	5,463.1	–	–	594.1	5,463.1	2.56	2,430
f_{floor} static	3,874.3	282.7	3,591.6	261.5	3,874.3	4.12	1,380
Controller	4,404.5	282.7	4,121.8	401.9	4,404.5	3.02	1,871
Δ (f_{floor} vs base)	−29.1%	–	–	−56.0%	−29.1%	+60.9%	–
Δ (Controller vs base)	−19.4%	–	–	−32.4%	−19.4%	+18.2%	–

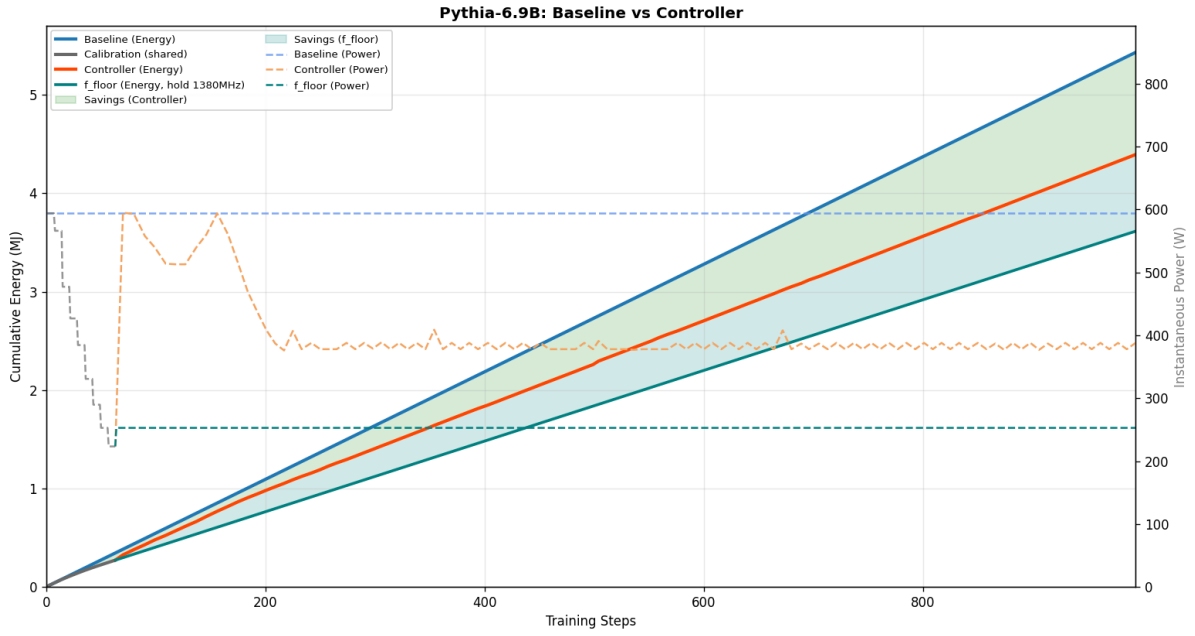

Figure 5: Pythia-6.9B pretraining: baseline vs controller over 1,000 steps. Cumulative GPU energy (solid lines) and instantaneous power (dashed lines). During the first training steps, the controller sweeps frequencies downward during calibration, visible as the declining power in the controller curve; LinUCB takes over once the energy valley is identified.

5. Conclusion

HALO demonstrates that black-box, safety-constrained DVFS control can deliver substantial energy reductions across both LLM inference and training without any modifications to the serving framework

or model. For inference, a hierarchical zone-gating mechanism combined with a contextual bandit reduces GPU energy by 39.4% on the primary Mistral-24B model and up to 50.0% across model scales on the Azure LLM Inference Trace, while maintaining tail latency under the SLO.

For pretraining, the same controller applied without modification reduces energy by 19.4% on Pythia-6.9B with convergence unaffected, showing that the approach generalizes beyond inference to training workloads.

These results suggest that simple structural safety constraints, rather than complex model-specific instrumentation, are sufficient for effective online GPU power management.

Limitations and future work. The controller is limited to SM-clock actuation on the current hardware; memory clock control, unavailable here, could yield additional savings. The data-parallel inference evaluation uses two independent HALO instances with round-robin routing; we plan to extend this to tensor- and pipeline-parallel deployments, where inter-stage dependencies require coordinated frequency decisions across GPUs. For training, the context vector uses frequency, power, and throughput as features; richer signals such as gradient norms or loss trajectory could provide more predictive information for the bandit and are worth exploring. Extending the controller to multi-GPU data-parallel and pipeline-parallel settings is a natural next step, as single-GPU training is increasingly rare for models at production scale; this requires coordinated DVFS across ranks while preserving the calibration-based frequency range identification.

Acknowledgments

This work was part-funded within the research project ESCADE (grant number: 01MN23004A, www.escade-project.de), funded by the German Federal Ministry of Research, Technology and Space (BMFTR), supported by the DLR project management agency.

Declaration on Generative AI

During the preparation of this work, the author(s) used Claude (Anthropic) and Gemini (Google) in order to: Improve writing style and Grammar and spelling check. After using these tool(s)/service(s), the author(s) reviewed and edited the content as needed and take(s) full responsibility for the publication's content.

References

[1] S. Poddar, P. Koley, J. Misra, N. Ganguly, S. Ghosh, Towards sustainable NLP: Insights from benchmarking inference energy in large language models, in: L. Chiruzzo, A. Ritter, L. Wang (Eds.), Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers), Association for Computational Linguistics, Albuquerque, New Mexico, 2025, pp. 12688–12704. URL: https://aclanthology.org/2025.naacl-long.632/. doi:10.18653/v1/2025.naacl-long.632.

