

POWERSLIDER: Exploiting Phase Asymmetry for LLM Serving under Demand Response

Yueying Li[†], Jiayang Chen[†], Yuanfan Chen[†], Leo Han[†], Haoran Qiu[‡], Esha Choukse[‡], Rodrigo Fonseca[‡], Udit Gupta[§]

[†]Cornell University [‡]Microsoft Azure Research [§]Cornell Tech

Abstract—AI inference clusters are increasingly constrained by *instantaneous power*, not just energy: grid operators condition new capacity on demand response, imposing time-varying power caps. Existing LLM serving systems optimize a static energy objective or shed fixed priority tiers under load; either way, goodput collapses when the power envelope moves. An LLM pipeline is not a uniform load: compute-bound prefill loses throughput almost linearly with GPU frequency, memory-bound answer decode sustains it down to $\sim 0.57\times$ nominal, and reasoning’s *thinking* phase couples KV-cache capacity to scheduling – so a cap should be steered to where each watt costs the least performance. POWERSLIDER does so with a new Flex SLO contract that turns bounded user slack into an optimization constraint, prefill–think–answer disaggregation exposing per-stage frequency and KV control, and a Karush–Kuhn–Tucker (KKT) online solver re-solving within 7.7ms of every cap change, backed by a consolidated fail-safe that power-gates drained instances when DVFS bottoms out on static power. On SGLang with production traces, POWERSLIDER sustains 78.3% online goodput at a 30% cap reduction versus 47.6% for the best of five baselines ($1.64\times$), holds latency-critical tails within $1.3\times$ of nominal (baselines: 2.3–6 $\times$, up to 12 $\times$), and delivers 92% mean goodput through a replayed CAISO grid-emergency day bottoming at $0.41\times$ (54% at the trough; every baseline below 7%).

I. INTRODUCTION

The growth of generative AI has made data centers a first-order load on electric grids. The U.S. Department of Energy projects data-center electricity consumption to grow $1.84\times$ – $3.3\times$ between 2023 and 2028 [38], [67], and the median wait from interconnection request to commercial operation now exceeds five years [62]. As a result, grid operators increasingly condition new capacity on *demand response*: the ability to shed load during grid stress in exchange for faster connection and capacity credits [4], [16], [30], [55]. The potential is large; one study estimates that curtailing data-center load for 0.25% of uptime could free up to 76GW of U.S. grid capacity [53]. For an AI cluster, demand response arrives as a concrete runtime constraint: a *time-varying power cap* $p_{\max}(t)$ that the serving system must satisfy at every instant (Fig. 1). This is a fundamentally different problem from minimizing energy consumption. An energy objective and a power cap are not interchangeable: energy is an integral that deferral can satisfy, whereas $p_{\max}(t)$ binds at every instant, and the energy-optimal operating point can itself be cap-infeasible; on A100 GPUs, energy per LLM token is minimized at an intermediate frequency (Figure 4) that a deep power cap might forbid.

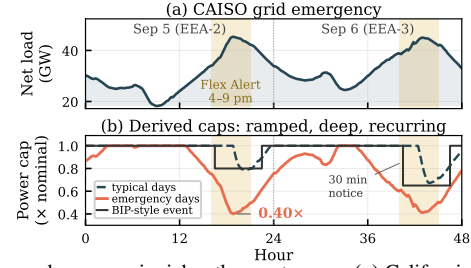


Fig. 1. Demand response is richer than a step cap. (a) California Independent System Operator (CAISO) net load during a grid emergencies; (b) Power caps under 5 min settlement interval in high-demand and typical day [18]. Black steps: base interruptible program (BIP) events, dispatched 30min after the alert, deeper on the more severe day [55]. EEA2/3 are grid emergency levels.

Meeting a moving cap takes two ingredients. The first is architectural: power reduction is not uniform across an LLM serving pipeline. Prefill phase is compute-bound and loses throughput almost linearly as GPU frequency drops. Answer decode phase is often memory-bandwidth-bound and can run at greatly reduced frequency with little throughput loss. Reasoning models further insert a long *thinking* phase whose accumulated KV-cache state couples memory capacity to scheduling. Hence, a demand-response event cannot be handled well by uniform power capping or by a statically profiled energy-optimal configuration; it requires a runtime that reallocates power across pipeline stages, service classes, and memory footprints as the cap changes. This paper shows that exploiting this *phase asymmetry* (in compute intensity, memory pressure, and DVFS sensitivity) is the key to sustaining goodput under deep power caps.

The second ingredient is knowing *where* performance may be given up. We observe that online inference carries significant latency slack that can be converted into power headroom. By serving part of the traffic under a *flexible* (Flex) tier that meets its base latency target most of the time but accepts bounded degradation during demand-response windows, the cluster gains room to modulate power without sacrificing any class entirely. Commercial APIs already expose this structure: Anthropic’s “Fast” versus “Standard” tiers trade up to $2.5\times$ response time for up to $6\times$ price [3], and Google’s Gemini “Flex” tier serves the same models at reduced cost in exchange for higher, variable latency [22]. The contract is therefore realistic; what is missing is a runtime that can honor it under a moving power envelope.

Key insight. A serving cluster’s power flexibility depends

TABLE I
COMPARISON OF PRIOR WORK TO POWERSLIDER

	Scope	Power-Aware	Reasoning	Dynamic Cap
PowerWeave [37]	GPU	✓		
Power Sloshing [10]	Server	✓		
PowerGrad / POLCA / Splitwise	Cluster	✓		
DynamoLLM [52], [56], [57], [65]				
SLOs-Serve [9]	Cluster		✓	
POWERSLIDER	Cluster	✓	✓	✓

on both system runtime and SLO contracts definition: the pipeline’s phase asymmetry (where shedding watts costs little throughput) and the Flex tier’s SLO tolerance (where delaying tokens costs little user harm). We propose POWERSLIDER, which applies this insight via a unified online optimization: it dynamically calculates both slacks to shed the watts that result in the least goodput degradation.

