LEARNING-BASED DYNAMIC POWER MANAGEMENT IN LLM SERVING WITH LATENCY SLOS

Anonymous authors
Paper under double-blind review

ABSTRACT

The request patterns of real-time LLM services are bursty and vary substantially over time, yet production GPUs are commonly held at their maximum frequency to protect tail-latency targets, a conservative practice that wastes a considerable amount of energy whenever the service has latency headroom. Lowering GPU energy under a tail-latency service-level objective (SLO) is difficult because frequency scaling affects latency in a nonlinear way that depends on queueing and execution dynamics hidden from the operator. We present HALO (Hierarchical Adaptive Latency-Oriented DVFS), which treats the serving system as a black box and reduces GPU energy consumption using only two externally observable signals, the GPU telemetry provided by the standard driver interface and the end-to-end latency of a lightweight probe stream. At the end of every control window, HALO compares the probe p95 latency with the SLO to place the system in one of three coarse operating regimes, and the regime fixes the set of admissible SM-frequency actions. Within this admissible set, we use a LinUCB contextual bandit method to select among small discrete frequency steps using window-level telemetry as its context, so the controller adapts online to non-stationary load without offline profiling, instrumentation of the model, or changes to the serving framework. The bandit learns from an energy-squared-delay cost with a soft penalty on the probe tail, which discourages operation close to the SLO and keeps the learning signal stable despite high latency variance. In a replay of an Azure LLM inference trace against vLLM on an NVIDIA A800, HALO lowers total GPU energy by 37.5% in the default setting and by 14.5% to 40.8% across offered loads, probe SLOs, and three served models, with unchanged throughput and probe p95 below the SLO in 99% of the control windows.

1 INTRODUCTION

Services built on foundation models have become a central part of the computing infrastructure, and serving them interactively places a continuous load on accelerators (Poddar et al., 2025). The computational demand of LLM inference varies substantially, since prompt lengths, decoding behavior, and the number of concurrent users all change over time. Most data-center deployments nevertheless operate their GPUs at or close to the highest supported frequency (Patel et al., 2024; Patterson et al., 2022). This "always-max" policy keeps latency management simple, but it wastes a considerable amount of energy whenever the load is light or moderate, an effect that earlier measurement studies of GPU dynamic voltage and frequency scaling (DVFS) have also documented (Mei et al., 2017; Tang et al., 2019).

The underlying motivating question is: how much GPU energy can be saved during LLM inference without compromising the latency guarantees given to users? Answering this question is difficult because the effect of the clock frequency on latency and energy is nonlinear and depends on the workload (Han et al., 2025). The nonlinearity originates in the interaction between processing speed and queueing. When requests arrive sparsely, most of them are served immediately, and a moderately slower GPU changes tail latency very little. As demand grows, requests begin to accumulate, and latency becomes dominated by waiting rather than by computation. In this regime, even a small reduction in processing speed can increase tail latency substantially. Frequency scaling is safe under light load and risky close to the latency target (Figure 1), so a controller must distinguish the two situations online. It must also work with the signals that production serving stacks expose. These

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

are usually limited to end-to-end latency, utilization, and power, since kernel timings, batching decisions, and model-specific throughput counters are either unavailable or tied to a particular engine version. Black-box control is therefore a necessary option for real deployments (Maliakel et al., 2025; Stojkovic et al., 2025).

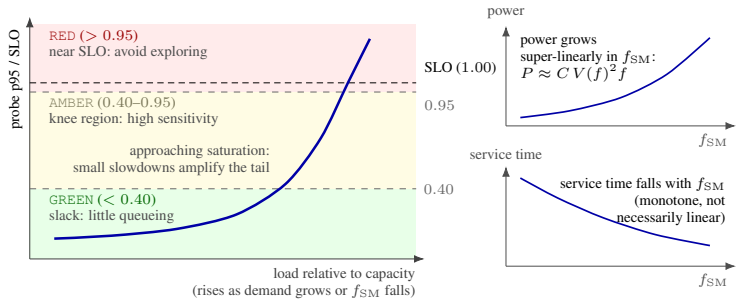

Figure 1: Conceptual model of frequency control in LLM serving. DVFS changes the effective service rate, and tail latency rises sharply near capacity. The main panel sketches the probe p95, normalized by the SLO, against the load relative to capacity, with the latency zones of Section 4.1 and the SLO marked, and the insets show the qualitative power and service-time trends with frequency.

Existing work addresses individual aspects of this problem, but no prior approach covers it completely. Classical DVFS controllers depend on hand-tuned rules, analytical performance models, or profiles collected offline (Mei et al., 2017; Tang et al., 2019). Such designs work well for steady HPC jobs or CNN inference, but they are poorly suited to generative inference, in which the length of requests, the speed of token generation, and the amount of queueing all vary widely over time (Li et al., 2023). Widely-used LLM serving systems like vLLM and SGLang improve batching, scheduling, and the GPU memory management (Li et al., 2023; Yu et al., 2022), but they neither expose nor control the frequency setting of the GPU. Learning-based controllers that rely on reinforcement learning or predictive control have also been proposed, but they usually require intrusive instrumentation, kernel-level telemetry, or carefully prepared training workloads, as in AGFT (Ye et al., 2025) and in recent work on predictive throttling (Kakolyris et al., 2025). Such requirements are difficult to satisfy in large multi-tenant deployments. Appendix A reviews the related work in full.

The structure of the problem is well matched to a (non-stationary) contextual bandit: decisions are made repeatedly under partial observability, feedback becomes available when a control window closes, and the workload is bursty and non-stationary (Ye et al., 2025). This motivates our design, a low-overhead controller in which a simple safety wrapper surrounds the bandit. The controller reads two signals only, the standard NVIDIA Management Library (NVML) telemetry of the GPU and the end-to-end latency of a lightweight probe stream that is issued alongside the user requests. The tail-latency SLO is enforced on the probe, whereas the latencies of the workload itself are used only for evaluation. The design requires no knowledge of the model, no instrumentation of kernels, and no change to the serving framework, and it runs on commodity NVIDIA GPUs. Our **contributions** are summarized as follows:

(i) We formulate GPU frequency control in LLM serving as a contextual bandit problem with safety constraints under partial observability, in which the only latency signal is the end-to-end latency of a lightweight probe stream.

(ii) We present HALO, a low-overhead black-box controller that combines three-zone action gating with online LinUCB learning (Li et al., 2010). At the end of every control window, the ratio of the probe p95 latency to the SLO assigns the system to one of three regimes, `GREEN` (ample latency headroom), `AMBER` (around the knee of the load-latency curve, Figure 1), or `RED` (at or past the SLO), and the regime gates the five discrete SM-frequency moves: downshifts in `GREEN`, moves in both directions in `AMBER`, and upshifts only in `RED`, with a fixed emergency upshift on the first entry into `RED`. Within the admissible set, learner selects the next move from the window-level context.

(iii) We propose an SLO-aware E^2D objective, an energy-squared-delay cost extended by a soft penalty that grows superlinearly as the probe tail approaches the SLO, which steers the learner away from risky operating points.

(iv) We evaluate HALO by replaying an Azure LLM inference trace against vLLM (Kwon et al., 2023) on an NVIDIA A800. In default, HALO lowers total GPU energy by 37.5% with unchanged throughput, while the probe p95 stays below the 10 s SLO in 99% of the control windows; across four offered loads, three probe SLOs, and three models, the reduction ranges from 14.5% to 40.8%, shrinking near saturation, where the queueing model of Section 2 predicts that latency headroom disappears, and SLO violations added over the always-max baseline are rare and short.

2 Design Intuition

This section describes how frequency scaling interacts with queueing in an LLM serving stack, and this interaction motivates both the design of HALO and the choice of the probe tail latency as its safety signal. To a first approximation, the serving stack behaves as a single-server queue whose service rate depends on the SM frequency (Gujarati et al., 2020; Nath & Tullsen, 2015). Batching and pipelining change the constants of this abstraction but not the effect that matters for DVFS, which is that a lower frequency reduces the effective service capacity and moves the operating point toward the knee of the load-latency curve. The abstraction builds on four well-documented properties of LLM inference on GPUs, namely that request arrivals are bursty and non-stationary (Patel et al., 2024), that service time decreases monotonically with frequency (Tang et al., 2019; Han et al., 2025), that power grows faster than linearly with frequency (Choi & Vuduc, 2016), and that tail latency near saturation is dominated by queueing (Lo et al., 2015; Lawniczak & Distler, 2024). In the rest of the paper, the term "tail latency" refers to the p95 of the end-to-end latency of the probe within a control window. Figure 1 summarizes the resulting behavior. With ample slack, a downshift lengthens the service time but barely changes the end-to-end latency, and because power rises steeply with frequency, it saves a substantial amount of energy (Vamanan et al., 2015). Near the knee, the same downshift is amplified by queueing and can push the tail past the SLO. The end-to-end latency of the probe exposes where on this curve the system sits, including the queueing that GPU telemetry alone does not reveal. The appropriate action therefore depends on the operating point, and the ratio of the probe p95 to the SLO is a sufficient signal for locating it. With ample headroom the controller can exploit the slack, around the knee it has to move carefully, and at the SLO it has to recover. The next two sections formalize the objective and build the controller around this signal.

3 Problem Formulation

Windowed control and actions. We consider a controller that adjusts the SM-frequency cap of the GPU once per control window of fixed length ΔT. When window t ends, the controller summarizes the probe latency statistics and the GPU telemetry of that window in the state $s_t = (\bar{L}_t, L_t^{95}, u_t, P_t, f_{\mathrm{SM},t}, f_{\mathrm{MEM},t})$, where $\bar{L}_t$ and L_t^{95} are the mean and the p95 of the end-to-end latency of the probe, u_t is the average SM utilization, P_t is the average GPU power, and $f_{\mathrm{SM},t}$ and $f_{\mathrm{MEM},t}$ are the caps in force on the SM and memory clocks. Since probe latency is measured end to end, it already reflects execution, batching, queueing, and interference as a client would experience them, and workload latencies are not part of s_t. Adjusting the cap is effective because the dynamic switching power of CMOS logic follows $P_{\mathrm{dyn}} \approx CV^2 f$ (Harris & Harris, 2007), and since higher GPU frequencies generally require a higher supply voltage, power grows faster than linearly with f (Liu et al., 2025). Lowering the cap therefore saves energy more than proportionally, with the voltage following the frequency implicitly. To avoid near-duplicate settings, the cap is confined to a coarse ladder of allowed values, and one control step moves it by at most two rungs:

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

On the GPU used in our experiments, the ladder runs from 240 to 1,410 MHz in steps of 45 MHz, so the five actions change the cap by -90, -45, 0, $+45$, and $+90$ MHz. Moves that would leave the ladder are removed, and the latency zone selects the admissible subset $\mathcal{A}_t \subseteq \mathcal{A}^{\mathrm{SM}}$ of each window (Section 4.1). The memory clock is left unmanaged.

Objective. The controller aims to reduce the long-run GPU energy over successive windows while tail latency stays within an operator-specified SLO. A good policy uses the latency slack of low-load intervals to save energy and reacts conservatively when p95 comes close to the SLO. Rather than imposing the SLO as a hard constraint, we treat violations as far more costly than energy

inefficiency (Zhou et al., 2020). This reflects how user-facing services are operated in practice, since a modest amount of wasted energy is preferable to a tail-latency incident. Writing E_t for the GPU energy consumed in window t, the controller seeks a policy π that minimizes the expected cumulative cost:

$$\min_{\pi} \mathbb{E}\left[\sum_{t=1}^{T} \mathrm{cost}\big(E_t, \bar{L}_t, L_t^{95}, \mathrm{SLO}\big)\right], \tag{2}$$

where the expectation is taken over the workload and over the action choices of the policy. No transition model is assumed, and each window is treated as a myopic decision that depends only on the observed context. The cost function (Section 4.3) trades energy against latency and sharply penalizes operation close to the SLO, so that the learning signal remains stable when the load is noisy and non-stationary.

4 HALO: Latency-Gated Contextual Bandit Control

The design of HALO builds on two observations from Section 2. Close to saturation, tail latency reacts sharply to small slowdowns, and under light load a lower frequency saves energy at almost no latency cost. HALO turns these observations into a black-box controller with two components, a safety wrapper and an online learner (Figure 2). The wrapper uses the probe p95 to decide which frequency moves are permitted, and the learner chooses among the permitted moves so as to lower the energy cost. Frequency control in LLM serving is a sequential decision problem in which the controller observes only part of the state. Signals that would help in principle, such as the queue length, batching decisions, timings of individual kernels, and KV-cache occupancy, are either unavailable or tied to one serving engine and its configuration.

Therefore, HALO uses nothing beyond the end-to-end latency of the probe and the NVML telemetry, so that the same controller can be applied to different serving systems without intrusive instrumentation. The setting also favors a contextual bandit over full reinforcement learning. Each action is a small step whose effect is visible within one window, workload conditions change on a similar timescale, and the safety wrapper already bounds exploration, so a bandit provides online adaptation at low overhead without a model of long-horizon dynamics.

Control loop. An action a_t (an adjustment of the SM cap) is selected when window $t-1$ closes and remains in force for the whole of window t, during which probe requests and NVML samples are collected. When window t ends, the controller (1) aggregates the completed window into the summary s_t and the context x_t, (2) computes the latency zone z_t from the probe p95 and derives from it the admissible set $\mathcal{A}_{t+1}$ of window $t+1$, (3) computes the window energy E_t and the reward r_t and updates the bandit model of the action that was in force with (x_{t-1}, a_t, r_t), and (4) selects $a_{t+1} \in \mathcal{A}_{t+1}$ from x_t and applies the corresponding cap for window $t+1$. The wrapper protects tail latency by restricting which actions are considered at all, and the learner optimizes energy only inside this admissible set.

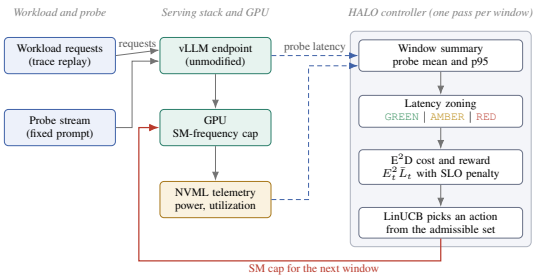

Figure 2: HALO architecture. The controller reads only the probe latency and the NVML telemetry, and it produces one SM-cap decision per control window. The serving engine is not modified.

4.1 Latency zones and action gating

HALO keeps tail-latency violations rare by restricting exploration with a single conservative safety signal, the end-to-end p95 latency of the probe. Every control window falls into one of three zones, defined by the ratio of the probe p95 to the SLO, and the zone fixes the admissible SM-frequency actions. The probe p95 is used for safety only, in the zoning and in the penalty of Section 4.3, whereas the smoother probe mean enters the energy-delay term. With L_t^{95} the probe p95 of window

t and SLO the tail-latency target of the operator, the zone is:

$$z_t = \begin{cases} \texttt{GREEN}, & L_t^{95}/\mathrm{SLO} < 0.40, \\ \texttt{AMBER}, & 0.40 \le L_t^{95}/\mathrm{SLO} \le 0.95, \\ \texttt{RED}, & L_t^{95}/\mathrm{SLO} > 0.95. \end{cases} \tag{3}$$

In GREEN (below 40% of the SLO), energy-saving downshifts are allowed, $\mathcal{A}_t = \{\mathrm{SM}_{-2}, \mathrm{SM}_{-1}, \mathrm{HOLD}\}$. In AMBER (40% to 95%), exploration in both directions is allowed, $\mathcal{A}_t = \{\mathrm{SM}_{-2}, \mathrm{SM}_{-1}, \mathrm{HOLD}, \mathrm{SM}_{+1}, \mathrm{SM}_{+2}\}$. In RED (above 95%), only recovery upshifts are allowed, $\mathcal{A}_t = \{\mathrm{SM}_{+1}, \mathrm{SM}_{+2}\}$. If a rung limit removes every admissible move (e.g., RED at the top of the ladder), the controller holds the current cap.