[2] P. Patel, E. Choukse, C. Zhang, I. n. Goiri, B. Warrier, N. Mahalingam, R. Bianchini, Characterizing power management opportunities for llms in the cloud, in: Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3, ASPLOS '24, Association for Computing Machinery, New York, NY, USA, 2024, p. 207–222. URL: https://doi.org/10.1145/3620666.3651329. doi:10.1145/3620666.3651329.

[3] P. J. Maliakel, S. Ilager, I. Brandic, Investigating energy efficiency and performance trade-offs in llm inference across tasks and dvfs settings, 2025. URL: https://arxiv.org/abs/2501.08219. arXiv:2501.08219.

[4] N. Jegham, M. Abdelatti, C. Y. Koh, L. Elmoubarki, A. Hendawi, How hungry is ai? benchmarking energy, water, and carbon footprint of llm inference, 2025. URL: https://arxiv.org/abs/2505.09598. arXiv:2505.09598.

[5] Z. Tang, Y. Wang, Q. Wang, X. Chu, The impact of GPU DVFS on the energy and performance of deep learning: An empirical study, in: Proceedings of the 10th ACM International Conference on Future Energy Systems (e-Energy 2019), ACM, 2019, pp. 315–325.

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

[7] L. Li, W. Chu, J. Langford, R. E. Schapire, A contextual-bandit approach to personalized news article recommendation, in: Proceedings of the 19th International Conference on World Wide Web (WWW 2010), ACM, 2010, pp. 661–670.

[8] A. K. Kakolyris, D. Masouros, P. Vavaroutsos, S. Xydis, D. Soudris, throttll'em: Predictive gpu throttling for energy efficient llm inference serving, in: 2025 IEEE International Symposium on High Performance Computer Architecture (HPCA), 2025, pp. 1363–1378. doi:10.1109/HPCA61900.2025.00103.

[9] Z. Ye, K. Zhang, G. Tang, Agft: An adaptive gpu frequency tuner for real-time llm inference optimization, 2025. URL: https://arxiv.org/abs/2508.01744. arXiv:2508.01744.

[10] Q. Liu, D. Huang, M. Zapater, D. Atienza, Greenllm: Slo-aware dynamic frequency scaling for energy-efficient llm serving, arXiv preprint arXiv:2508.16449 (2025).

[11] O. Basit, Y. Liu, Z. J. Kong, Y. C. Hu, DualScale: Energy-efficient disaggregated LLM serving via phase-aware placement and DVFS, 2026. arXiv:2602.18755.

[12] J. McDonald, B. Li, N. Frey, D. Tiwari, V. Gadepally, S. Samsi, Great power, great responsibility: Recommendations for reducing energy for training language models, in: Findings of the Association for Computational Linguistics: NAACL 2022, Association for Computational Linguistics, Seattle, United States, 2022, pp. 1962–1970. URL: https://aclanthology.org/2022.findings-naacl.151/. doi:10.18653/v1/2022.findings-naacl.151.

[13] J. You, J. Chung, M. Chowdhury, Zeus: Understanding and optimizing GPU energy consumption of DNN training, in: 20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23), USENIX Association, Boston, MA, 2023, pp. 119–139. URL: https://www.usenix.org/conference/nsdi23/presentation/you.

[14] J.-W. Chung, Y. Gu, I. Jang, L. Meng, N. Bansal, M. Chowdhury, Reducing energy bloat in large model training, in: Proceedings of the ACM SIGOPS 30th Symposium on Operating Systems Principles (SOSP '24), ACM, 2024. doi:10.1145/3694715.3695970.

[15] F. Wang, W. Zhang, S. Lai, M. Hao, Z. Wang, Dynamic GPU energy optimization for machine learning training workloads, IEEE Transactions on Parallel and Distributed Systems 33 (2022) 2943–2954. doi:10.1109/TPDS.2021.3137867.

[16] Y. Wang, M. Hao, H. He, W. Zhang, Q. Tang, X. Sun, Z. Wang, DRLCAP: Runtime GPU frequency capping with deep reinforcement learning, IEEE Transactions on Sustainable Computing 9 (2024) 712–726. doi:10.1109/TSUSC.2024.3362697.

[17] V. Sachidananda, A. Sivaraman, Erlang: Application-aware autoscaling for cloud microservices, in: Proceedings of the Nineteenth European Conference on Computer Systems (EuroSys '24), ACM, 2024. doi:10.1145/3627703.3650084.

[18] A. Kazerouni, M. Ghavamzadeh, Y. Abbasi-Yadkori, B. Van Roy, Conservative contextual linear bandits, in: Advances in Neural Information Processing Systems 30 (NeurIPS 2017), 2017, pp. 3913–3922.

[19] W. Kwon, Z. Li, S. Zhuang, Y. Sheng, L. Zheng, C. H. Yu, J. Gonzalez, H. Zhang, I. Stoica, Efficient memory management for large language model serving with pagedattention, in: Proceedings of the 29th Symposium on Operating Systems Principles, SOSP '23, Association for Computing Machinery, New York, NY, USA, 2023, p. 611–626. URL: https://doi.org/10.1145/3600006.3613165. doi:10.1145/3600006.3613165.

[20] D. Soboleva, F. Al-Khateeb, R. Myers, J. R. Steeves, J. Hestness, N. Dey, SlimPajama: A 627b

token cleaned and deduplicated version of RedPajama, https://huggingface.co/datasets/cerebras/SlimPajama-627B, 2023.