Existing systems address neither half. Energy-efficient LLM serving optimizes a *static* energy objective that can be found offline [64], [65], [71], [77], [78]. Multi-SLO serving optimizes latency and throughput across tiers but treats power as unconstrained [9], [82]. Power-oversubscription work caps power with fixed priority thresholds, without per-class guarantees, time-varying caps, or support for reasoning workloads [57]. Recent power-management work spans every scope below the grid, from spatial DVFS within a GPU [37] to budget redistribution within a server [10] and across clusters [52], but each optimizes under a fixed envelope; none takes a moving cap as a first-class input. Table I positions POWERSLIDER against the representative system in each category.

To our knowledge, POWERSLIDER is the first system to support dynamic power caps for online LLM serving with both non-reasoning and reasoning workloads. These gaps raise three problems, which §III examines with evidence: (1) how to expose bounded user slack without violating service guarantees, (2) how to search a GPU runtime configuration space efficiently, (3) how to handle reasoning with long, unpredictable thinking phase.

Motivated by these problems, we design POWERSLIDER, an architecture-aware power-capping runtime for LLM inference. Given a dynamic power cap, POWERSLIDER jointly controls per-stage GPU allocation, stage-aware DVFS, and KV-cache memory partitioning. A Flex SLO contract converts user-visible slack into an optimization constraint, and a prefill–think–answer (PTA) disaggregated pipeline gives the optimizer per-stage control over reasoning workloads. Because the configuration space shifts with every cap, POWERSLIDER replaces offline profiling with an online solver derived from the Karush–Kuhn–Tucker (KKT) conditions of a convex relaxation of the joint allocation problem. We implement POWERSLIDER atop SGLang [80] and evaluate it on Azure LLM inference traces and reasoning workloads under swept and time-varying power caps.

In summary, this paper makes the following contributions:

- **Characterization.** GPU LLM serving under DVFS is phase-asymmetric: compute-bound prefill loses throughput almost linearly in frequency, memory-bound answer decode holds

its throughput better than prefill, and reasoning’s thinking phase introduces a new memory-capacity constraint that couples KV-cache size to scheduling. §III quantifies this asymmetry and its implications for power capping.

- **Design.** POWERSLIDER converts that asymmetry, plus a calibrated (α_c, ρ_c) Flex contract, into a single runtime: PTA disaggregation exposes per-stage frequency and KV knobs, an impact metric turns contracted slack into a convex constraint, and a KKT-based solver re-solves the joint allocation problem with scalable efficiency.
- **Result.** On SGLang with production traces, POWERSLIDER makes the reductions grid programs actually request (5–20%, §II-A) essentially free, holding 100% goodput for non-reasoning workloads up to a 60% reduction, and absorbs deep caps that break every baseline: 78.3% versus 47.6% online goodput at a 30% reduction under high-load reasoning ($1.64\times$), and $\geq 98\%$ goodput through a replayed CAISO grid-emergency day whose cap bottoms at $0.41\times$ (§VII). The residual gap at the deepest caps is hardware, not scheduling: at $f_{\min}$, static power bounds what DVFS can shed, motivating the future power-gating direction.

II. BACKGROUND

A. Demand Response Imposes Dynamic Power Caps

Datacenters face power-modulation scenarios from millisecond-scale grid emergencies to week-long regional shortages [79]. Simultaneously, the push for sustainability drives datacenters toward renewables: wind and solar supplied a record 17% of U.S. electricity in 2025 and roughly 90% of new generating-capacity additions [68], [69]. Their variability imposes a time-varying power envelope the datacenter must respect. We focus on its most structured instance, **demand response** (DR): enrolled customers curtail load on request under capacity and reliability programs. The obligation is a committed reduction against a measured baseline—commonly 5–20% of load for C&I participants—over events of 2–4 hours [53]. Lead time varies more, from day-ahead scheduling to the 10-minute response required of demand resources in PJM’s Synchronized Reserve market [59]; PG&E’s BIP sits near the tight end: 15–30 minutes of notice for events of up to 6 hours (at most 10 per month), with the curtailment level set by a customer-elected firm service level at least 15% below peak demand [55].

Importantly, DR imposes an *instantaneous* cap $p_{\max}(t)$, not an energy budget: the serving system must stay under the envelope at every interval while maintaining SLOs. Real grids enforce this at fine granularity: U.S. real-time markets dispatch and settle every 5 minutes under FERC Order 825 [18], and up to 450MW of flexible compute load in ERCOT already follows this 5-minute dispatch as Controllable Load Resources [17]. Throughout, we express cap severity as a *power-cap reduction*: an $X\%$ reduction caps the cluster at $(100-X)\%$ of nominal serving power.

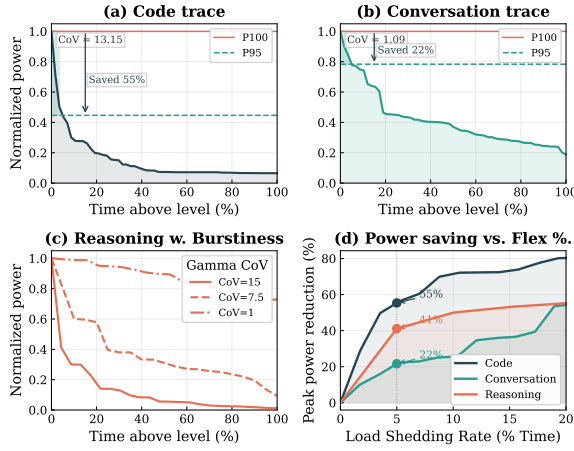


Fig. 2. Load-duration trends show that peak power demand is driven by tail behavior: a small amount of service flexibility (5%) substantially reduces required peak power provisioning. We sweep arrival burstiness (larger coefficient of variation gives more power headroom) and quantify how different load-shedding rates flatten the peak. This burst-driven slack is the raw material POWERSLIDER converts into demand-response headroom (O1).