The thresholds are coarse by design, because the probe p95 fluctuates by 5 to 10% from one window to the next and narrow margins would be brittle. A tail-latency violation is also far more costly for an operator than a modest amount of wasted energy, so the zone boundaries are placed well below the nominal SLO. The thresholds were kept fixed across every trace, model, and SLO in the evaluation. When a GREEN or AMBER window is followed by a RED one, HALO bypasses the bandit and applies a fixed emergency upshift of eight rungs (360 MHz), which restores headroom quickly. Later RED windows use the ordinary set $\{\mathrm{SM}_{+1}, \mathrm{SM}_{+2}\}$. The emergency move does not update any bandit model, so an unusual burst does not distort what the learner has found about the ordinary actions.

4.2 Contextual bandit policy

Once the wrapper has fixed $\mathcal{A}_t$, the remaining decision is which small cap adjustment reduces energy without giving up latency headroom. Feedback is local to a window, since one action is held for the whole window and the energy and latency statistics of that window are observed when it ends. Arrival rates and generation lengths drift, so a frequency setting chosen offline would soon become outdated. Each window is therefore treated as an independent contextual decision with immediate feedback. The policy is LinUCB (Li et al., 2010), which maintains a separate linear reward model with ridge regularization for each action and selects actions optimistically under uncertainty, with closed-form updates of negligible cost.

Context vector. From s_t, the controller forms a $d = 5$ context:

$$x_t = \left[\frac{\bar{L}_t}{\mathrm{SLO}},\ \frac{u_t}{100},\ \frac{P_t}{P_{\max}},\ \frac{f_{\mathrm{SM},t}}{f_{\mathrm{SM}}^{\max}},\ \frac{f_{\mathrm{MEM},t}}{f_{\mathrm{MEM}}^{\max}} \right]^\top, \tag{4}$$

where $\bar{L}_t$ is the mean probe latency of window t and u_t, P_t, $f_{\mathrm{SM},t}$, and $f_{\mathrm{MEM},t}$ are the window averages of utilization, power, and the two active clock caps. The normalizers $P_{\max}$, $f_{\mathrm{SM}}^{\max}$, and $f_{\mathrm{MEM}}^{\max}$ are fixed device constants (the power limit and the highest supported caps), and the latency feature is clipped at $3\times$ so that outliers have bounded influence. The probe p95 is not part of x_t, since it is reserved for the safety layer (Section 4.1).

Reward model and action selection. LinUCB represents the expected reward of every action $a \in \mathcal{A}^{\mathrm{SM}}$ as $\mathbb{E}[r_t \mid x_t, a] = \theta_a^\top x_t$, with $\theta_a \in \mathbb{R}^d$ fitted online by ridge regression, and the next action is chosen within zone-filtered set ($A_a = \lambda I + \sum_{\tau:a_\tau=a} x_\tau x_\tau^\top, b_a = \sum_{\tau:a_\tau=a} x_\tau r_\tau, \hat{\theta}_a = A_a^{-1} b_a,$):

$$a_{t+1} = \arg\max_{a \in \mathcal{A}_{t+1}} \left(\hat{\theta}_a^\top x_t + \alpha(t) \sqrt{x_t^\top A_a^{-1} x_t} \right) \tag{5}$$

The linear model is a local approximation over the region the gate leaves open, in which every action moves the cap by at most 90 MHz. Only windows controlled by an ordinary action update equation 5, and the context stored with an action is the one that was available when the action was chosen, so that learning is aligned with the window the action actually affected. The exploration coefficient decays as $\alpha(t) = \max\left(\alpha_{\min}, \alpha_0/\sqrt{1+t}\right)$ with $\alpha_0 = 1.0$ and $\alpha_{\min} = 0.1$. Exploration remains subject to the gate, since $\alpha(t)$ affects only the preference among admissible actions.

4.3 Cost function and reward

We propose an SLO-aware cost that gives the learner a single scalar signal for the trade-off between energy and latency and makes operating points with little remaining tail-latency slack unattractive. The zones enforce the hard constraints by restricting the action set, and the cost defined below shapes

the preferences of the learner within the region in which exploration is allowed. For each window, the GPU energy E_t is the time integral of the NVML power samples, which are taken every 50 ms (Section 4.4). With $\bar{L}_t$ the mean probe latency of the same window, we take the energy-squared-delay score as follows:

$$\mathrm{cost}_{\mathrm{base}}(E_t, \bar{L}_t) = E_t^2 \cdot \bar{L}_t, \tag{6}$$

as the base of the cost, in units of $(\mathrm{kJ})^2\,\mathrm{s}$. Compared with the usual energy-delay product (Ye et al., 2025), squaring the energy gives high-energy operating points more weight and encourages the controller to downclock whenever slack is available.

The second ingredient, and the one that makes the cost SLO-aware, is a soft penalty on the probe p95 that we add on top of the base score. Let $L_0 = 0.65 \times \mathrm{SLO}$ be the point at which a conservative penalty region begins. For $L_t^{95} \le L_0$ the cost is unchanged, and for $L_t^{95} > L_0$

$$\mathrm{cost}(E_t, \bar{L}_t, L_t^{95}, \mathrm{SLO}) = \mathrm{cost}_{\mathrm{base}}(E_t, \bar{L}_t) \cdot \Big(1 + \lambda\Big[\big(L_t^{95}/L_0\big)^{\gamma} - 1\Big]\Big), \tag{7}$$

with $\lambda = 1.0$ and $\gamma = 2.0$. The penalty grows superlinearly with the probe p95, so it stays small for the noise-level excursions just above L_0 and becomes large near the SLO, where the tail is most sensitive. The penalty steers the learner away from operating points with little remaining slack, well before the controller reaches the emergency regime. The reward handed to the bandit is the negated cost, $r_t = -\mathrm{cost}(E_t, \bar{L}_t, L_t^{95}, \mathrm{SLO})/\beta$, where the constant β (100 in our experiments) keeps the rewards within a numerically stable range for the learner and preserves the ordering of actions induced by the cost.

4.4 Sensing and measurement

HALO depends on two inputs, standard GPU telemetry and the end-to-end latency of a lightweight probe stream. A probe client sends requests with the same prompt and a fixed, short generation length (one input token and 100 output tokens in our implementation) to the LLM endpoint one at a time, independently of the main workload, so the probe adds little load. Because the probe passes through the same endpoint and competes for the same GPU as the user requests, its latency combines execution, batching, queueing, and interference, and it remains meaningful across runtimes and configurations in which engine-specific counters are not available.

The per-window aggregates of the probe latencies play two roles: the mean is the smooth signal in the context vector and in the energy-delay term, and the p95 is used by the safety layer only. Workload latencies are logged for evaluation only and are never observed by the controller. GPU power is sampled through NVML every 50 ms and integrated into the window energy E_t, and NVML also supplies the utilization and the clock caps in force (Appendix B).

5 Experimental Evaluation

This section empirically examines whether HALO lowers the GPU energy of an LLM service under bursty, production-derived load while throughput is preserved and the tail-latency budget of the operator is respected. We conduct the evaluation based on four questions. First, how much energy does HALO save in the default setting, and at what cost in latency and SLO violations? Second, where do the savings come from and how is the latency budget of the SLO spent, in terms of the zones, operating regimes, and frequency choices of the controller? Third, does the probe tail latency, the only latency signal the controller sees, track the latency that users experience? Fourth, how sensitive are the results to the offered load, the probe SLO, and the served model?

5.1 Experimental setup

Platform and control loop. Every experiment uses one NVIDIA A800 (80 GB) with an unmodified vLLM server (Kwon et al., 2023) that serves models in bfloat16 through its OpenAI-compatible endpoint. HALO runs as a separate process and interacts with the server only through NVML telemetry and the SM-clock interface. The control window is $\Delta T = 30$ s, power is sampled every 50 ms, the SM ladder covers 240 to 1,410 MHz with a step of 45 MHz, the memory clock stays at its default, and every run starts at the maximum cap. Appendix B lists the complete platform, serving, and controller configuration.

Workload and replay. The experiments replay the Coding workload of the Azure LLM inference trace (Microsoft Azure, 2024; Stojkovic et al., 2025), which records the arrival time and the input and output token counts of every request over one week. To match the offered load to a single GPU without losing the burstiness of the trace, we subsample the whole week with a Bernoulli filter of retention probability $p = 1/30$ (seed 42) and then extract the contiguous daytime segment from 09:00 to 21:00, a sustained high-load period with frequent bursts that is more demanding than the quiet night hours. The retention probability was fixed by a capacity screen before any HALO run. At $p = 1/3$ the server saturated even at the maximum clock, and $1/30$ is the largest value of the form $1/n$ for which the offered prompt-token rate at the highest replay speed remains below the measured saturation throughput (Appendix C). The slice contains 56,382 requests, which arrive at an average rate of 1.31 requests/s and carry on average 2,602 input tokens and 22.4 output tokens each. Each request is issued at its trace timestamp with its recorded input length and decoded to exactly its recorded output length, replay speed factors scale the interarrival times only, and every run covers the whole slice (1,440 windows at $1\times$).

SLO, metrics, and baseline. Unless stated otherwise, the SLO on the probe tail latency is 10 s (p95). HALO observes only probe latencies, but all reported performance numbers (latency and throughput) are computed from the workload requests recorded by the replay client, excluding the probes, whereas total energy includes the probe overhead. The reported metrics are total energy, average power, the energy per request, mean and p95 workload latency, throughput, the offline mean E^2D objective (equation 6 with the workload mean latency and without the penalty), and the numbers of windows in `RED` and with probe p95 above the SLO. The baseline is *MaxFreq*, a static policy that locks the SM clock at its maximum of 1,410 MHz (no DVFS, no controller), which corresponds to the latency-conservative practice in production. MaxFreq runs the same probe stream as HALO, so the two policies differ only in the frequency setting. HALO uses the actions, zones, context, LinUCB constants, and cost of Section 4 without change in every experiment (Appendix B lists every value). Each configuration is executed once over the full slice, which gives eight HALO runs and six MaxFreq runs (the MaxFreq run at $1\times$ speed is shared by the three SLO settings, because the SLO does not affect a static policy).

5.2 Default setting: large energy savings with unchanged throughput

The default setting serves Mistral-7B-Instruct-v0.3 at $1\times$ replay speed under a 10 s probe SLO. Table 1 summarizes the main metrics. Over the 12-hour slice, HALO lowers total GPU energy from 11,659 kJ to 7,284 kJ, a saving of 37.5%, and average power falls accordingly from 270 W to 169 W. Throughput is identical to three decimals, since both runs complete all 56,382 requests within the same 12 hours, and energy per request falls by the same 37.5%. Mean workload latency increases from 784 ms to 1,223 ms (+56.1%) and workload p95 from 2.3 s to 3.8 s (+64.3%). Tail-latency safety is judged on the probe SLO. HALO spends 1,422 of the 1,440 windows (98.8%) in `GREEN` or `AMBER`, enters `RED` 18 times, and exceeds the 10 s probe SLO in 14 windows (1.0%), each of them a short excursion followed by recovery. The offline E^2D objective also improves, from 49.3 to 31.6 (-35.9%), so the energy saving outweighs the longer latency in the combined score. Overall, HALO reduces GPU energy substantially while almost satisfying the latency SLO, with violations in only 1% of the windows.

5.3 Energy savings over time

Figure 3 shows how the savings accumulate over the 12-hour run, with cumulative energy and instantaneous power for both policies. The two cumulative curves separate within the first hour and keep diverging at an almost constant rate for the rest of the day, so the 37.5% gap does not come from a few quiet intervals. MaxFreq draws between 230 and 300 W for most of the slice, whereas HALO stays between roughly 130 and 200 W during the first half of the day and rises to 200 to 260 W only when the afternoon load builds up. The savings come from a systematic reduction of the power draw in the many windows that have latency headroom, not from throttling during bursts.

5.4 Latency behavior and SLO usage

Table 1 shows that HALO increases mean workload latency by 56% and workload p95 by 64%. This increase is the price of the energy saving, and it stays within the latency budget of the operator, since

Table 1: Default setting of Section 5.1 (1,440 windows). Latency and throughput refer to workload requests, parentheses give the change relative to MaxFreq, and the last two rows count windows with their fraction in parentheses.

Metric	MaxFreq	HALO
Energy (kJ)	11,658.8	7,283.6 (-37.5%)
Avg. power (W)	269.9	168.6 (-37.5%)
Energy/request (J)	206.8	129.2 (-37.5%)
Mean latency (ms)	784	1,223 (+56.1%)
p95 latency (ms)	2,325	3,819 (+64.3%)
Throughput (req/s)	1.305	1.305 (<0.1%)
Mean E^2D	49.3	31.6 (-35.9%)
RED windows	0 (0.0%)	18 (1.2%)
Probe p95 > SLO	0 (0.0%)	14 (1.0%)

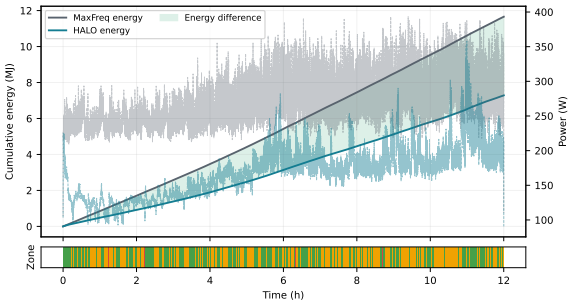

Figure 3: Cumulative GPU energy (solid lines) and instantaneous power (thin traces) over the 12-hour default run. The shaded band is the energy saved by HALO relative to MaxFreq. The strip below shows the latency zone of HALO in every window (green, amber, red).

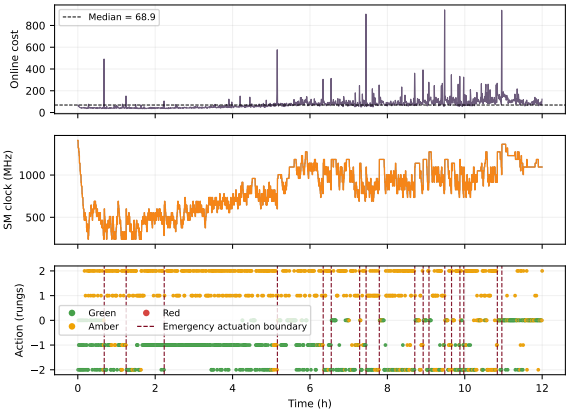

Figure 4: HALO controller behavior on the default run. Top: online E^2D cost with the soft SLO penalty per window (dashed line: median). Middle: window-averaged measured SM clock. Bottom: action taken in every window, colored by the zone in which it was chosen, with emergency ladder-ups marked by dashed vertical lines.

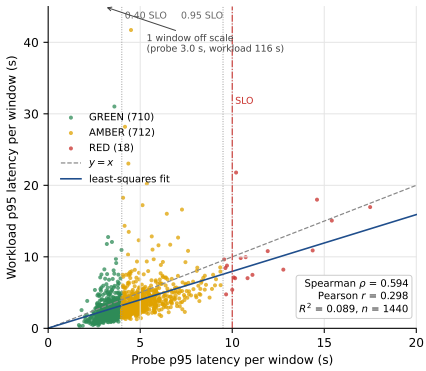

Figure 5: Probe p95 versus workload p95 per 30 s window in the default run (1,440 windows), colored by zone. The dashed line is $y = x$, the solid line is the least-squares fit, and the vertical lines are the zone thresholds and the SLO. One window lies outside the axes (probe 3.0 s, workload 116 s).

the probe p95 exceeds the 10 s SLO in only 14 of the 1,440 windows (1.0%). The controller keeps the SM cap well below the maximum for much of the run, and it may do so as long as the probe tail remains within the SLO. The zone strip in Figure 3 shows how this budget is used. The timeline is divided almost evenly between GREEN (710 windows, 49.3%) and AMBER (712 windows, 49.4%), with GREEN dominant in the quiet morning and AMBER in the busier afternoon.