B. LLM Serving Phases and Their Metrics

Modern LLM inference proceeds in two classical phases. The *prefill* phase processes the input prompt, computing attention over all input tokens in parallel; it is compute-bound with high arithmetic intensity. The *decode* phase generates output tokens autoregressively; it is memory-bandwidth-bound, dominated by loading large KV-cache tensors. This asymmetry motivated prefill–decode (PD) disaggregation [56], [81], which serves the two phases on separate GPUs. Production serving systems build on this structure, from vLLM [36] and Sarathi-Serve [2] to multi-SLO schedulers such as SLOs-Serve [9] and PolyServe [82]; all optimize throughput and latency, and none treats instantaneous power as a first-class constraint.

Reasoning adds a third phase. Reasoning-capable LLMs (e.g., DeepSeek-R1 [24], OpenAI o1/o3) generate an extended *thinking* chain of internal tokens before the user-visible answer. Thinking tokens are autoregressive like decode but with far longer sequences (often 10–100× the answer length), making thinking phase *memory-capacity*-intensive in addition to bandwidth-bound. This introduces a new latency metrics: *time-to-first-answer-token* (TTFAT), the delay until the first visible output (prefill plus the entire thinking chain), which is the primary user-facing SLO for reasoning workloads; *time-between-answer-tokens* (TBAT), the inter-token latency of visible answer tokens; and *time-to-last-token* (TTLT), the end-to-end completion latency.

III. MOTIVATION

This section establishes three observations that together define the design space. Each observation ends with the design element it motivates; §IV opens with the full mapping.

A. Observation 1: Online Traffic Carries Bounded Slack that Rigid Tiers Cannot Convert into Headroom

Online LLM demand contains latency slack that can be converted into power headroom. Figure 2 runs open-source non-reasoning and reasoning request traces with different arrival

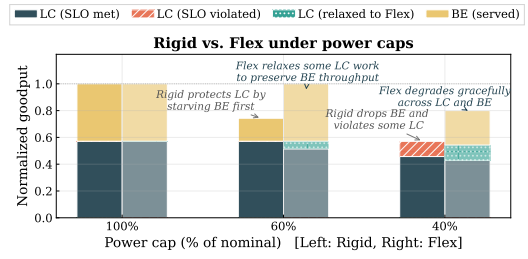


Fig. 3. **Flexible SLOs enable graceful degradation across power-cap reductions.** At full power both paradigms deliver the same total goodput. As the cap tightens, the rigid policy protects LC requests by rejecting BE traffic and still incurs LC violations; the flexible-SLO policy allows controlled LC degradation, preserving BE service and higher overall goodput. Rigid tiers waste the slack of under a cap; Flex tier exists to spend it.

patterns on a cluster simulator and convert the SM / memory utilization into GPU power with profiling-based power and performance models [49], [74]. We find that *peak* power is set by rare bursts rather than sustained demand – a tail that survives statistical multiplexing (the traces are cluster-level aggregates), while a cap can bind even below the average draw (Fig. 1): allowing a small fraction of requests to experience controlled degradation, reduces the peak substantially.

However, exploiting this slack requires more than relaxing SLOs. Under a rigid two-tier SLO policy (latency-critical and best-effort), the only lever when the cap tightens is shedding load: best-effort traffic is starved first, and once the cap is deep enough, latency-critical (LC) requests violate their targets as well. Figure 3 makes this concrete at 0%, 40%, and 60% power-cap reductions on A100 serving a Llama-70B model: the rigid policy protects LC by rejecting best-effort (BE) traffic entirely, yet still incurs LC violations at the 60% reduction, sharply reducing total goodput. A *Flex* tier that accepts bounded, intermittent degradation breaks this dynamic and improves goodput for both LC and BE.

The challenge, however, is not that some users can tolerate latency inflation; commercial APIs already sell such tiers [3], [22]. The challenge is *calibration*: enforcing and mapping a contract for SLO (e.g., at most $\alpha \times$ the base latency, for at most a ρ fraction of the time) into a runtime resource-allocation decision under a changing power cap.

Implication: the system needs a Flex SLO contract that is simultaneously meaningful to users and usable as an optimization constraint (§IV-B).

B. Observation 2: Uniform Power Capping Wastes the Stage Asymmetry of GPU DVFS

GPU frequency scaling does not affect all serving phases equally. Figure 4 profiles CodeLlama-34B serving on A100-80GB across GPU frequencies (210–1410 MHz) and batch sizes. We observe two regimes. At small batch sizes (≤ 128), throughput saturates above ~ 810 MHz ($0.57 \times$ nominal); at large batch sizes (≥ 256), it scales nearly linearly up to the maximum 1,410 MHz. A roofline analysis explains the split and why the knee shifts with different batch sizes. The A100 can sustain ≈ 156 FLOP per byte of HBM traffic (312 FP16 TFLOP/s over 2.0 TB/s); a kernel below this ridge point is

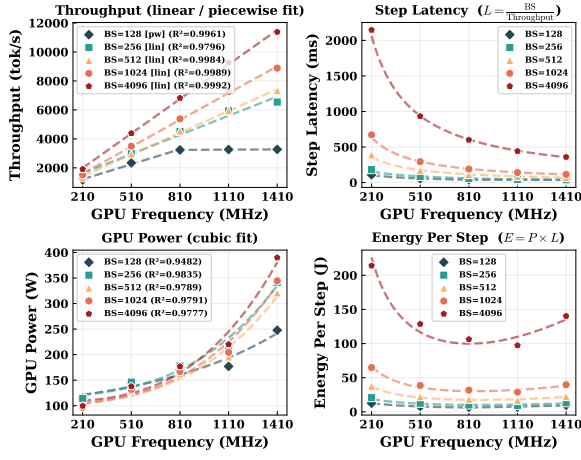


Fig. 4. Measured throughput, power, energy, and latency vs. GPU frequency for five batch sizes (BS) on A100-80GB. The frequency at which throughput saturates shifts with batch size, and $BS < 128$ (memory-bandwidth-bound) saturates within the DVFS range, above 810 MHz, saturation happens lower for lower BS; from $BS > 128$ (compute-bound) the knee lies beyond the maximum 1,410 MHz, so any frequency decrease costs a proportional throughput drop. POWERSLIDER exploits this differential: it reduces batch-constrained answer-stage frequency first, shedding power at minimal performance cost. This stage and batch-related asymmetry is the hardware lever POWERSLIDER’s solver pulls whenever the cap moves (O2).

limited by memory bandwidth, one above it by compute throughput. Decode sits far below the ridge: each decode step performs roughly one multiply-accumulate per weight byte for every request in the batch, so a batch of B has an arithmetic intensity of only $\approx B$ FLOP/byte, and charging the KV-cache reads lowers it further. Decode throughput is therefore set by HBM bandwidth, which core DVFS does not change; the core clock must only be fast enough to keep the memory pipeline full, which on the A100 is the ~ 810 MHz knee, so frequency above the knee is wasted on decode and frequency below it starves the memory system. Prefill is the opposite case: its multi-thousand-token GEMMs sit well above the ridge, so throughput is set by compute and falls in proportion to f . Mapped onto the serving pipeline, prefill follows the compute-bound curves, answer decode at small batch the memory-bound curves, and thinking falls in between. Concurrent work reports the same prefill/decode frequency-sensitivity split on B200-class GPUs [37], indicating the asymmetry is a property of the workload rather than of one hardware generation. Uniform power capping, which applies the same frequency to every GPU, therefore pays the *worst-case* cost: it slows compute-bound prefill, where each MHz costs throughput, to save power that memory-bound decode could have given up at little performance cost.

Exploiting this asymmetry requires jointly choosing batch size, frequency, and GPU allocation per stage and SLO class, with reoptimization at every cap change (Table II), since the saturation knee moves with batch size and batch formation depends on GPU allocation and KV chunk size – each request’s per-stage KV-cache reservation, in tokens.

This makes offline profiling impractical: with 9 stage-class groups (3 classes $\times$ 3 stages), each choosing among ~ 10 GPU-count options and ~ 9 discretized frequency/chunk levels,

TABLE II
THE PER-CAP CONFIGURATION SPACE POWERSLIDER MUST NAVIGATE.

Dimension	Examples
Stage	Prefill / Think / Answer
SLO class	LC / Flex / BE
GPU allocation	k_i GPUs per stage-class
Frequency	210–1410 MHz per stage-class
KV chunk size	e.g., 512 / 1024 / 2048 tokens
Model / TP setting	Llama-70B, CodeLlama-34B, Qwen-32B; TP-4

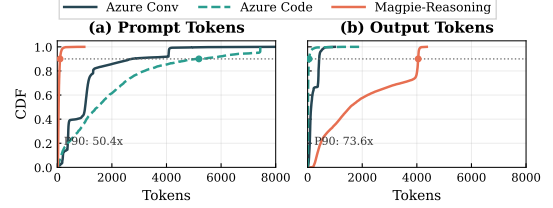


Fig. 5. Reasoning workloads have shorter prompts (left) but far longer, heavier-tailed output distributions (right), defeating the output-length prediction that prior resource-allocation and routing schemes depend on. These tails are why POWERSLIDER admits reasoning work by observation rather than by length prediction (O3), which matters in the near-saturation regime.

the joint space is $(10 \times 9)^9$ configurations, and the optimum shifts with every new cap and workload mix, so no pre-profiled grid can be dense enough. Prior systems avoid this problem rather than solve it: DynamoLLM [65] pre-profiles configurations for a single static power target and POLCA [57] applies threshold-based frequency capping; neither tracks a moving cap (§VIII).

Implication: A dynamic power cap with various knobs across the software-hardware stack turns power allocation into an online control problem: the stage-aware optimum must be derived efficiently at event rate (§IV-C) and actuated across different granularities (§V).

C. Observation 3: Reasoning Workloads Break PD Disaggregation and Prediction-Based Routing

A power cap pushes the cluster toward saturation, where reasoning workloads break two key mechanisms: PD disaggregation and prediction-based routing.

Reasoning workloads stress the memory system asymmetrically. Long-lived thinking chains occupy KV-cache memory for thousands to tens of thousands of tokens; when thinking and answering share the same decode workers, the accumulated thinking state caps the achievable batch size and head-of-line blocks short decode sequences. We can bound this pressure in bytes. On Llama-70B, each token pins 320 KiB of KV state, so a 4k-token thinking chain holds 1.3 GB of HBM for its lifetime. A TP-4 instance with ~ 180 GB of post-weight HBM capacity therefore saturates at roughly 140 such concurrent thinking-stage requests, a *memory-capacity* limit on decode concurrency. Figure 14 (Appendix E2) quantifies the cost: at a think-to-answer (T/A) ratio of 2:1 and 7 req/s, P90 TBAT reaches 57ms under PD versus 28.3ms with three-stage separation (a 51% reduction), and the gap widens monotonically with the T/A ratio (71% at 16:1) as longer thinking chains thrash the shared KV cache.

TABLE III
POWERSLIDER CONTROL COMPONENTS

Component	Cadence	Crit. path?	Overhead
PSRoute (dispatch/admission)	per request	yes	virtual-queue lookup
PSSched: pool reallocation	5 min + 10 s refine	no	drain-bounded
→ vote-commit DVFS	50–100 ms	no	one NVML call/GPU
PSOpt (KKT solve)	DR event / hourly	no	under 100 ms

Prediction-based routing fails under reasoning workloads. Reasoning output lengths are heavy-tailed (Figure 5; the 90th percentile reaches 4050 tokens, up to $74\times$ non-reasoning workloads). A DistBert-style length predictor as used in DynamoLLM [65] misroutes 37.1% of requests overall after fine-tuning to reasoning traces. Among these, 26.6% of Medium and 16.3% of Long requests land in the Short pool, inflating its load by 25%. Due to the queuing effect, under high load or saturation, this inflates P99 TTFT by over an order of magnitude through routing-induced queueing (details in Appendix E1). This failure mode applies broadly: any pre-sized-pool allocation fails when decode-length variance is high. Power caps exacerbate the problem by pushing the system into near-saturation state where mispredictions cascade into SLO violations.