Moreover, only 18 windows (1.25%) enter RED, and 14 of them exceed the SLO. The other four lie in the 9.5 to 10 s band, in which the gate fires at $0.95 \times$ SLO and starts recovery before the SLO is violated. All 14 violations have the same cause. The gate can act only at a window boundary, so a burst that arrives inside a window pushes the probe tail past the SLO before the controller can respond. The $0.95 \times$SLO threshold and the emergency upshift shorten such an excursion rather than prevent it, and a controller that never exceeds the SLO would need admission control or a predictive component on top of HALO.

In comparison, MaxFreq never enters RED and spends 1,347 windows in GREEN, so the absence of violations is obtained at the price of a large amount of unused headroom. Figure 4 shows the controller traces behind these numbers (details in Appendix G.1). The bandit lowers the cap from 1,410 MHz into the 300 to 500 MHz range within the first hour, holds it between 300 and 700 MHz through the quiet first third of the day, and raises it in stages to 900 to 1,300 MHz as afternoon load arrives, with the emergency ladder-ups concentrated in a busier afternoon.

Table 2: Sensitivity to offered load (Mistral-7B, 10 s SLO), probe SLO (Mistral-7B, 1×), and served model (1×, 10 s SLO). Changes are relative to the matched MaxFreq run, and throughput is unchanged in every setting. The last two columns give window counts for both policies, with the fraction of windows for HALO. Zones and violations are defined relative to the SLO, so the counters of the MaxFreq run, which is shared by the three SLO settings, are re-scored at each threshold from the per-window probe p95 of that run.

					MaxFreq → HALO	
Axis	Setting	Δ energy	Δ mean lat.	Δ p95 lat.	RED	p95 > SLO
Load	0.75×	-40.8%	+73.2%	+78.1%	0 → 9	0 → 7 (0.4%)
Load	1.0× (default)	-37.5%	+56.1%	+64.3%	0 → 18	0 → 14 (1.0%)
Load	1.5×	-28.5%	+35.6%	+39.8%	2 → 28	2 → 24 (2.5%)
Load	2.0×	-18.3%	+9.9%	+7.4%	71 → 89	61 → 77 (10.7%)
SLO	7 s	-32.7%	+36.3%	+40.5%	0 → 31	0 → 22 (1.5%)
SLO	5 s	-26.4%	+13.1%	+13.4%	16 → 55	10 → 39 (2.7%)
Model	Qwen2.5-7B	-40.6%	+83.1%	+93.9%	0 → 14	0 → 11 (0.8%)
Model	Qwen3-14B†	-14.5%	+5.1%	+4.4%	221 → 258	185 → 223 (15.5%)

†Beyond single-GPU capacity, where MaxFreq itself misses the SLO in 12.8% of the windows.

5.5 Probe latency as a safety signal

HALO never reads workload latency, and its probe (one input token, 100 output tokens) differs in shape from a typical request (2,602 input tokens, 22 output tokens), which raises the question of whether the probe p95 says anything about the tail latency that users experience. The probe does not have to reproduce user latency but only has to move with the congestion in the serving stack, and Figure 5 tests this on the 1,440 windows of the default run by plotting probe p95 against workload p95. The Spearman rank correlation is 0.594, so the two rise together. The fitted slope of 0.79 means that workload p95 is on average, lower than probe p95, and the probe therefore errs on the conservative side. The scatter is large ($R^2 = 0.089$), which is why HALO uses the probe through coarse zones and a rank-like penalty rather than as a calibrated estimate of user latency (More details can be found in Appendix D). Taken together, the probe p95 tracks the tail latency that users experience and errs on the conservative side.

5.6 Generality and sensitivity

Table 2 repeats the default experiment across four offered loads, three probe SLOs, and three models with every controller constant unchanged, and Appendix F discusses each axis and gives the absolute numbers for both policies. Throughput is unchanged in every setting, since every run completes the entire trace. The saving decreases monotonically with offered load, from 40.8% at 0.75× to 18.3% at 2×, while the mean latency inflation falls from +73% to +9.9%, which is the queueing-knee behavior of Section 2. At 2× the MaxFreq baseline itself misses the SLO in 8.5% of the windows against 10.7% under HALO, so most of the tail risk at that load comes from the workload and not from the controller. More details can be found in (Appendix D).

6 Conclusion

This paper formulated GPU frequency control for LLM serving as a contextual bandit problem with safety constraints and presented HALO, a black-box controller in which the probe p95 latency gates the admissible SM-frequency moves. Within the admissible set, a LinUCB learner selects the next move from window-level telemetry, guided by an energy-squared-delay cost that is extended by a soft penalty on tail latency. In replays of an Azure LLM inference trace, HALO reduced total GPU energy by 37.5% in the default setting and by 14.5% to 40.8% across offered loads, probe SLOs, and models, with unchanged throughput, and the SLO violations it adds beyond those of the maximum-frequency baseline are rare and short. The savings follow the latency headroom that the workload leaves, in agreement with the queueing intuition behind the design. These results indicate that a simple safety gate combined with model-independent low-overhead online learning is sufficient to convert that headroom into energy savings for interactive LLM services.

AI USE STATEMENT

In this work, we used generative AI tools (OpenAI Codex and Anthropic Claude) for writing assistance and editing, and for help implementing the experimental code, including the data-processing scripts and the plotting code. The research problem, the controller design, and the experimental setting were defined by the authors. All AI-assisted code was reviewed and tested by the authors, every reported number was verified against the raw experiment logs, and the authors reviewed and edited all text. We take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

Sections 4 and 5.1 specify the controller and the evaluation protocol in full, including every constant used in the experiments. Appendix B lists the hardware, software versions, serving configuration, metric definitions, and telemetry validation rules. Appendix C gives the exact trace transformation (seed, retention probability, time slice, and the capacity screen that fixed the retention probability) together with summary statistics of the processed replay. The anonymous supplementary material contains the experiment code.

REFERENCES

Ghazanfar Ali, Mert Side, Sridutt Bhalachandra, Nicholas J. Wright, and Yong Chen. Performance-aware energy-efficient GPU frequency selection using DNN-based models. In *Proceedings of the 52nd International Conference on Parallel Processing (ICPP '23)*, pp. 433–442. ACM, 2023. doi: 10.1145/3605573.3605600.

Enrico Calore, Alessandro Gabbana, Sebastiano Fabio Schifano, and Raffaele Tripiccione. Evaluation of DVFS techniques on modern HPC processors and accelerators for energy-aware applications. *Concurrency and Computation: Practice and Experience*, 29(12):e4143, 2017. doi: 10.1002/cpe.4143.

Shuang Chen, Angela Jin, Christina Delimitrou, and José F. Martínez. ReTail: Opting for learning simplicity to enable QoS-aware power management in the cloud. In *Proceedings of the IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pp. 155–168. IEEE, 2022. doi: 10.1109/HPCA53966.2022.00020.

Jee W. Choi and Richard W. Vuduc. Analyzing the energy efficiency of the fast multipole method using a DVFS-aware energy model. In *Proceedings of the IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pp. 79–88. IEEE, 2016. doi: 10.1109/IPDPSW.2016.206.

Henri Maxime Demoulin, Joshua Fried, Isaac Pedisich, Marios Kogias, Boon Thau Loo, Linh Thi Xuan Phan, and Irene Zhang. When idling is ideal: Optimizing tail-latency for heavy-tailed datacenter workloads with perséphone. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles (SOSP '21)*, pp. 621–637. ACM, 2021. doi: 10.1145/3477132.3483571.

Bishwajit Dutta, Vignesh Adhinarayanan, and Wu-chun Feng. GPU power prediction via ensemble machine learning for DVFS space exploration. In *Proceedings of the 15th ACM International Conference on Computing Frontiers (CF '18)*, pp. 240–243. ACM, 2018. doi: 10.1145/3203217.3203273.

João Guerreiro, Aleksandar Ilic, Nuno Roma, and Pedro Tomás. Multi-kernel auto-tuning on GPUs: Performance and energy-aware optimization. In *Proceedings of the 23rd Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*, pp. 438–445, 2015. doi: 10.1109/PDP.2015.44.

Arpan Gujarati, Reza Karimi, Safya Alzayat, Wei Hao, Antoine Kaufmann, Ymir Vigfusson, and Jonathan Mace. Serving DNNs like Clockwork: Performance predictability from the bottom up. In *Proceedings of the 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pp. 443–462. USENIX Association, 2020.

Yunchu Han, Zhaojun Nan, Sheng Zhou, and Zhisheng Niu. DVFS-aware DNN inference on GPUs: Latency modeling and performance analysis. In *Proceedings of the IEEE International Conference on Communications (ICC)*, pp. 1274–1279. IEEE, 2025. doi: 10.1109/ICC52391.2025.11161883.

David Money Harris and Sarah L. Harris. *Digital Design and Computer Architecture*. Morgan Kaufmann, San Francisco, CA, 2007.

Fleur Jeanquartier, Claire Jean-Quartier, Paul Rieder, Vedad Misirlić, Christian Pasero, Richard Hohensinner, Heimo Müller, and Andreas Holzinger. Assessing the carbon footprint of language models: Towards sustainability in AI. *Resources, Conservation and Recycling*, 226:108670, 2026. doi: 10.1016/j.resconrec.2025.108670.

Nidhal Jegham, Marwan Abdelatti, Chan Young Koh, Lassad Elmoubarki, and Abdeltawab Hendawi. How hungry is AI? Benchmarking energy, water, and carbon footprint of LLM inference. arXiv preprint arXiv:2505.09598, 2025.

Andreas Kosmas Kakolyris, Dimosthenis Masouros, Sotirios Xydis, and Dimitrios Soudris. SLO-aware GPU DVFS for energy-efficient LLM inference serving. *IEEE Computer Architecture Letters*, 23(2):150–153, 2024. doi: 10.1109/LCA.2024.3406038.

Andreas Kosmas Kakolyris, Dimosthenis Masouros, Petros Vavaroutsos, Sotirios Xydis, and Dimitrios Soudris. throttLL'eM: Predictive GPU throttling for energy efficient LLM inference serving. In *Proceedings of the IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pp. 1363–1378, 2025. doi: 10.1109/HPCA61900.2025.00103.

Young Geun Kim and Carole-Jean Wu. AutoScale: Energy efficiency optimization for stochastic edge inference using reinforcement learning. In *Proceedings of the 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 1082–1096, 2020. doi: 10.1109/MICRO50266.2020.00090.

Toshiya Komoda, Shingo Hayashi, Takashi Nakada, Shinobu Miwa, and Hiroshi Nakamura. Power capping of CPU-GPU heterogeneous systems through coordinating DVFS and task mapping. In *Proceedings of the 31st IEEE International Conference on Computer Design (ICCD)*, pp. 349–356, 2013. doi: 10.1109/ICCD.2013.6657064.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pp. 611–626, 2023. doi: 10.1145/3600006.3613165.

Laura Lawniczak and Tobias Distler. Targeting tail latency in replicated systems with proactive rejection. In *Proceedings of the 25th ACM/IFIP International Middleware Conference (Middleware '24)*, pp. 327–340. ACM, 2024. doi: 10.1145/3652892.3700775.

Lihong Li, Wei Chu, John Langford, and Robert E. Schapire. A contextual-bandit approach to personalized news article recommendation. In *Proceedings of the 19th International Conference on World Wide Web (WWW 2010)*, pp. 661–670. ACM, 2010.

Zhuohan Li, Lianmin Zheng, Yinmin Zhong, Vincent Liu, Ying Sheng, Xin Jin, Yanping Huang, Zhifeng Chen, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. AlpaServe: Statistical multiplexing with model parallelism for deep learning serving. In *Proceedings of the 17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*, pp. 663–679. USENIX Association, 2023.

Qunyou Liu, Darong Huang, Marina Zapater, and David Atienza. GreenLLM: SLO-aware dynamic frequency scaling for energy-efficient LLM serving. arXiv preprint arXiv:2508.16449, 2025.

David Lo, Liqun Cheng, Rama Govindaraju, Parthasarathy Ranganathan, and Christos Kozyrakis. Heracles: Improving resource efficiency at scale. In *Proceedings of the 42nd Annual International Symposium on Computer Architecture (ISCA '15)*, pp. 450–462. ACM, 2015. doi: 10.1145/2749469.2749475.

Paul Joe Maliakel, Shashikant Ilager, and Ivona Brandic. Investigating energy efficiency and performance trade-offs in LLM inference across tasks and DVFS settings. arXiv preprint arXiv:2501.08219v3, 2025.

Xinxin Mei, Qiang Wang, and Xiaowen Chu. A survey and measurement study of GPU DVFS on energy conservation. *Digital Communications and Networks*, 3(2):89–100, 2017.

Microsoft Azure. Azure public dataset: LLM inference trace 2024. https://github.com/Azure/AzurePublicDataset/blob/master/AzureLLMInferenceDataset2024.md, 2024.

Mistral AI. Mistral-7B-Instruct-v0.3 model card. https://huggingface.co/mistralai/Mistral-7B-Instruct-v0.3, 2024.

Seyed Morteza Nabavinejad, Sherief Reda, and Masoumeh Ebrahimi. BatchSizer: Power-performance trade-off for DNN inference. In *Proceedings of the 26th Asia and South Pacific Design Automation Conference (ASPDAC '21)*, pp. 819–824. ACM, 2021. doi: 10.1145/3394885.3431535.

Seyed Morteza Nabavinejad, Sherief Reda, and Masoumeh Ebrahimi. Coordinated batching and DVFS for DNN inference on GPU accelerators. *IEEE Transactions on Parallel and Distributed Systems*, 33(10):2496–2508, 2022. doi: 10.1109/TPDS.2022.3144614.

Rajib Nath and Dean M. Tullsen. The CRISP performance model for dynamic voltage and frequency scaling in a GPGPU. In *Proceedings of the 48th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-48)*, pp. 281–293. ACM, 2015. doi: 10.1145/2830772.2830826.

Rajiv Nishtala, Vinicius Petrucci, Paul Carpenter, and Magnus Sjalander. Twig: Multi-agent task management for colocated latency-critical cloud services. In *Proceedings of the IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pp. 167–179, 2020. doi: 10.1109/HPCA47549.2020.00023.

NVIDIA Corporation. *NVML API Reference Guide R550: Device Queries*, 2024. URL https://docs.nvidia.com/deploy/archive/R550/nvml-api/group__nvmlDeviceQueries.html.

NVIDIA Corporation. *NVIDIA System Management Interface: nvidia-smi Documentation*, 2026. URL https://docs.nvidia.com/deploy/nvidia-smi/index.html.

Pratyush Patel, Esha Choukse, Chaojie Zhang, Íñigo Goiri, Brijesh Warrier, Nithish Mahalingam, and Ricardo 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)*, pp. 207–222. ACM, 2024. doi: 10.1145/3620666.3651329.

David Patterson, Joseph Gonzalez, Urs Hölzle, Quoc Le, Chen Liang, Lluís-Miquel Munguia, Daniel Rothchild, David R. So, Maud Texier, and Jeff Dean. The carbon footprint of machine learning training will plateau, then shrink. *Computer*, 55(7):18–28, 2022. doi: 10.1109/MC.2022.3148714.

Indrani Paul, Vignesh Ravi, Srilatha Manne, Manish Arora, and Sudhakar Yalamanchili. Coordinated energy management in heterogeneous processors. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '13)*, pp. 1–12, 2013. doi: 10.1145/2503210.2503227.