Implication: reasoning workloads require two mechanisms: (i) stage-level isolation of thinking phase (PTA disaggregation) to prevent KV-cache thrashing, and (ii) admission control that adapts to observed output lengths rather than predicting them (§IV-A).

IV. POWERSLIDER DESIGN

POWERSLIDER is an online, scalable and dynamic architecture-aware power-capping runtime. When the grid operator lowers a cluster’s power cap (e.g., by 20–40%), POWERSLIDER decides which stages and SLO classes to slow down, by how much, and with which control knobs. Figure 6 summarizes the architecture.

POWERSLIDER comprises an optimizer, **PSOpt**, and two runtime actuators, **PSSched** and **PSRoute**; Table III summarizes each component’s cadence and critical-path role. The section follows the control loop of Figure 6: §IV-A presents the pipeline substrate; §IV-B and §IV-C present PSOpt – the contract interface that turns bounded slack into a constraint, and the solver that exploits O2’s asymmetry; and §IV-D shows how PSSched and PSRoute actuate the solution.

A. Pipeline Substrate: PTA Disaggregation

Observation 3 (§III-C) showed that reasoning workloads defeat both PD disaggregation and prediction-based routing. POWERSLIDER addresses the first with the PTA pipeline below; the second is answered by PSRoute’s adaptive admission, an actuator policy (§IV-D). PTA is the substrate the rest of the design manipulates: it defines the stage pools and the per-stage variables (f_i , CS_i , k_i) the optimizer ranges over – without stage isolation there is nothing stage-aware to optimize.

PTA disaggregation. POWERSLIDER extends PD disaggregation into the three-stage Prefill–Think–Answer pipeline, where each stage runs on a dedicated GPU pool. Thinking tokens are

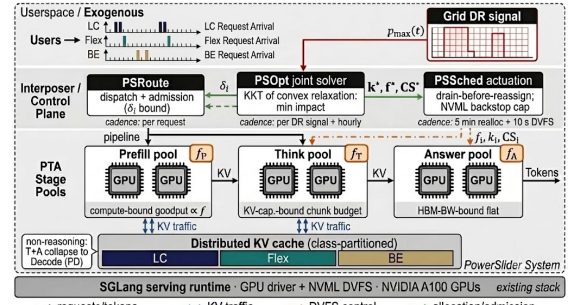


Fig. 6. Architecture overview of POWERSLIDER. Dark-headed boxes are POWERSLIDER’s components; the gray bar is the existing stack (SGLang runtime, GPU driver, NVML DVFS). Three SLO classes flow left-to-right through the Prefill–Think–Answer pipeline; each pool forms its own frequency domain (f_P , f_T , f_A), and the KV cache is partitioned per class (colors match the user chips). PSOpt turns the grid cap $p_{max}(t)$ into per-stage allocations, frequencies, and KV budgets; PSSched actuates them; PSRoute admits and dispatches per request.

generated on T-workers, isolating them from both the compute-bound prefill pipeline and the memory-capacity-bound answer decode path. This stage isolation has two benefits. First, each stage reserves its own chunk size (CS_T for think, CS_A for answer), whereas PD must unify the values; at typical parameters this gives PTA’s think instances 44% higher batch capacity than PD. Second, PTA enables independent, per-stage knobs such as frequency f_i : it decouples the frequency and chunk size of the two decode-like stages and exposes the stage asymmetry (Observation 2) to the solver.

B. PSOpt Interface: The Flex Contract

Central to POWERSLIDER is serving three distinct classes of requests. Latency-critical (LC) requests must meet strict per-workload latency targets, typically $O(1\text{--}10\text{ s})$ TTFAT and $O(10\text{ s of ms})$ TBT. Flexible (Flex) requests tolerate bounded latency degradation, formalized by the following contract.

Definition 1 (Flex SLO Contract (α_c, ρ_c)). A Flex request class c with base latency target L_c is governed by a pair (α_c, ρ_c) with $\alpha_c \geq 1$ and $\rho_c \in [0, 1]$. The operator may serve class- c requests at a latency of up to $\alpha_c L_c$ for at most a ρ_c fraction of the time during a demand-response event, and must meet L_c otherwise. The contract is satisfied when

$$\Pr [L_c(t) > \alpha_c L_c] \leq \rho_c$$

over the event horizon, where $L_c(t)$ is the realized latency of class c at time t .

For example, $\alpha_c=2$ and $\rho_c=0.4$ allow up to $2\times$ latency for at most 40% of a demand-response event. Best-effort (BE) requests have no per-request latency target, but must finish within 24 hours and can be deferred. This differs from threshold schemes such as POLCA [57], which cap low-priority work at fixed frequencies with no per-class guarantees or calibrated cap-to-degradation mapping. The (α_c, ρ_c) contract turns a hard SLO into a tunable probabilistic one. Its parameters trade headroom, not safety: they bound what class- c users can experience (enforced at runtime, §V) while setting how much power headroom the operator gains. This calibrated flexibility, rather than SLO relaxation alone, lets the system absorb power-cap reductions with limited goodput loss.

The (α_c, ρ_c) contract maps to a smooth impact bound: for each stage-class, there is a throughput threshold below which overload probability exceeds ρ_c . $I_i(C) = (1/\bar{\lambda}_i) \int_{C_0}^C p_i(\lambda) (\lambda - C) d\lambda$ (with C_0 the current throughput, p_i the arrival-rate distribution, and $\bar{\lambda}_i$ its mean).

C. PSOpt Engine: Formulation and KKT Solver

With the contract mapped to per-stage impact bounds and the knobs of §IV-A in hand, we formalize the optimization POWERSLIDER solves at each demand-response event.

System model. For each stage-class i , serving capacity depends on whether the stage is compute- or memory-bottlenecked. Compute-limited throughput scales with GPU count k_i (as $k_i^{\nu_i}$, where $\nu_i \leq 1$ is the measured multi-GPU scaling exponent) and frequency f_i , modulated by the compute-bound fraction $\gamma_i \in (0, 1)$: prefill is highly compute-bound (high γ_i), answer decode at small batch has low γ_i , and thinking falls in between (derivation in Appendix B2). Memory-limited throughput is set by how many concurrent requests fit in KV cache at the current chunk size, divided by mean service time. Effective capacity cap_i is the minimum of the two.

Power and throughput models. The form of the power model follows from device physics. Dynamic CMOS power scales as $P_{\text{dyn}} \propto C_{\text{eff}} V^2 f$ [28], [50], and on the GPU’s DVFS ladder voltage rises roughly affinely with frequency ($V \approx V_0 + \beta f$) [33], [39], which expands to a cubic in f (the expansion’s f^2 cross-term is small over the operating range and is absorbed into the fitted coefficients), the form established GPU power models validate [27], [40]; static leakage plus the board/HBM idle floor contribute the frequency-independent term [7]. Per-GPU power is therefore $P(f) = c_2 f^3 + c_1 f + c_0$, with coefficients profiled per model and batch-size breakpoint (32–4096 tokens) from SGLang [80] serving different model families like Llama-70B, and Qwen-32B with different TP (the characterization of Figure 4 is one of it); the fits achieve $R^2 > 0.95$ across profiled points. Two architectural consequences follow. Inverting $P(f)$ maps a per-GPU power budget to an achievable frequency floor, which lets PSOpt translate $p_{\text{max}}(t)$ into per-stage frequency ranges. The constant term c_0 also limits how far DVFS can reduce power: below roughly $c_0 \cdot N_{\text{GPU}} (\approx 40\%$ of nominal on our testbed) further savings require consolidating work onto fewer GPUs, so PSOpt jointly controls k_i and f_i .

Each hardware type needs its own cubic fit; fits are profiled offline, and since serving produces fresh (frequency, batch, latency, power) tuples each DVFS window, they can be refit online when observed latency drifts.

Joint formulation. At each decision epoch with power cap $p_{\text{max}}(t)$, POWERSLIDER solves:

$$\min_{\mathbf{k}, \mathbf{f}, \mathbf{CS}} \sum_i w_i I_i(C) + \kappa \|\Delta \mathbf{k}\|_1 \quad (1)$$

subject to the power cap $P \leq p_{\text{max}}(t)$, per-stage-class impact bounds $I_i \leq \delta_i$, and global GPU budget $\sum_i k_i \leq K$. The objective is a weighted combination of user-visible impact, and GPU-reallocation churn; the weight w_i encodes the priority of

stage-class i ($w_{\text{LC}} \gg w_{\text{Flex}} \gg w_{\text{BE}}$). The decision variables – GPU allocation ($\mathbf{k}$), frequency ($\mathbf{f}$), and chunk size ($\mathbf{CS}$) – are deeply coupled and must be optimized jointly.

Convex relaxation and KKT-guided solution. The joint space of Table II spans roughly 10^{18} configurations (§III-B), so exhaustive search, online or offline, is impractical. Instead, POWERSLIDER solves a *convex relaxation* of the per-epoch problem: integer GPU counts $\mathbf{k}$ are relaxed to continuous values, and frequencies and chunk sizes are treated on their continuous ranges. On the relaxation, every objective term is convex: impact satisfies $\partial^2 I_i / \partial C^2 = p_i(C) / \lambda_i \geq 0$; the churn term is a norm; and the budget is linear. The KKT conditions [5], [32], [34] are therefore necessary and sufficient *for the relaxation*. This is why POWERSLIDER solves via KKT rather than a general-purpose solver: necessity and sufficiency mean the closed-form conditions *are* the optimum, so each re-solve evaluates analytic gradients instead of iterating a generic convex program or integer solver, and completes in 7.7 ms independent of cluster size (§VII-F), fast enough to track every cap change online. At the KKT stationary point, each stage-class balances the marginal watts saved by lowering f_i against the marginal impact incurred (Appendix B1). The ranking key is the impact incurred per watt reclaimed, i.e., the impact gradient of Eq. (4) over the power gradient,

$$R_i = \frac{(w_i + \eta_i) |\partial I_i / \partial \text{cap}_i| \gamma_i k_i^{\nu_i - 1}}{f_{\text{max}} (3c_2 f_i^2 + c_1)}, \quad (2)$$

and PSOpt lowers frequency in ascending order of R_i ; this sequence is the *degradation ordering*. Memory-bound stages (low γ_i) shrink the numerator, and classes with slack impact bounds have $\eta_i=0$ by complementary slackness, so the solver reclaims the cheapest watts first. This ordering restates the hardware characterization in optimization terms: γ_i is the compute-bound fraction measured in Figure 4, so in effect the solver sheds watts from the most memory-bandwidth-bound stage of the most latency-tolerant class first.

In a representative workload under a deep cap, the ordering plays out accordingly: answer decode drops onto the flat region of its throughput curve, thinking settles somewhat higher, and LC prefill stays at nominal frequency. Uniform capping instead pulls all three stages down together, paying full throughput cost on prefill for watts that decode could have given up at little cost. **PSOpt** evaluates these conditions directly and then *projects* the relaxed solution onto discrete GPU counts, DVFS states, and chunk-size levels (full derivation in Appendix B). The projection means the deployed configuration is near-optimal for the discrete problem; the resulting degradation ordering is nonetheless the exact optimum of the relaxation, not a heuristic, and §VII-F shows the full solve completes under 10 milliseconds.

D. PSSched and PSRoute: Actuating the Solution

PSOpt’s output is a target configuration; two actuators realize it on the cluster.

PSSched: soft stage boundaries. PSSched applies pool sizes and frequencies on the 5-minute loop with drain-before-

reassign (no KV state is ever recomputed) and 10-second DVFS refinement between epochs, making the PTA boundary soft rather than a fixed topology. Pools are *stage-tagged* instances within one deployment, not separate services. As arrival rates and length distributions drift, PSSched retags instances across stages, and PSRoute can borrow instances into a shared mixed pool at per-request timescale (Appendix C). Thus, PD is a special case, not a separate mode: when reasoning traffic drops, the think pool drains to zero and requests follow prefill-decode; when reasoning traffic rises, the think pool regrows.