Soham Poddar, Paramita Koley, Janardan Misra, Niloy Ganguly, and Saptarshi Ghosh. Towards sustainable NLP: Insights from benchmarking inference energy in large language models. In *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)*, pp. 12688–12704. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.naacl-long.632.

George Prekas, Mia Primorac, Adam Belay, Christos Kozyrakis, and Edouard Bugnion. Energy proportionality and workload consolidation for latency-critical applications. In *Proceedings of the 6th ACM Symposium on Cloud Computing (SoCC '15)*, pp. 342–355. ACM, 2015. doi: 10.1145/2806777.2806848.

Qwen Team. Qwen2.5-7B-Instruct model card. `https://huggingface.co/Qwen/Qwen2.5-7B-Instruct`, 2024.

Qwen Team. Qwen3-14B model card. `https://huggingface.co/Qwen/Qwen3-14B`, 2025.

Peter J. Sevcik. Defining the application performance index. *Business Communications Review*, 35 (3):8–10, March 2005.

Jovan Stojkovic, Nikoleta Iliakopoulou, Tianyin Xu, Hubertus Franke, and Josep Torrellas. EcoFaaS: Rethinking the design of serverless environments for energy efficiency. In *Proceedings of the 51st Annual International Symposium on Computer Architecture (ISCA '24)*, pp. 471–486. IEEE Press, 2024. doi: 10.1109/ISCA59077.2024.00042.

Jovan Stojkovic, Chaojie Zhang, Í nigo Goiri, Josep Torrellas, and Esha Choukse. DynamoLLM: Designing LLM inference clusters for performance and energy efficiency. In *Proceedings of the IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pp. 1348–1362, 2025.

Thierry Tambe, Coleman Hooper, Lillian Pentecost, Tianyu Jia, En-Yu Yang, Marco Donato, Victor Sanh, Paul Whatmough, Alexander M. Rush, David Brooks, and Gu-Yeon Wei. EdgeBERT: Sentence-level energy optimizations for latency-aware multi-task NLP inference. In *Proceedings of the 54th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO '21)*, pp. 830–844. ACM, 2021. doi: 10.1145/3466752.3480095.

Zhenheng Tang, Yuxin Wang, Qiang Wang, and Xiaowen 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)*, pp. 315–325. ACM, 2019. doi: 10.1145/3307772.3328315.

Balajee Vamanan, Hamza Bin Sohail, Jahangir Hasan, and T. N. Vijaykumar. TimeTrader: Exploiting latency tail to save datacenter energy for online search. In *Proceedings of the 48th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-48)*, pp. 585–597. ACM, 2015. doi: 10.1145/2830772.2830779.

Chengcheng Wan, Muhammad Santriaji, Eri Rogers, Henry Hoffmann, Michael Maire, and Shan Lu. ALERT: Accurate learning for energy and timeliness. In *Proceedings of the 2020 USENIX Annual Technical Conference (USENIX ATC '20)*, pp. 353–369. USENIX Association, 2020.

Farui Wang, Weizhe Zhang, Shichao Lai, Meng Hao, and Zheng Wang. Dynamic GPU energy optimization for machine learning training workloads. *IEEE Transactions on Parallel and Distributed Systems*, 33(11):2943–2954, 2022. doi: 10.1109/TPDS.2021.3137867.

Tianyu Wang, Gourav Rattihalli, Aditya Dhakal, Xulong Tang, and Dejan Milojicic. WAGES: Workload-aware GPU sharing system for energy-efficient serverless LLM serving. In *Proceedings of the SC Workshops '25*, pp. 508–515. ACM, 2025a. doi: 10.1145/3731599.3767396.

Yi Wang, Qinglai Guo, and Min Chen. Providing load flexibility by reshaping power profiles of large language model workloads. *Advances in Applied Energy*, 19:100232, 2025b. doi: 10.1016/j.adapen.2025.100232.

Yiming Wang, Meng Hao, Hui He, Weizhe Zhang, Qiuyuan Tang, Xiaoyang Sun, and Zheng Wang. DRLCAP: Runtime GPU frequency capping with deep reinforcement learning. *IEEE Transactions on Sustainable Computing*, 9(5):712–726, 2024. doi: 10.1109/TSUSC.2024.3362697.

Grant Wilkins, Srinivasan Keshav, and Richard Mortier. Offline energy-optimal LLM serving: Workload-based energy models for LLM inference on heterogeneous systems. *ACM SIGEnergy Energy Informatics Review*, 4(5):113–119, April 2025. doi: 10.1145/3727200.3727217.

Hao Xu, Long Peng, Shezheng Song, Xiaodong Liu, Jun Ma, Shasha Li, Jie Yu, and Xiaoguang Mao. Camel: Energy-aware LLM inference on resource-constrained devices. arXiv preprint arXiv:2508.09173, 2025.

Zicong Ye, Kunming Zhang, and Guoming Tang. AGFT: An adaptive GPU frequency tuner for real-time LLM inference optimization, 2025.

Jie You, Jae-Won Chung, and Mosharaf Chowdhury. Zeus: Understanding and optimizing GPU energy consumption of DNN training. In *Proceedings of the 20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pp. 119–139. USENIX Association, 2023.

Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soojeong Kim, and Byung-Gon Chun. Orca: A distributed serving system for transformer-based generative models. In *Proceedings of the 16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 2022)*, pp. 521–538. USENIX Association, 2022.

Yijia Zhang, Qiang Wang, Zhe Lin, Pengxiang Xu, and Bingqiang Wang. Improving GPU energy efficiency through an application-transparent frequency scaling policy with performance assurance. In *Proceedings of the Nineteenth European Conference on Computer Systems (EuroSys '24)*, pp. 769–785. ACM, 2024. doi: 10.1145/3627703.3629584.

Liang Zhou, Laxmi N. Bhuyan, and K. K. Ramakrishnan. Gemini: Learning to manage CPU power for latency-critical search engines. In *Proceedings of the 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 637–649. IEEE, 2020. doi: 10.1109/MICRO50266.2020.00059.

A Related Work

DVFS for CPUs, GPUs, and neural-network workloads. DVFS is a mature technique for balancing energy against performance and has been studied on many platforms. Early work concentrated on CPUs and reported energy reductions for general-purpose as well as machine-learning workloads, in many cases without a performance guarantee (Chen et al., 2022; Demoulin et al., 2021; Nishtala et al., 2020; Prekas et al., 2015; Stojkovic et al., 2024). As GPUs became the dominant deep-learning accelerator, later studies characterized their distinct power-performance behavior and showed that a careful choice of the core frequency improves energy efficiency considerably (Ali et al., 2023; Calore et al., 2017; Dutta et al., 2018; Guerreiro et al., 2015; Komoda et al., 2013; Zhang et al., 2024). These findings motivated a number of DVFS-based methods for training and inference (Wang et al., 2022), including resource partitioning (Tambe et al., 2021), autoscaling, and static or dynamic frequency scaling (Kakolyris et al., 2024; Kim & Wu, 2020; Nabavinejad et al., 2021; 2022). Several of the early designs optimized energy in isolation without an explicit latency or tail-performance bound, which limits their use for interactive inference (Guerreiro et al., 2015; Paul et al., 2013). With the rise of deep networks and then LLMs, attention moved to energy-aware management of inference GPUs (Wan et al., 2020; You et al., 2023). Earlier GPU DVFS work considered convolutional networks and similar workloads, for which coarse control is often sufficient (Wang et al., 2025b). LLM serving is different, because execution alternates between a compute-heavy prefill phase and latency-sensitive, highly irregular decode loops (Liu et al., 2025; Patel et al., 2024), and the fine-grained per-operator DVFS that has been demonstrated on NPUs depends on detailed workload models and custom hardware support (Stojkovic et al., 2025).

Energy management for LLM serving. At the system level, cluster-scale designs oversubscribe power budgets so that more inference servers fit under a data-center power cap, or they fit offline energy models whose parameters are the prompt and output lengths and use them to build energy-optimal heterogeneous schedulers (Patel et al., 2024; Wilkins et al., 2025). GreenLLM comes closer to runtime control, since it fits compact latency-power models for the prefill phase online and adjusts the frequency through queueing-aware optimization (Liu et al., 2025). Iteration-level DVFS controllers follow the decode loop in order to hold a high-percentile latency target (Kakolyris et al., 2024), and later extensions add projections of KV-cache occupancy and autoscaling (Kakolyris et al., 2025). In parallel, measurement studies have quantified the footprint of LLM inference and found that a single query may cost tens of watt-hours (Jegham et al., 2025; Jeanquartier et al., 2026; Wang et al., 2025a). Learning-based controllers adjust frequency caps online, for example reinforcement-learning agents that operate under a latency constraint (Wang et al., 2024) and the AGFT bandit tuner (Ye et al., 2025). These methods usually depend on extensive offline profiling, assume a stationary or predictable load, require access to the internals of the model or the scheduler, or act at a coarse granularity. Despite this progress (Maliakel et al., 2025; Xu et al., 2025), none of the existing approaches gives operators a practical and non-intrusive way to control the energy of each GPU at fine granularity in a live multi-tenant LLM service, where the load is bursty and the internal state of the model and the scheduler cannot be observed. HALO addresses this gap with a profile-free controller at the hardware level for which the GPU is a black box, and it is orthogonal to the work on scheduling and memory management discussed above. Partitioning response times into a few qualitatively distinct regions is itself an established practice in performance engineering. One example is the Apdex standard with its zones for "satisfied", "tolerating", and "frustrated" users, which are combined into a satisfaction score (Sevcik, 2005). Our zones follow a similar idea, but they gate the admissible DVFS actions instead of feeding a score.

B Experimental Details

This appendix records the platform, serving configuration, controller constants, and metric definitions behind Section 5. Every HALO run uses the same controller configuration.

B.1 Platform and serving configuration

The replay client keeps the offered load open loop. Every request is dispatched at its scheduled time regardless of how many requests are still outstanding, and the client logs its own dispatch lag separately so that a slow server cannot silently reshape the arrival process. A run is discarded if any

Table 3: Evaluation platform. The same machine and software versions were used for all fourteen runs.

Component	Value
GPU	one NVIDIA A800-SXM4-80GB (80 GB memory, 400 W power limit)
CPU and host memory	Intel Xeon Platinum 8358P (8 cores, 16 hardware threads), 236 GiB
OS, driver, CUDA	Ubuntu 22.04.1 LTS, driver 550.144.03, CUDA 12.4 driver runtime
Serving engine	vLLM 0.19.0 (stock scheduler, OpenAI-compatible server), PyTorch 2.10.0
Telemetry library	nvidia-ml-py 13.595.45 (NVML)
SM ladder	240 to 1,410 MHz in 45 MHz steps (27 rungs), locked with `nvidia-smi -lgc`
Memory clock	unmanaged for both policies (device ceiling 1,593 MHz)

Table 4: Model and serving configuration, identical for MaxFreq and HALO.

Item	Value
Models	Mistral-7B-Instruct-v0.3 (Mistral AI, 2024), Qwen2.5-7B-Instruct (Qwen Team, 2024), Qwen3-14B (Qwen Team, 2025)
Numerical format	bfloat16, no quantization, one GPU (no tensor parallelism)
Context and batching	maximum sequence length 16,384 tokens, at most 512 sequences and 16,384 batched tokens per step, chunked prefill enabled, prefix caching disabled, eager execution
GPU memory target	0.90 of device memory
Decoding	temperature 0, top-p 1, output length fixed to the trace value, EOS ignored
Request construction	a single tokenizer-safe token repeated to the recorded input length
HTTP transport	one pooled client for both policies, single-attempt requests, no retries

request is missing, duplicated, or truncated, if any request fails, or if any response has an output length that differs from the trace. All fourteen reported runs completed all 56,382 requests.

B.2 Controller configuration

Table 5: Controller configuration used in every HALO run.

Parameter	Value
Control window ΔT	30 s
Actions	$\{\mathrm{SM}_{-2}, \mathrm{SM}_{-1}, \mathrm{HOLD}, \mathrm{SM}_{+1}, \mathrm{SM}_{+2}\}$, i.e., $\{-90, -45, 0, +45, +90\}$ MHz on the 45 MHz ladder
Zone thresholds	`GREEN`: $L_t^{95}/\mathrm{SLO} < 0.40$, `AMBER`: $0.40 \le L_t^{95}/\mathrm{SLO} \le 0.95$, `RED`: $L_t^{95}/\mathrm{SLO} > 0.95$
Emergency upshift	eight rungs (360 MHz) on a `GREEN`/`AMBER`-to-`RED` transition, capped at 1,410 MHz
Penalty	$L_0 = 0.65 \times \mathrm{SLO}$, $\lambda = 1.0$, $\gamma = 2.0$
LinUCB	ridge regularization 1.0, $\alpha_0 = 1.0$, $\alpha_{\min} = 0.1$
Reward scale β	100
Latency feature clip	$3\times$
Ties and seed	seeded uniform choice among numerically tied scores, seed 42
Ladder boundary	intersect the zone set with the moves that stay on the ladder, hold if the intersection is empty
Initial state	1,410 MHz cap, no previous zone (a `RED` first window holds rather than triggering an emergency move)
Windows that do not update LinUCB	emergency windows, forced holds at a ladder boundary, and windows without a successful probe (such a window requests the maximum cap)
Probe	one input token, exactly 100 output tokens, issued back to back, identical for both policies

Window alignment. Let a_t be the move requested at the boundary before window t. The learner stores the context x_{t-1} that was available when a_t was chosen. When window t ends, it computes r_t and updates the model of a_t with (x_{t-1}, r_t), then uses x_t to choose a_{t+1}. The emergency path is logged separately from the five ordinary actions, changes the cap, and creates no sixth bandit model. A forced hold at a ladder boundary is likewise logged separately and does not update the HOLD model.

Actuation. At every window boundary the controller verifies the GPU identity, applies the new cap with `nvidia-smi -lgc` (NVIDIA Corporation, 2026), and reads the resulting cap back. A

timeout, a nonzero exit status, or a readback that does not match the request aborts the run, so no reported run contains a window whose cap is uncertain. Actuation takes about 50 ms and does not interrupt power sampling, which is buffered by the driver.

B.3 Telemetry and energy

Power is read from the total-power sample buffer of the driver (`NVML_TOTAL_POWER_SAMPLES`) at intervals of 50 ms (NVIDIA Corporation, 2024), and each driver record is consumed exactly once. On the A800 the driver produces a new sample roughly every 20 ms. Window energy E_t is the time integral of the power samples under a causal zero-order hold, i.e., between two samples the power is taken to be the latest sample at or before that time. Each record is assumed to describe at most 50 ms, and a window is valid only if these support intervals cover at least 98% of it, which excludes windows with sparse telemetry. SM utilization and the clock caps in force are polled separately at the same interval and with the same coverage rule. The cumulative energy counter of the device is recorded as a whole-run diagnostic only. It does not enter the controller, the reward, or the reported numbers.

B.4 Metric definitions

Total energy is the sum of the window energies over the full measurement interval, including the final partial window. Average power is total energy divided by the exact run duration. Energy per request is total energy divided by the number of completed workload requests (probe requests are excluded from the count but their energy is included). Workload latency is measured by the replay client from the scheduled arrival time to the observed completion of the response, so it includes any client-side dispatch lag. Mean latency is averaged over all completed workload requests, workload p95 uses linear interpolation over those latencies, and throughput is the number of completed workload requests per second of the measurement interval. Requests are assigned to windows by completion time. The offline $\mathrm{E}^2\mathrm{D}$ objective is the mean over complete windows of $E_t^2 \bar{L}_t^{\mathrm{user}}$ with E_t in kJ and the workload mean latency $\bar{L}_t^{\mathrm{user}}$ in seconds, without the online probe penalty. Zone counts, online costs, and the probe validation use complete windows only.