PSRoute: adaptive admission instead of prediction. PSRoute enforces the contract per request: class-visibility dispatch, and admission metered against the impact bound δ_i , adapting to observed lengths rather than predicting them. Prediction-based routing to pre-sized pools cascades under heavy-tailed reasoning lengths (§III-C); instead, PSRoute tracks a rolling average of observed per-stage token lengths as the current chunk size CS_i . When outputs run shorter than expected, chunk sizes shrink and more requests are admitted; when lengths grow, concurrency is reduced to avoid KV-cache thrashing. Within pools, PSRoute uses standard multi-SLO scheduler machinery [9], [82]: per-class virtual queues with priority-aware dispatch, plus pool rebalancing between PSOpt epochs (Appendix C); we claim no novelty for these primitives, only for the power-aware layer above them.

Together the actuators close the loop across time scales – PSOpt plans per demand-response event, PSSched moves pools in minutes, and PSSched’s vote-commit DVFS daemon absorbs millisecond fluctuations; §V details the mechanisms.

V. IMPLEMENTATION

POWERSLIDER builds upon SGLang [80] to support a disaggregated prefill/think/answer architecture where GPU pools exchange KV cache memory via TCP or RDMA. We structure this section around four key deployment decisions; Table III outlines the execution frequency of each component and identifies which operations lie on the critical path for requests.

How is GPU frequency actually set. POWERSLIDER’s three-stage DVFS policy changes GPU frequency at stage boundaries (e.g., from $f_{i,\text{prefill}}$ to $f_{i,\text{think}}$ when a request enters the thinking phase). Two hardware realities shape the design. First, NVIDIA’s frequency-lock call (`nvmlDeviceSetGpuLockedClocks`) is slow relative to request timescales, taking tens to hundreds of milliseconds. Second, stage boundaries are *per-request* while DVFS is *per-GPU*, creating a coordination problem: co-resident requests from different SLO classes on one GPU may want different frequencies. We resolve both with a batched *vote-commit* protocol (Algorithm 1): requests register frequency votes at stage boundaries, and every $\Delta t_{\text{dvfs}} = 50\text{--}100\text{ ms}$ the daemon applies the SLO-weighted argmax with a single NVML call, amortizing the transition cost across all co-resident requests.

How are GPUs reassigned without recomputation. When PSSched moves a GPU between stage pools (Prefill/Think/An-

Algorithm 1 Vote-commit DVFS daemon (one per GPU g).

```

1: state: vote table  $V_g : r \mapsto (f^*, w)$ ; current frequency  $f_g$ 
2: on request  $r$  of class  $c$  entering stage  $\pi$  on  $g$ :
3:    $V_g[r] \leftarrow (f_{\pi,c}^*, w_c)$  {target from PSOpt’s solution}
4: on  $r$  completing or migrating off  $g$ : delete  $V_g[r]$ 
5: every  $\Delta t_{\text{dvfs}} = 50\text{--}100\text{ ms}$ :
6:    $f^+ \leftarrow \arg \max_{f \in F} \sum_{(f^*, w) \in V_g} w \cdot b(f; f^*)$ 
7:   { $b$  penalizes running a vote below its target  $f^*$  (SLO risk) and above
   it (wasted power)}
8: if  $f^+ \neq f_g$  then
9:   LOCKCLOCKS( $g, f^+$ );  $f_g \leftarrow f^+$  {one NVML call}
10: end if
11: reassert POWERLIMIT( $g, P_g^{\text{max}}$ ) {hardware backstop, independent of
   votes (§V)}

```

swer; SLO classes share these pools under PSRoute’s class-visibility filters), it uses a drain-before-reassign protocol: the instance stops admitting new requests, completes or migrates the KV state of in-flight ones, and only then joins its new pool. Draining bounds reassignment cost by the residual work of in-flight requests, at the price of reallocation latency – hence reallocation on the slow 5-minute loop while DVFS absorbs faster fluctuations.

What does KV transfer cost. PTA introduces one extra KV transfer per reasoning request (think $\rightarrow$ answer) than traditional PD disaggregation. We replace SGLang-Mooncake’s [60] per-transfer connections with a *persistent*, reference-counted session pool between worker pairs, multiplexed with sliding-window flow control against head-of-line blocking; scale-down drains rather than truncates in-flight transfers. An RDMA backend uses one-sided writes with the same framing, achieving 11.2 GB/s per link versus 6.8 GB/s for TCP on 100 Gbps InfiniBand. §VII-F shows the transfer overhead is largely hidden by pipelining.

The cap is enforced in hardware, not by the optimizer.

PSOpt’s solution is an operating point, not the enforcement mechanism: PSSched always programs per-GPU NVML power limits as a backstop (the final step of Algorithm 1), so a mispredicted model or stale solve affects how much goodput survives – never whether the cap holds beyond a bounded, sub-second transient (§VII-E). The stakes are contractual: under BIP, usage above the committed level during an event is penalized at \$6/kWh [55], which is why enforcement belongs in hardware rather than in the optimizer.

Frequency floor and consolidation. The solver optimizes over $[f_{\min}, f_{\max}]$ only, the range where its fitted models are valid; if a cap is too deep to hold every active GPU even at $f_{\min}$, PSOpt reduces k_i instead, consolidating load onto fewer GPUs at the next drain-bounded reallocation (until it lands, the NVML backstop holds the cap).

Preventing request starvation. LC targets are hard constraints – BE, then Flex slack, sheds first – and PSRoute promotes any Flex request older than its contract bound $\alpha_c L_c$ to LC-equivalent priority, so degradation ends in bounded queueing; if even LC becomes infeasible, PSOpt reports it and PSRoute sheds load by admission control rather than silently violating targets.

VI. EXPERIMENTAL METHODOLOGY

Cluster. We use DGX-A100 and GH200 servers as serving instances for models under different TP/EP for real-system (under 8 GPU nodes) (with SGLang + Mooncake [60] KV transfer engine), scaling to 64-512 GPUs with a modified SplitwiseSim [56] discrete-event simulator with profiled power and throughput model: three-stage disaggregation, KV-transfer flows, per-instance DVFS power control, and a runtime control plan implementation. Each GPU runs at 1410MHz nominal with a 210MHz DVFS floor. We cross-validated the simulator’s testbed-profiled models (§IV-C) component-wise against held-out measurements to be less than 4% MAPE; power-fit $R^2 > 0.95$. Our end-to-end replay of the 8-GPU workload trace through the simulator, matches measured system goodput within 4% and P90 TTLT within 9% over 5k requests.