C Trace Construction

Source records. We use the Coding workload of the official Azure trace of LLM inference requests, which covers one week (Microsoft Azure, 2024). Each record carries a request timestamp, an input-token count q_i, and a generated-token count g_i. The experimental replay is produced in two steps, load scaling by Bernoulli subsampling and time-of-day slicing, and a third diagnostic step checks that subsampling preserves the temporal structure of the week.

Step 1: load scaling by Bernoulli subsampling. To match the offered load to a single GPU we sample requests independently. Every row of the full week is visited in file order and retained with probability $p = 1/30$ using a fixed pseudo-random seed (`RandomState(42)`). This preserves the per-request characteristics (the token counts) exactly, preserves temporal burstiness in expectation, and scales the arrival rate by a factor of approximately p. The full-week draw is completed before any time filter is applied, so the sample does not depend on the slice.

Step 2: daytime slice. The subsampled trace is then cut to the contiguous 12-hour daytime segment from 09:00:00 to 21:00:00 UTC of the first trace day (2024-05-10). This segment is a high-load daytime interval that has less latency headroom and more bursts than the quiet night hours, and it therefore gives a more demanding setting for controller behavior under sustained pressure. Timestamps are shifted so that the first retained request arrives at $t = 0$. A replay speed factor v schedules request i at $(t_i - t_0)/v$ and changes nothing else, so all four speeds replay the same request sequence.

Why $p = 1/30$. The screen started from $p = 1/3$. On the A800 with the serving configuration of Appendix B, a preliminary run at that retention probability saturated the server even with the SM clock held at its maximum. The controller stayed at 1,410 MHz, thousands of requests waited in the queue, and the prompt-processing rate of the server reached a plateau. Over the 31 ten-second reporting intervals of that run with at least 3,900 waiting requests, the prompt throughput had a median of 14,719 and a first quartile of 14,510 prompt tokens/s. We rounded the first quartile down

to a reference capacity $C_{\text{ref}} = 14{,}500$ prompt tokens/s and fixed the following screen before any HALO experiment. Candidate retention probabilities $p = 1/n$ are nested. Every row receives one uniform draw u_i and is retained for candidate n exactly when $u_i < 1/n$, so a larger n removes rows but never draws a different sample. For each candidate we project the prompt-token rate of every 300 s interval W of the daytime slice at the highest replay speed, $R_{2\times}(W) = 2\sum_{i:t_i \in W} q_i/300$ s, and require the 99th percentile over the 144 fixed bins to stay below $0.85\, C_{\text{ref}}$ and the maximum over all rolling 300 s intervals to stay below $0.95\, C_{\text{ref}}$. Candidate $n = 29$ fails both conditions and $n = 30$ is the first to pass (12,289 against a limit of 12,325 prompt tokens/s for the fixed bins, and 13,665 against 13,775 for the rolling maximum). The screen uses prompt tokens only. Before the formal runs, a readiness run over the densest 16-minute interval of the slice (the shortest interval that contains both the prompt-heaviest and the generation-heaviest 300 s windows, replayed at $2\times$ for Mistral-7B and at $1\times$ for the two Qwen models) confirmed for every model that all requests complete, that the client never has to delay a dispatch, and that peak concurrency stays below the 512-sequence limit of the server.

Step 3: full-week intensity check. To check that the daily rhythm and the bursts of the original trace survive subsampling, we bin the full original week and the full subsampled week, before slicing, into fixed intervals of $\Delta = 300$ s and compute the token rate offered in each bin

$$\text{TokRate}(b) = \frac{\sum_{i \in b}(q_i + g_i)}{\Delta}. \tag{8}$$

Figure 6 plots both curves, each normalized by its own peak so that their shapes can be compared directly (the sampled curve is about 30 times smaller in absolute terms). The daily rhythm and the burst structure of the week survive the sampling, and the shaded daytime segment of the first day is the slice replayed in all experiments.

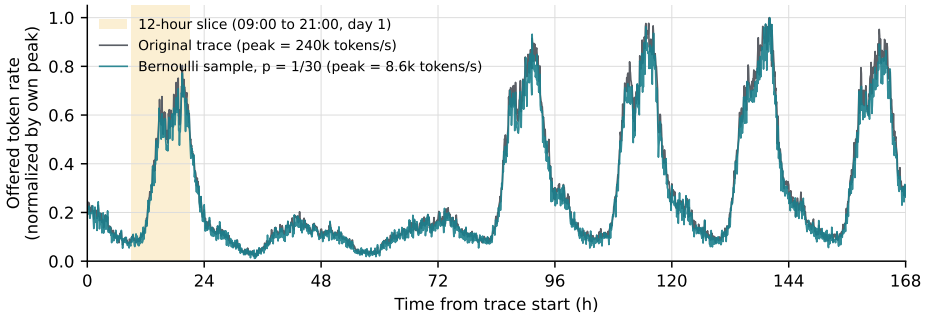

Figure 6: Offered token rate of the original one-week Azure Coding trace and of the $p = 1/30$ Bernoulli sample in 300 s bins, each normalized by its own peak. The shaded band marks the 09:00 to 21:00 slice of the first day used in the experiments.

Processed replay summary. Table 6 summarizes the final replay. The trace retains the daily variation of the load and the burstiness within intervals that characterize production LLM inference. Before every run the replay code re-verifies the hash of the file, the request count, the timestamps, and the token statistics.

Table 6: Statistics of the processed 12-hour replay ($p = 1/30$, seed 42, 09:00 to 21:00).

Statistic	Value
Requests	56,382
Duration	43,197 s (12.0 h)
Mean request rate	1.305 requests/s
Mean / median interarrival time	766 ms / 422 ms
Mean / maximum input tokens	2,602 / 7,691
Mean / maximum generated tokens	22.4 / 5,000
Mean total tokens per request	2,625

D PROBE LATENCY VALIDATION

Section 5.5 shows that the probe p95 moves with the workload p95 on the default run. This appendix gives the full statistics behind Figure 5 and repeats them for all fourteen runs.

For every complete 30 s window of the default HALO run that has at least one successful probe and one completed workload request (all 1,440 windows qualify), we compute the probe p95 and the workload p95 of the requests completed in that window. Table 7 lists the statistics of the relation plotted in Figure 5. The Spearman rank correlation of $\rho = 0.594$ shows a clear monotone relation. When workload latency rises because of congestion, probe latency rises with it. The fitted line has a slope of 0.79 with an intercept near zero, so on average workload p95 is somewhat lower than probe p95. The probe therefore errs on the conservative side. Two effects explain the offset and the scatter. A probe request spends 100 decode iterations in the batch and accumulates queueing and batching delay throughout that period, whereas most workload requests are prefill-dominated and leave after about 22 decode steps. In addition, the workload p95 of a window is computed over a variable and sometimes small number of completions. The linear fit consequently explains only a small part of the variance ($R^2 = 0.089$, Pearson $r = 0.298$), which is why HALO uses the probe through coarse zones and a rank-like penalty rather than as a calibrated estimate of user latency.

The zone counts confirm that the probe identifies the safe operating region. Of the 1,440 windows, 710 are `GREEN` (probe p95 below 40% of the SLO), 712 are `AMBER`, and only 18 are `RED`, and 14 of those exceed the 10 s probe SLO. Table 8 repeats the association statistics for all fourteen runs. The rank correlation is moderate under light load, where both latencies are small and dominated by noise, and rises to 0.76 to 0.84 at 2× speed and for Qwen3-14B, where congestion dominates both signals. The safety signal matters most in this regime, and this is also where the probe follows the workload most closely.

Table 7: Probe validation statistics for the default HALO run.

Statistic	Value
Aligned complete windows	1440 of 1440
Spearman rank correlation ρ	0.594
Pearson correlation r	0.298
Least-squares fit (workload p95 vs. probe p95)	$y = 0.794\,x + 25$ ms
R^2 of the fit	0.089
`GREEN` / `AMBER` / `RED` windows	710 / 712 / 18
Windows with probe p95 above the SLO	14

Table 8: Window-level association between probe p95 and workload p95 for all fourteen runs (workload p95 regressed on probe p95).

Setting	Arm	Spearman ρ	Pearson r	Slope	R^2
Mistral-7B, 0.75×, 10 s	MaxFreq	0.495	0.169	0.575	0.029
	HALO	0.555	0.271	0.742	0.073
Mistral-7B, 1×, 10 s	MaxFreq	0.532	0.194	0.483	0.037
	HALO	0.594	0.298	0.794	0.089
Mistral-7B, 1.5×, 10 s	MaxFreq	0.696	0.436	0.584	0.190
	HALO	0.675	0.628	0.667	0.394
Mistral-7B, 2×, 10 s	MaxFreq	0.843	0.757	0.701	0.573
	HALO	0.785	0.780	0.706	0.609
Mistral-7B, 1×, 7 s	MaxFreq	0.532	0.194	0.483	0.037
	HALO	0.593	0.320	0.633	0.102
Mistral-7B, 1×, 5 s	MaxFreq	0.532	0.194	0.483	0.037
	HALO	0.563	0.277	0.588	0.077
Qwen2.5-7B, 1×, 10 s	MaxFreq	0.510	0.162	0.468	0.026
	HALO	0.610	0.284	0.757	0.081
Qwen3-14B, 1×, 10 s	MaxFreq	0.789	0.712	0.772	0.507
	HALO	0.761	0.688	0.752	0.473

E SENSITIVITY

Tightening the probe SLO from 10 s to 5 s lowers the saving from 37.5% to 26.4% and the mean latency inflation from +56% to +13%, so the SLO acts as a conservatism setting for the operator. Qwen2.5-7B behaves like Mistral-7B (40.6% saving, 14 RED windows). Qwen3-14B is an overloaded setting, since at this offered load the model exceeds the capacity of a single A800 and no frequency policy can meet the SLO, with MaxFreq itself missing it in 12.8% of the windows. That row therefore measures how much energy HALO still saves under overload and how few violations it adds rather than SLO compliance, and HALO saves 14.5% while raising the violation fraction to 15.5%. Normalizing whole-GPU energy by all generated tokens, probe included, still gives 12% to 27% less energy per token under HALO (Appendix F).

F COMPLETE AGGREGATE RESULTS

Tables 9 to 11 give the absolute values behind the summary in Table 2 for both policies in every setting. Energy is whole-GPU energy including the probe stream, and latency is computed over workload requests only. Throughput is omitted because every run completes all 56,382 requests in the same wall-clock time (0.979, 1.305, 1.957, and 2.609 requests/s at $0.75\times$, $1\times$, $1.5\times$, and $2\times$), and energy per request and average power follow from total energy and the fixed request count and run length (Table 1 lists them for the default pair). Parentheses give the change of the HALO run relative to its matched MaxFreq run, and the risk columns count windows (fraction of complete windows). Table 12 lists how the windows of each run are distributed over the three zones, and Table 13 reports the probe volume of every run and the energy per generated token.

F.1 DISCUSSION OF THE THREE SWEEPS

Offered load (replay speed). At speeds of $0.75\times$, $1\times$, $1.5\times$, and $2\times$, the energy saving decreases monotonically, from 40.8% to 37.5%, 28.5%, and 18.3%. This is the queueing-knee behavior of Section 2, since the latency headroom disappears as the load approaches capacity and the controller must remain at high caps more often. The latency cost moves in the opposite direction. The inflation of mean latency is largest when slack is plentiful (+73% at $0.75\times$) and falls to +9.9% at $2\times$, and p95 inflation falls from +78% to +7.4%, which reflects the more conservative choices of the controller under sustained pressure. Tail risk grows with load, from 9 RED windows at $0.75\times$ to 89 at $2\times$, but at $2\times$ the MaxFreq baseline itself has 71 RED windows and 61 windows above the probe SLO, so most of that risk is caused by the workload rather than by the controller. The $2\times$ slice runs the GPU close to saturation, and HALO still removes 18% of the energy while it adds 16 violation windows over the six-hour run.

Operator policy (probe SLO). The probe SLO gives the operator direct control over how conservative the policy is. When the SLO is tightened at $1\times$ speed from 10 s to 7 s and then to 5 s, HALO spends less time at low caps, so the energy saving decreases (37.5%, 32.7%, 26.4%) and the inflation of mean workload latency falls in step (+56%, +36%, +13%). Throughput is unchanged, so a tighter SLO moves the operating point between energy and latency without changing the capacity of the system. The number of RED windows grows as the thresholds move closer to the latency of the workload itself (18, 31, 55), because a 5 s probe SLO leaves a much smaller absolute margin above normal operation. The same margin binds the baseline: re-scored at 5 s, the MaxFreq run has 16 RED windows and 10 windows above the SLO, against none at 7 and 10 s.

Across models. The default experiment is repeated with Qwen2.5-7B-Instruct and Qwen3-14B while every controller constant stays fixed. On Qwen2.5-7B, HALO behaves as on Mistral-7B: 40.6% less energy, unchanged throughput, and 14 RED windows. The 14B model behaves differently, because at this offered load it runs close to capacity even at the maximum clock. Its MaxFreq run already has 221 RED windows (15.3%) and 185 windows above the probe SLO, and its mean workload latency is 2.2 s against 0.8 s for the 7B models. HALO still saves 14.5% of the energy, almost all of it in the quieter first third of the slice, while it adds only 5% to the mean latency and 38 windows to the violation count of the baseline itself, because the gate keeps the cap near the maximum once the afternoon load arrives. The controller therefore transfers across models without retuning, and the size of the saving is determined by the headroom that the workload leaves, as Section 2 predicts.

F.2 TABLES

Table 9: Offered-load sweep for Mistral-7B with a 10 s probe SLO. Run lengths are 16, 12, 8, and 6 hours (1,920, 1,440, 960, and 720 windows).

Speed	Arm	Energy (kJ)	Mean lat. (ms)	p95 lat. (ms)	E^2D	RED	p95 > SLO
0.75×	MaxFreq	15,235.1	685	1,972	41.7	0 (0.0%)	0 (0.0%)
	HALO	9,021.1 (-40.8%)	1,187 (+73.2%)	3,512 (+78.1%)	26.0 (-37.6%)	9 (0.5%)	7 (0.4%)
1.0×	MaxFreq	11,658.8	784	2,325	49.3	0 (0.0%)	0 (0.0%)
	HALO	7,283.6 (-37.5%)	1,223 (+56.1%)	3,819 (+64.3%)	31.6 (-35.9%)	18 (1.2%)	14 (1.0%)
1.5×	MaxFreq	8,483.7	1,020	3,233	73.8	2 (0.2%)	2 (0.2%)
	HALO	6,062.5 (-28.5%)	1,384 (+35.6%)	4,521 (+39.8%)	56.3 (-23.7%)	28 (2.9%)	24 (2.5%)
2.0×	MaxFreq	6,753.4	1,649	5,789	132.7	71 (9.9%)	61 (8.5%)
	HALO	5,519.1 (-18.3%)	1,812 (+9.9%)	6,215 (+7.4%)	117.8 (-11.3%)	89 (12.4%)	77 (10.7%)

Table 10: Probe-SLO sensitivity for Mistral-7B at 1× speed. The MaxFreq run is the same physical run in all three rows, and its zone and violation counters are re-scored at each threshold from the per-window probe p95 of that run.

Probe SLO	Arm	Energy (kJ)	Mean lat. (ms)	p95 lat. (ms)	E^2D	RED	p95 > SLO
10 s	MaxFreq	11,658.8	784	2,325	49.3	0 (0.0%)	0 (0.0%)
	HALO	7,283.6 (-37.5%)	1,223 (+56.1%)	3,819 (+64.3%)	31.6 (-35.9%)	18 (1.2%)	14 (1.0%)
7 s	MaxFreq	11,658.8	784	2,325	49.3	0 (0.0%)	0 (0.0%)
	HALO	7,842.0 (-32.7%)	1,068 (+36.3%)	3,266 (+40.5%)	30.8 (-37.6%)	31 (2.2%)	22 (1.5%)
5 s	MaxFreq	11,658.8	784	2,325	49.3	16 (1.1%)	10 (0.7%)
	HALO	8,577.2 (-26.4%)	886 (+13.1%)	2,638 (+13.4%)	30.9 (-37.3%)	55 (3.8%)	39 (2.7%)

Table 11: Cross-model comparison at 1× speed with a 10 s probe SLO and unchanged controller constants.

Model	Arm	Energy (kJ)	Mean lat. (ms)	p95 lat. (ms)	E^2D	RED	p95 > SLO
Mistral-7B	MaxFreq	11,658.8	784	2,325	49.3	0 (0.0%)	0 (0.0%)
	HALO	7,283.6 (-37.5%)	1,223 (+56.1%)	3,819 (+64.3%)	31.6 (-35.9%)	18 (1.2%)	14 (1.0%)
Qwen2.5-7B	MaxFreq	11,935.6	678	1,986	44.8	0 (0.0%)	0 (0.0%)
	HALO	7,087.0 (-40.6%)	1,240 (+83.1%)	3,851 (+93.9%)	30.3 (-32.4%)	14 (1.0%)	11 (0.8%)
Qwen3-14B	MaxFreq	14,660.7	2,242	7,694	198.3	221 (15.3%)	185 (12.8%)
	HALO	12,541.2 (-14.5%)	2,356 (+5.1%)	8,032 (+4.4%)	175.1 (-11.7%)	258 (17.9%)	223 (15.5%)

Table 12: Distribution of complete windows over latency zones for every run, with the number of windows whose probe p95 exceeds the SLO. Zones of MaxFreq runs are computed from the same probe stream and thresholds but never influence the static policy. The MaxFreq run at 1× is shared by the three SLO settings, and its counts are re-scored at each threshold.

Setting	Arm	Windows	GREEN	AMBER	RED	p95 > SLO
Mistral-7B, 0.75×, 10 s	MaxFreq	1,920	1,912 (99.6%)	8 (0.4%)	0 (0.0%)	0 (0.0%)
	HALO	1,920	1,153 (60.1%)	758 (39.5%)	9 (0.5%)	7 (0.4%)
Mistral-7B, 1×, 10 s	MaxFreq	1,440	1,347 (93.5%)	93 (6.5%)	0 (0.0%)	0 (0.0%)
	HALO	1,440	710 (49.3%)	712 (49.4%)	18 (1.2%)	14 (1.0%)
Mistral-7B, 1.5×, 10 s	MaxFreq	960	650 (67.7%)	308 (32.1%)	2 (0.2%)	2 (0.2%)
	HALO	960	400 (41.7%)	532 (55.4%)	28 (2.9%)	24 (2.5%)
Mistral-7B, 2×, 10 s	MaxFreq	720	327 (45.4%)	322 (44.7%)	71 (9.9%)	61 (8.5%)
	HALO	720	206 (28.6%)	425 (59.0%)	89 (12.4%)	77 (10.7%)
Mistral-7B, 1×, 7 s	MaxFreq	1,440	796 (55.3%)	644 (44.7%)	0 (0.0%)	0 (0.0%)
	HALO	1,440	379 (26.3%)	1,030 (71.5%)	31 (2.2%)	22 (1.5%)
Mistral-7B, 1×, 5 s	MaxFreq	1,440	262 (18.2%)	1,162 (80.7%)	16 (1.1%)	10 (0.7%)
	HALO	1,440	176 (12.2%)	1,209 (84.0%)	55 (3.8%)	39 (2.7%)
Qwen2.5-7B, 1×, 10 s	MaxFreq	1,440	1,427 (99.1%)	13 (0.9%)	0 (0.0%)	0 (0.0%)
	HALO	1,440	710 (49.3%)	716 (49.7%)	14 (1.0%)	11 (0.8%)
Qwen3-14B, 1×, 10 s	MaxFreq	1,440	495 (34.4%)	724 (50.3%)	221 (15.3%)	185 (12.8%)
	HALO	1,440	382 (26.5%)	800 (55.6%)	258 (17.9%)	223 (15.5%)

Probe work and energy per token. Both policies run the same back-to-back probe stream, but a slower system completes fewer probes, so the probe volume differs between the two runs of a pair. Table 13 reports the number of completed probes and probe output tokens of every run together with whole-GPU energy divided by all successfully generated output tokens, workload and probe combined (the workload token count is the same in every run, 1,264,145 tokens). Each of the first three columns gives the MaxFreq value and then the HALO value. The energy advantage of HALO

survives when the output of the probe itself is credited to the energy total. Energy per generated token is 12% to 27% lower under HALO across the eight settings.

Table 13: Probe volume and whole-GPU energy per generated token (workload plus probe), MaxFreq before HALO in the first three columns.

Setting	Probe requests	Probe tokens (M)	J per token	Δ J/token	Δ energy
Mistral-7B, 0.75×, 10 s	31,589 / 23,084	3.16 / 2.31	3.44 / 2.53	-26.7%	-40.8%
Mistral-7B, 1×, 10 s	21,719 / 16,092	2.17 / 1.61	3.39 / 2.53	-25.3%	-37.5%
Mistral-7B, 1.5×, 10 s	13,124 / 10,491	1.31 / 1.05	3.29 / 2.62	-20.4%	-28.5%
Mistral-7B, 2×, 10 s	8,336 / 6,838	0.83 / 0.68	3.22 / 2.83	-12.0%	-18.3%
Mistral-7B, 1×, 7 s	21,719 / 18,940	2.17 / 1.89	3.39 / 2.48	-26.8%	-32.7%
Mistral-7B, 1×, 5 s	21,719 / 21,139	2.17 / 2.11	3.39 / 2.54	-25.2%	-26.4%
Qwen2.5-7B, 1×, 10 s	24,567 / 16,622	2.46 / 1.66	3.21 / 2.42	-24.5%	-40.6%
Qwen3-14B, 1×, 10 s	12,751 / 12,045	1.28 / 1.20	5.77 / 5.08	-12.0%	-14.5%

G Complete Time Series and Controller Traces

This appendix first walks through the controller traces of the default run (Figure 4) and then shows, for every setting, the same two panels as Figure 3 and Figure 4: cumulative energy and instantaneous power of both policies with the zone strip of HALO (top), and the online cost, the window-averaged measured SM clock, and the per-window actions of HALO with emergency ladder-ups (bottom). The default run is repeated in each group so that every group is self-contained.

G.1 Controller behavior on the default run

Figure 4 shows the internal signals of the controller on the default run (Mistral-7B, 1× speed, 10 s probe SLO). Early in the run the bandit explores aggressively. It starts from the maximum cap of 1,410 MHz, and within the first hour repeated GREEN-zone downshifts bring the cap into the 300 to 500 MHz range, in 45 MHz steps. During the quiet first third of the day the cap stays mostly between 300 and 700 MHz, and the action panel is dominated by SM_{-1} and SM_{-2} chosen in GREEN together with holds in AMBER. As the load builds up after hour four, most windows are classified as AMBER, and the bandit raises the cap in stages to the 900 to 1,300 MHz band, with occasional downshifts that are reverted when the cost rises. The online cost stays near its median of 68.9 for most of the run (the same cost has a median of 136 on the MaxFreq run), which confirms that the low caps save energy without compromising the SLO. The spikes of the cost coincide with the emergency ladder-ups in the bottom panel, which cluster in the busier afternoon. Each transition into RED adds 360 MHz at once, and the cap then drifts back down over the following windows as the bandit rediscovers headroom.

G.2 OFFERED-LOAD SWEEP

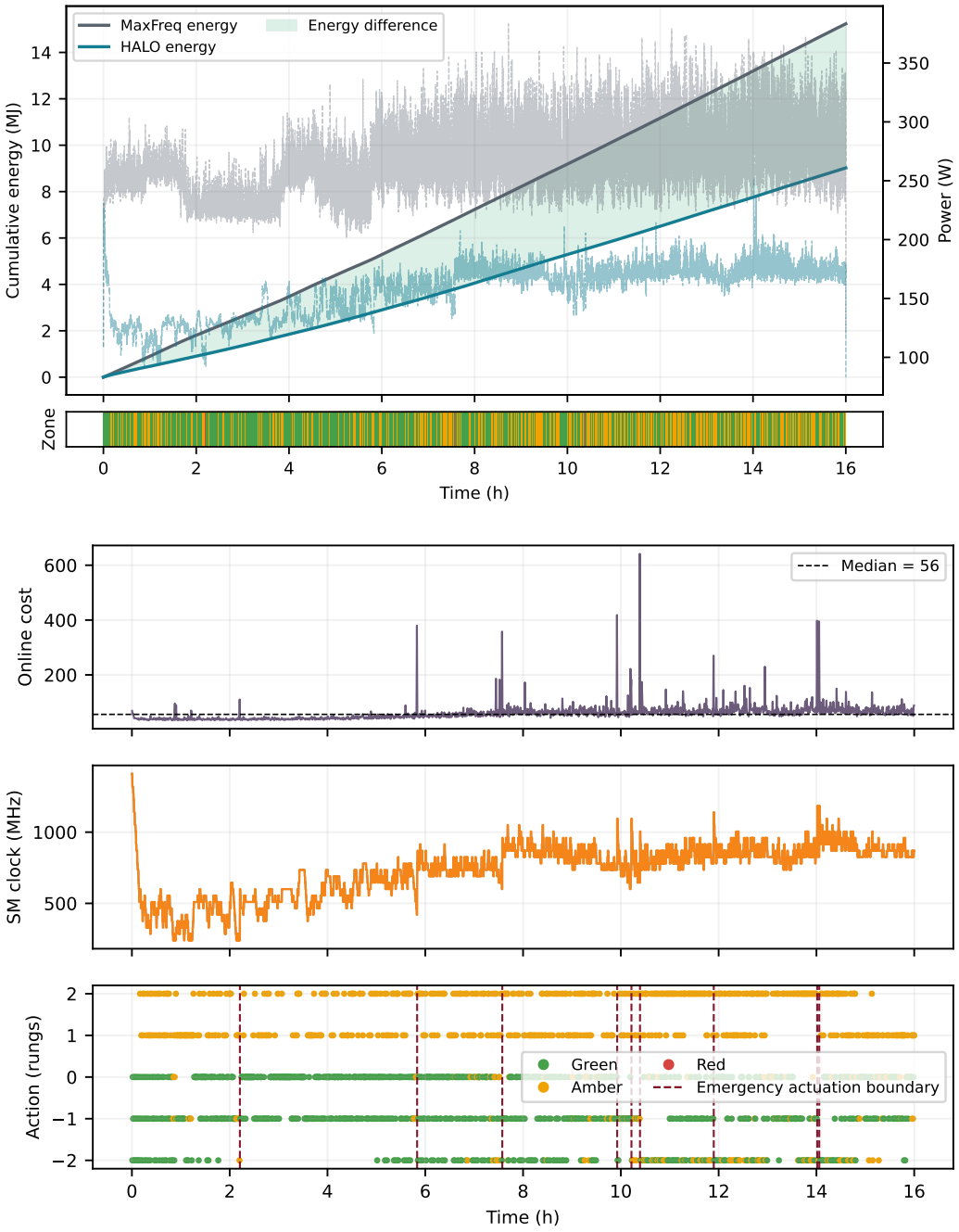

Figure 7: Mistral-7B, 0.75× speed, 10 s probe SLO (16-hour run). HALO uses 9,021 kJ against 15,235 kJ for MaxFreq (−40.8%), with 9 RED windows.

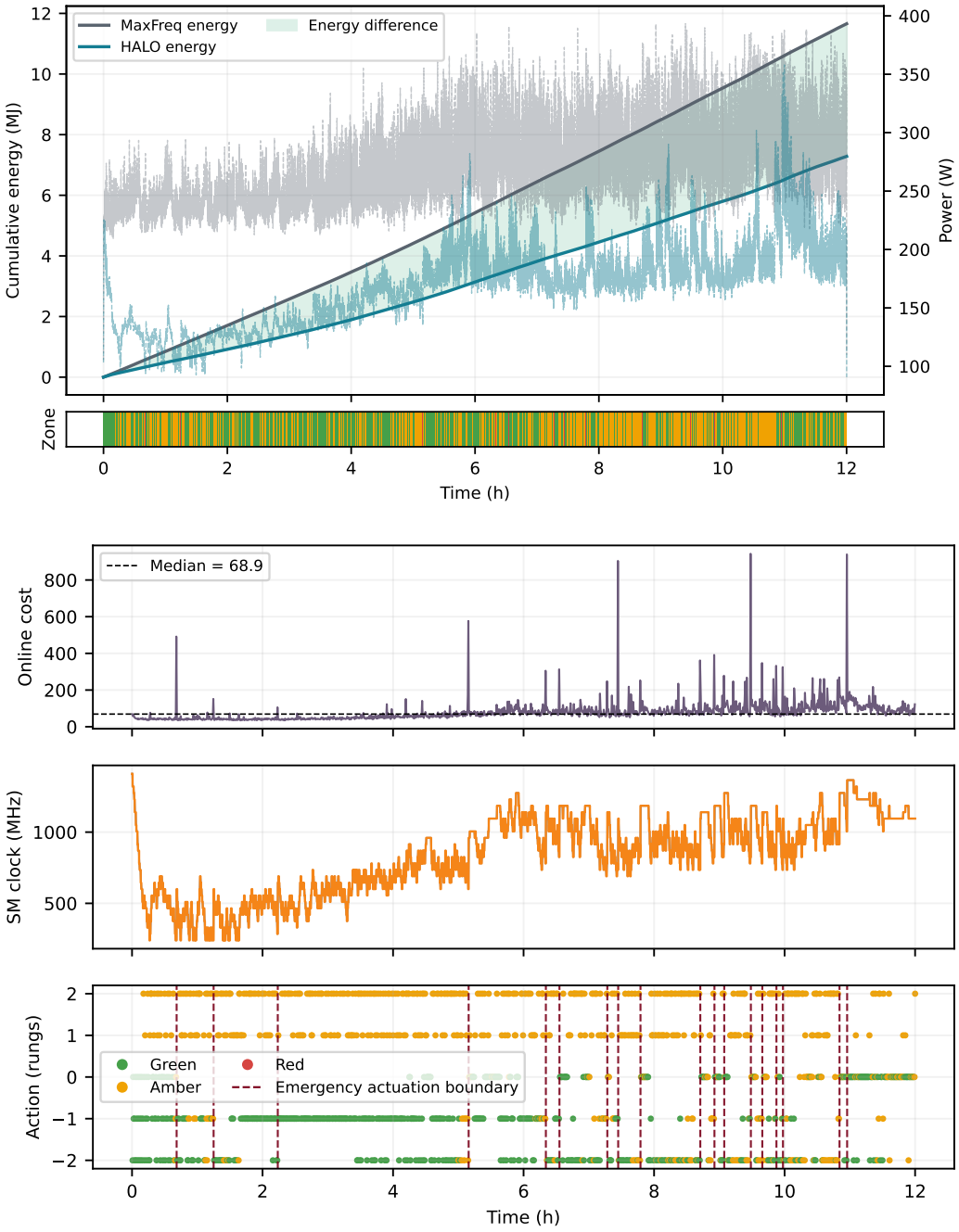

Figure 8: Mistral-7B, 1× speed, 10 s probe SLO (the default run, 12 hours). HALO uses 7,284 kJ against 11,659 kJ (−37.5%), with 18 RED windows.

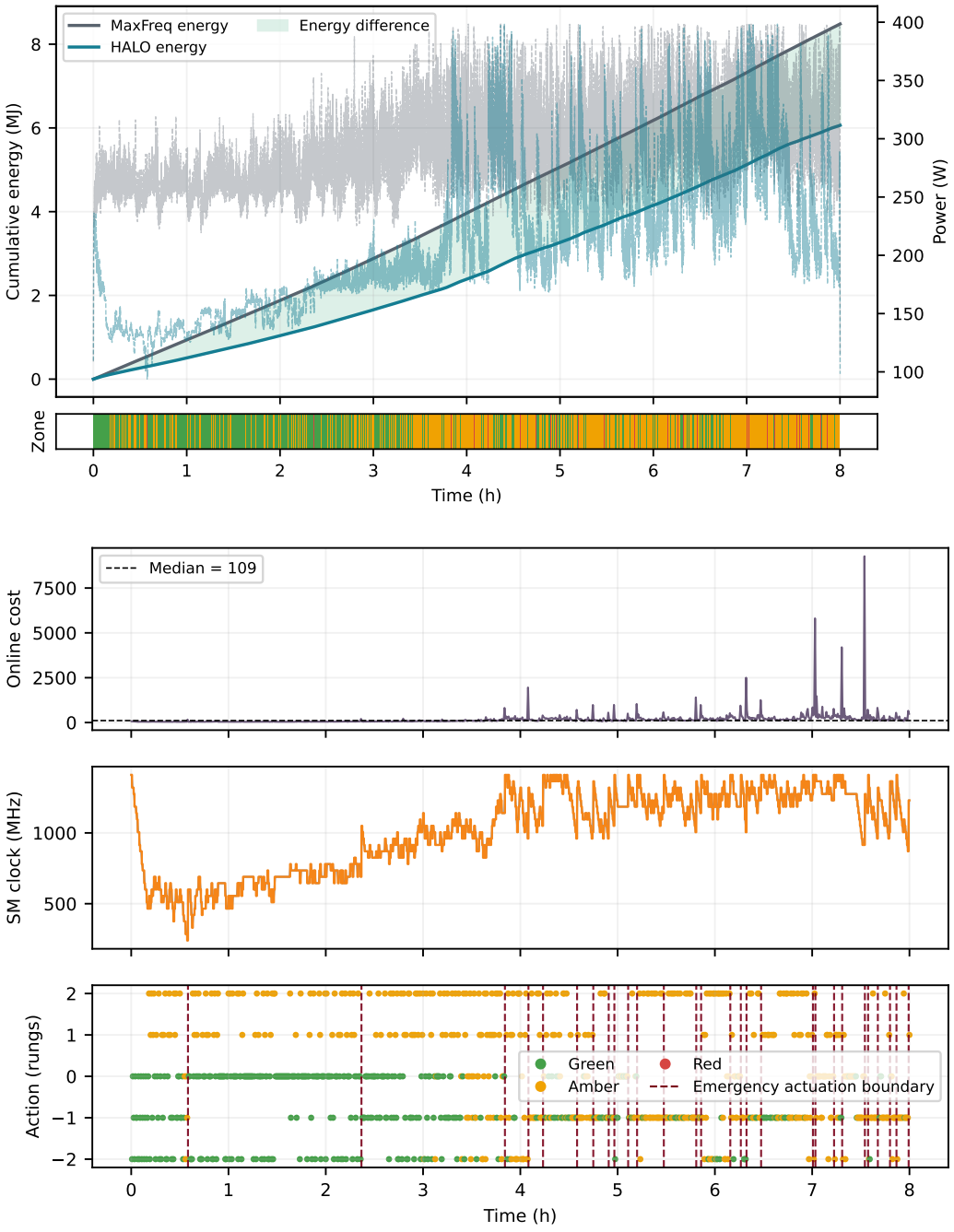

Figure 9: Mistral-7B, 1.5× speed, 10 s probe SLO (8-hour run). HALO uses 6,062 kJ against 8,484 kJ (−28.5%), with 28 RED windows against 2 for MaxFreq. The cap rises with the afternoon load and the emergency ladder-ups concentrate in the second half.

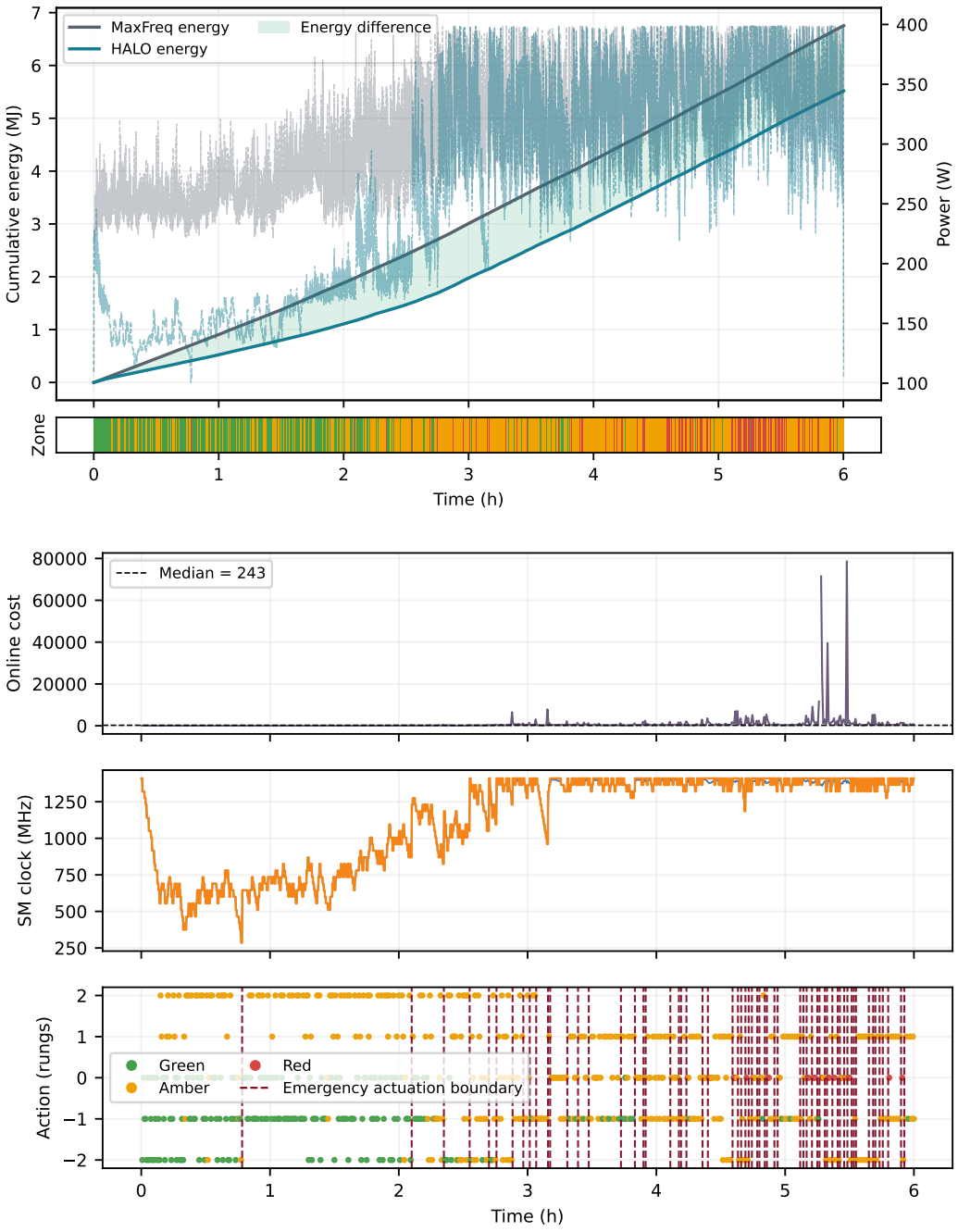

Figure 10: Mistral-7B, 2× speed, 10 s probe SLO (6-hour run). HALO uses 5,519 kJ against 6,753 kJ (−18.3%). The workload runs close to saturation, and MaxFreq itself has 71 RED windows. HALO holds the cap near the maximum through the sustained RED period after hour 3.5.

G.3 PROBE-SLO SENSITIVITY

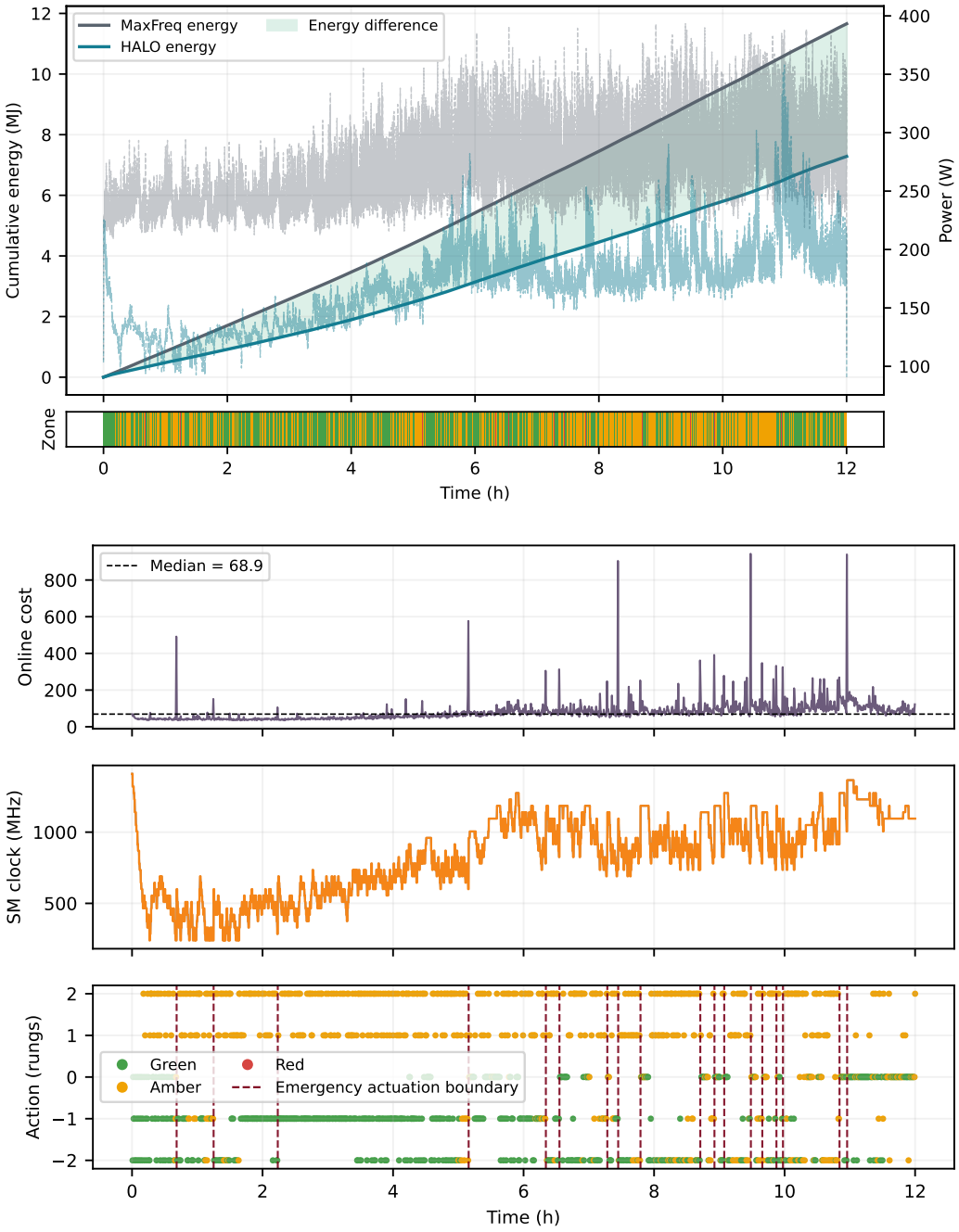

Figure 11: Mistral-7B, 1× speed, 10 s probe SLO (the default run).

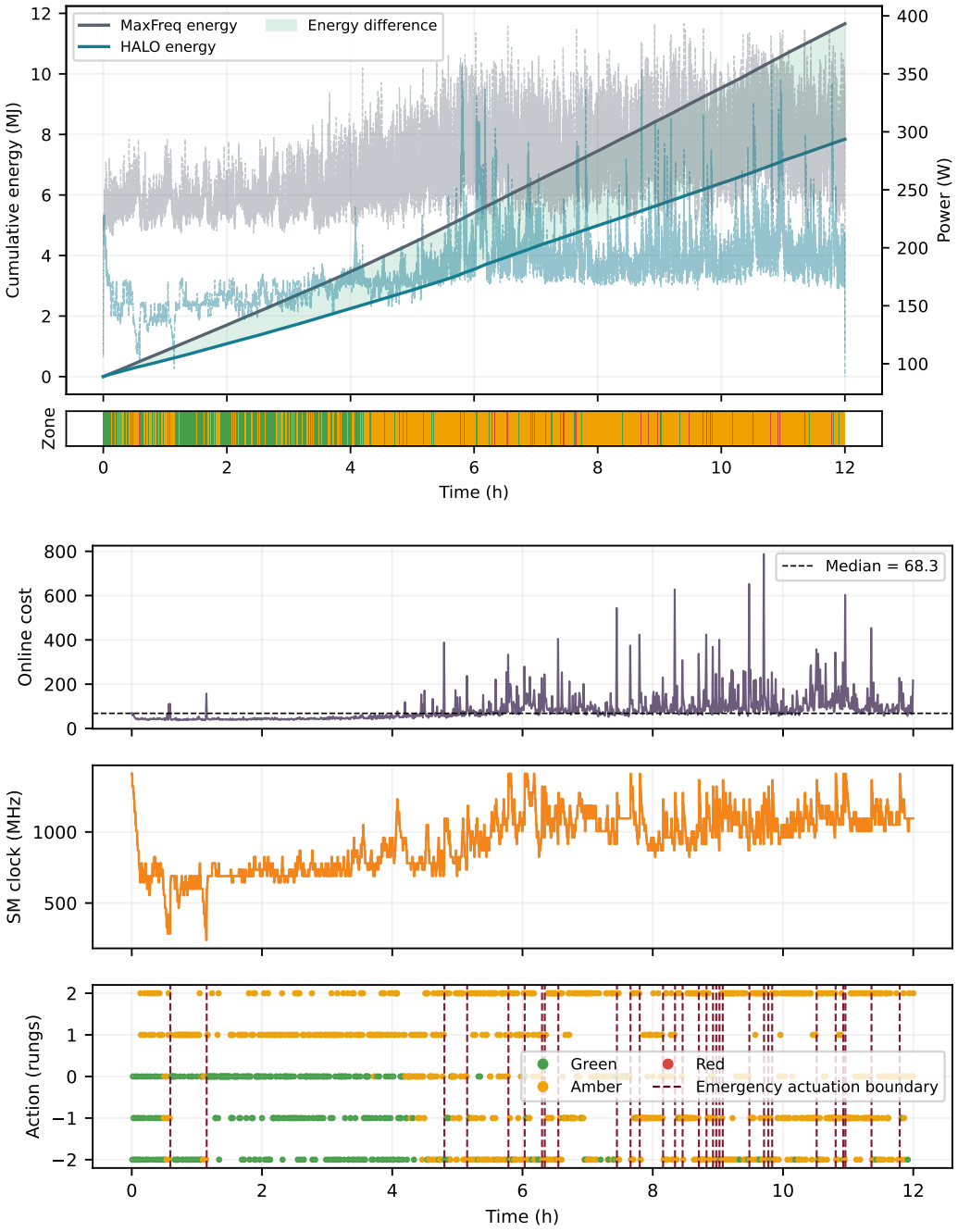

Figure 12: Mistral-7B, 1× speed, 7 s probe SLO. HALO uses 7,842 kJ (−32.7%) with 31 RED windows. Zones, penalty onset, and violation counts all use the 7 s threshold.

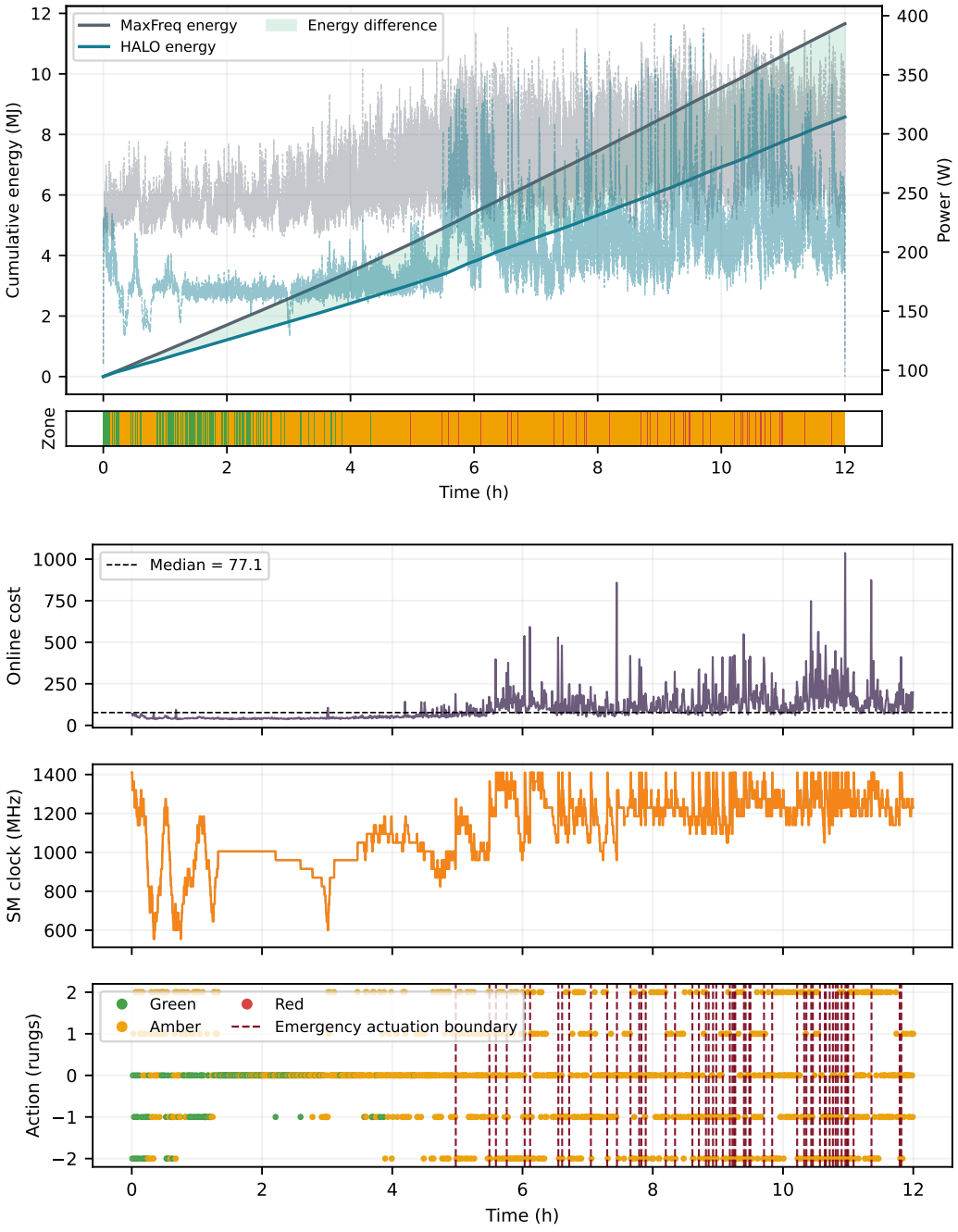

Figure 13: Mistral-7B, 1× speed, 5 s probe SLO. HALO uses 8,577 kJ (−26.4%) with 55 RED windows. With the tighter SLO the controller leaves the low-frequency band much earlier and spends most of the afternoon above 1,000 MHz.

G.4 CROSS-MODEL COMPARISON

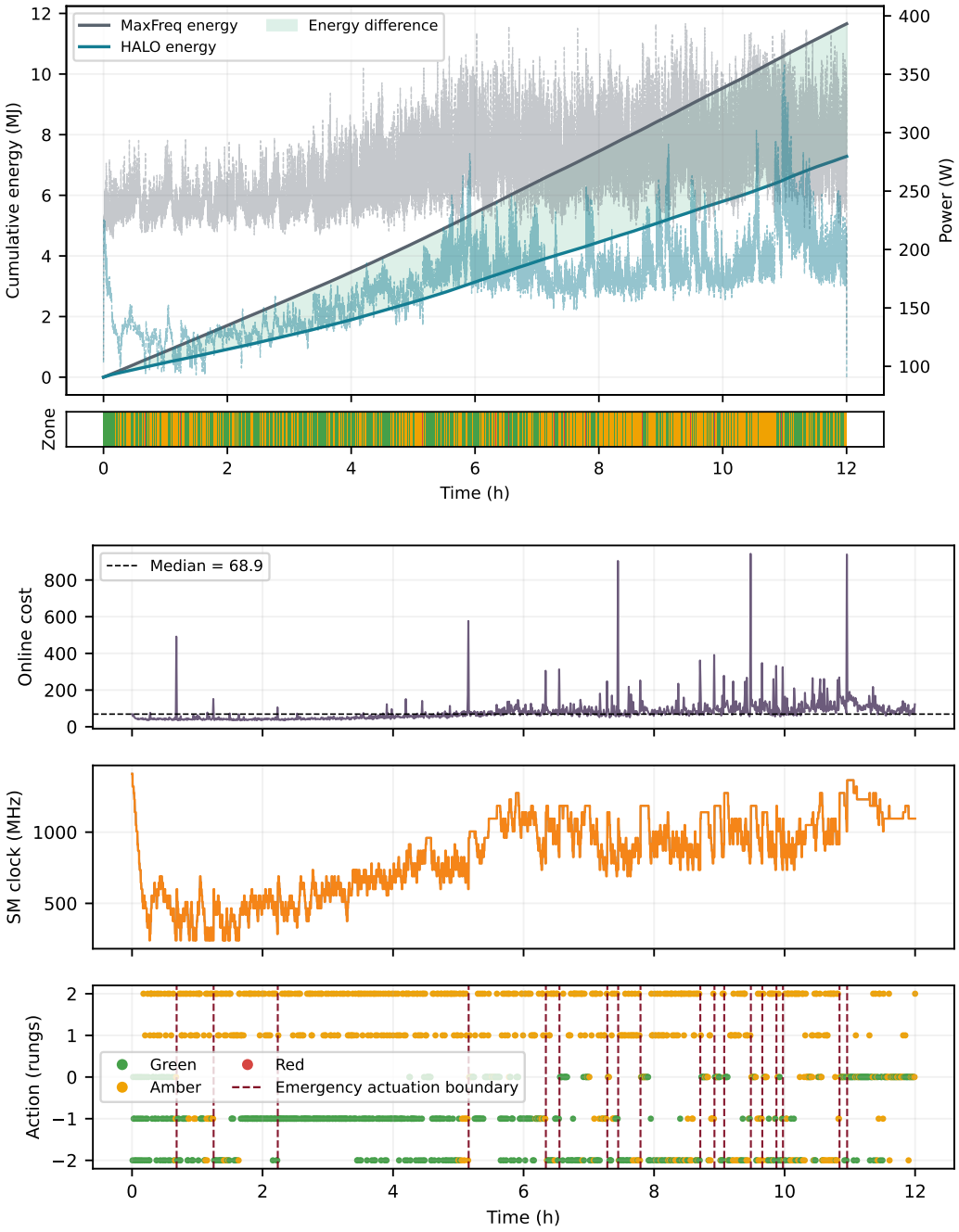

Figure 14: Mistral-7B-Instruct-v0.3, 1× speed, 10 s probe SLO (the default run).

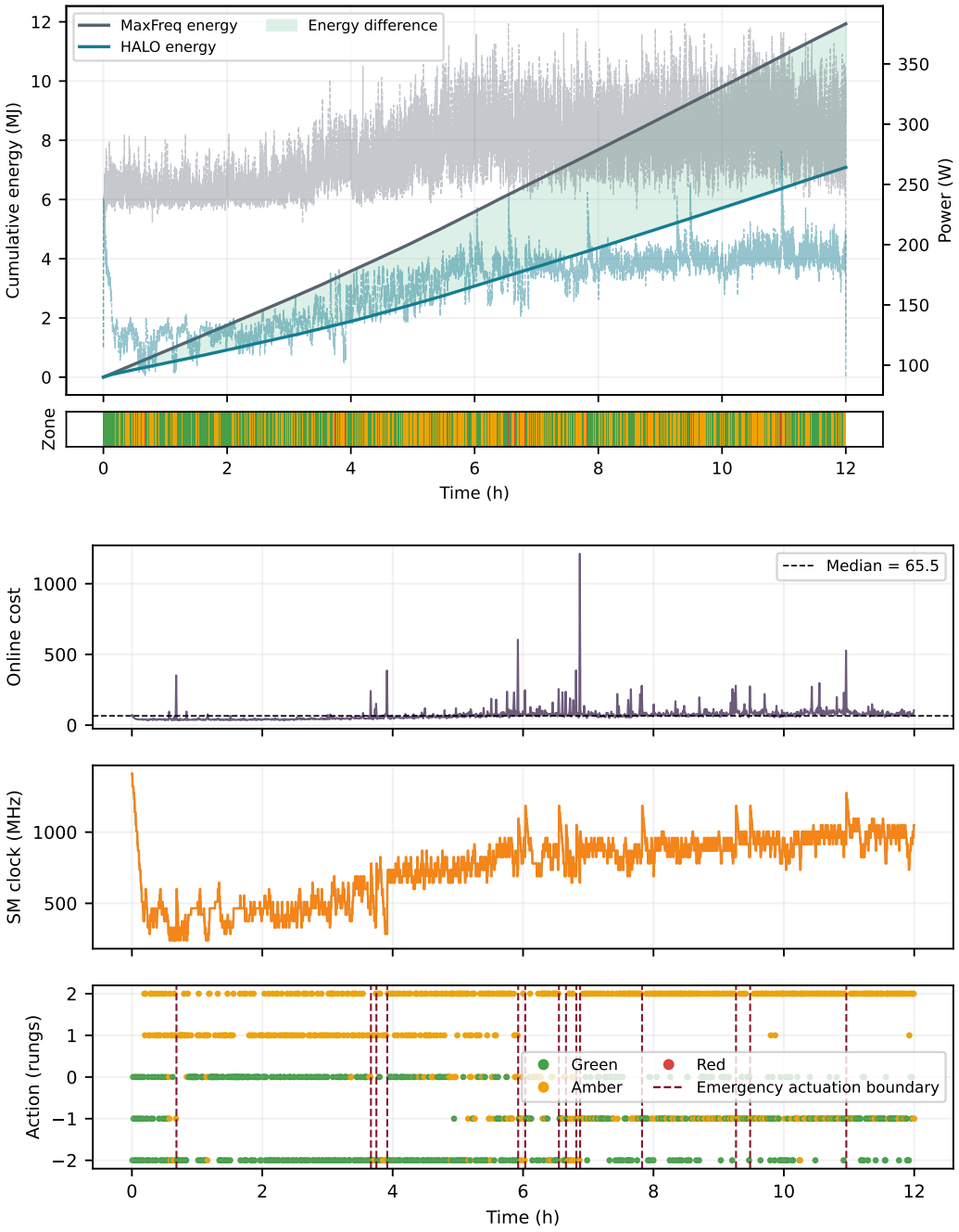

Figure 15: Qwen2.5-7B-Instruct, 1× speed, 10 s probe SLO. HALO uses 7,087 kJ against 11,936 kJ (−40.6%) with 14 RED windows. The controller behaves as on Mistral-7B without any retuning.

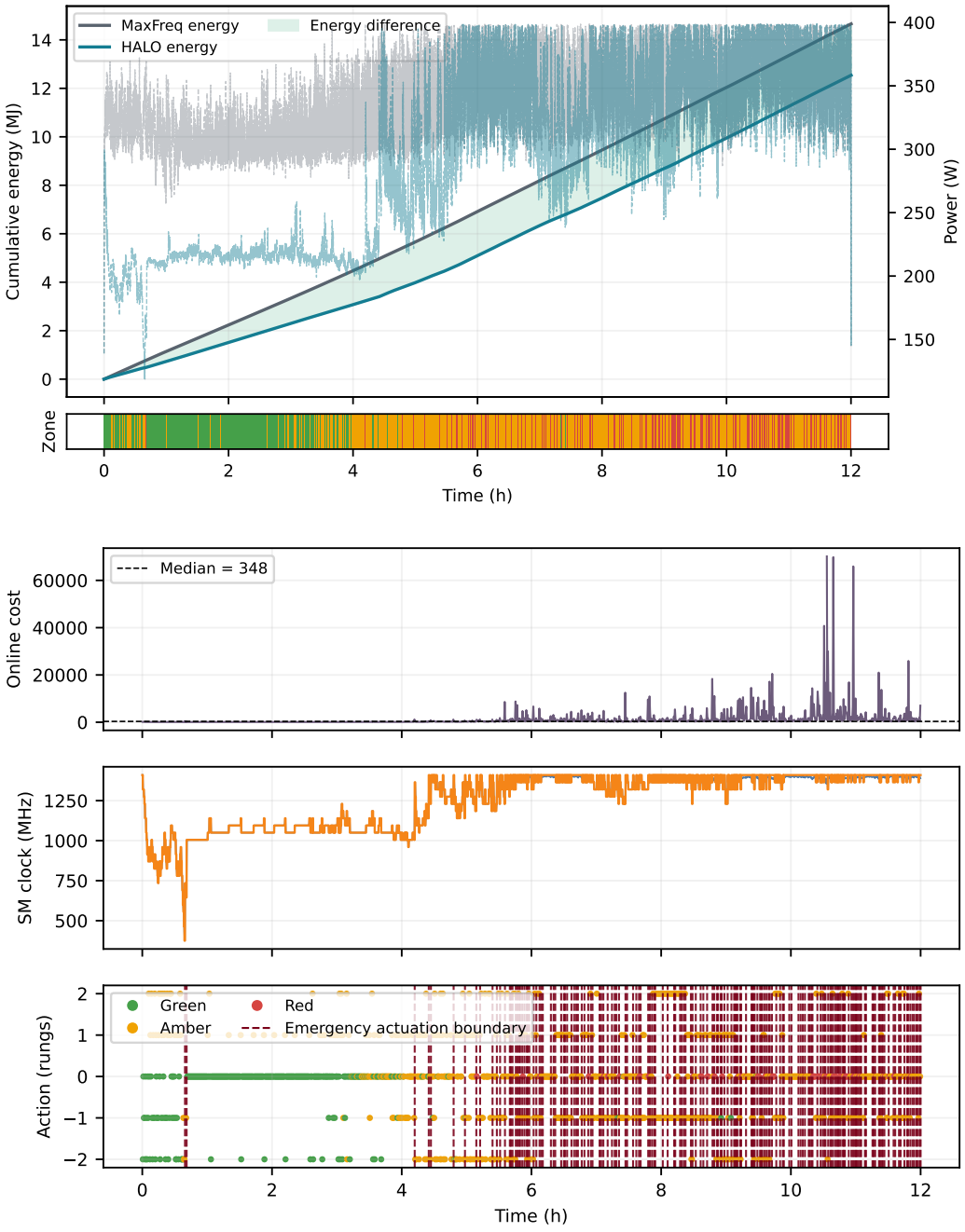

Figure 16: Qwen3-14B, 1× speed, 10 s probe SLO. HALO uses 12,541 kJ against 14,661 kJ (−14.5%). The 14B model saturates the GPU at this load. MaxFreq has 221 RED windows on its own, the savings come from the quieter first third of the slice, and the gate keeps the cap at or near 1,410 MHz for most of the afternoon.