Workload. We mix both reasoning and non-reasoning workload dynamically. For controlled study, we mix the Magpie-Reasoning and S1K reasoning traces [51], [74], [75] under a bursty gamma arrival process, and Azure LLM inference traces for coding and chat with their own scaled arrival times [49]. The default traffic mix is 30% LC, 30% Flex ($\alpha_c=3$, $\rho_c=0.3$), 40% BE at saturation throughput; variations in the mix, including drift over time, are covered in Appendix F (Figures 17). Workload and model details are summarized in Table IV.

Demand-response scenarios. We sweep both static power-cap reduction levels (0–60%) and real dynamic demand response trace for 24 hours (§VII-E). At saturation of high reasoning-mixture load, deeper caps fall below the power the cluster can shed through DVFS alone (the static-power floor of §IV-C) and engage the consolidation fail-safe of §V; the power-tracking study (Figure 11) examines that regime.

Metrics. Per-class goodput (fraction of requests meeting their SLO), P90/P99 TTFAT and TTLT, and normalized BE throughput. Latency is measured only for successfully completed requests in the stable window, excluding warmup and draining.

Baselines. **B1** Uniform — PD-disaggregated: a single GPU frequency applied cluster-wide with overlap-KV join-shortest-queue scheduling. **B2** POLCA [57] — colocated, priority-aware DVFS: low-priority requests absorb cap reductions; no probabilistic SLO tiers or per-stage control. **B3** SplitWise [56] — PD disaggregation with stage-aware, γ -proportional DVFS and overlap-KV JSQ scheduling, but no multi-SLO awareness. **B4** DynamoLLM+ [65] — colocated with uniform DVFS plus a multi-SLO autoscaler; assumes a static power budget and cannot reconfigure online. **B5** SLOs-Serve+ [9] — multi-SLO scheduling with uniform DVFS to meet power caps. Table V summarizes these axes. Concurrent fixed-envelope systems cannot accept a moving cluster cap (§VIII); POLCA is the closest GPU-deployable representative.

VII. EVALUATION

We evaluate POWERSLIDER against the five baselines across three design axes – architecture (PD vs. colocated vs. PTA), DVFS strategy (uniform vs. stage-aware vs. KKT-

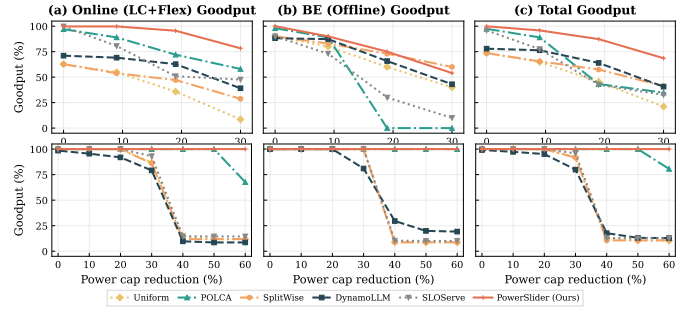


Fig. 7. Normalized online and BE goodput vs. power-cap reduction for **(top)** reasoning and **(bottom)** non-reasoning workloads. POWERSLIDER sustains $1.64\times$ the best baseline’s goodput at a 30% reduction (top) and 100% goodput for all classes through a 60% reduction (bottom). Main end-to-end result: POWERSLIDER widens the demand-response operating envelope, making the 5–20% reductions grid programs request (§II-A) essentially free.

driven), and multi-SLO resource management (single-class vs. priority-based vs. Flex) – and answer seven questions:

Q1: Does POWERSLIDER maintain goodput under static power-cap reductions? (§VII-A) **Q2:** Does it preserve tail latency for LC and Flex classes? (§VII-B) **Q3:** How sensitive is it to the Flex contract parameters (α_c , ρ_c)? (§VII-C) **Q4:** Which components matter most? (§VII-D) **Q5:** Can the solver and runtime adapt fast enough to a moving cap? (§VII-E) **Q6:** What are the overheads, and when does PTA hurt? (§VII-F) **Q7:** Does it generalize across hardware, and what does grid participation buy? (§VII-G)

Main result. Figure 7 shows: under high-load reasoning with , POWERSLIDER sustains $1.64\times$ more goodput than the best baseline, and Figure 10 shows the same system riding a real grid-emergency day at $\geq 98\%$ goodput, order of magnitude better. The remaining questions explain where the gains come from (Q2–Q4), how fast the system adapts (Q5), and what it costs (Q6–Q7).

A. Q1: End-to-End Goodput under Power-Cap Reductions

Across the sweep of different workloads, Figure 7 shows POWERSLIDER preserves goodputs at power reductions where prior systems sacrifice online traffic or starve BE jobs.

Non-reasoning (bottom). At QPS = 14 under bursty Gamma arrivals (CoV = 7.5, Azure Function Trace), POWERSLIDER holds 100% goodput for all three classes through a 60% reduction; POLCA drops to 67.7% online, and every other baseline falls below 15% beyond a 40% reduction. POWERSLIDER leverages how decode pools can drop to the $0.57\times$ frequency knee (§III-B), roughly halving their power at negligible throughput cost, and Flex slack absorbs the remainder – the cap becomes free exactly when its depth fits inside the hardware asymmetry plus the contracted slack.

Reasoning (top). The gap widens in the 128-GPU setting (QPS = 60, CoV = 13), where long thinking phases are decode-heavy enough that Uniform and SplitWise reach only 63% online goodput even uncapped. At a 30% reduction, POWERSLIDER retains 78.3% online and 54% BE goodput versus 47.6% online and zero BE for the best baseline (SLOs-Serve+), a $1.64\times$ improvement. Reasoning is intrinsically

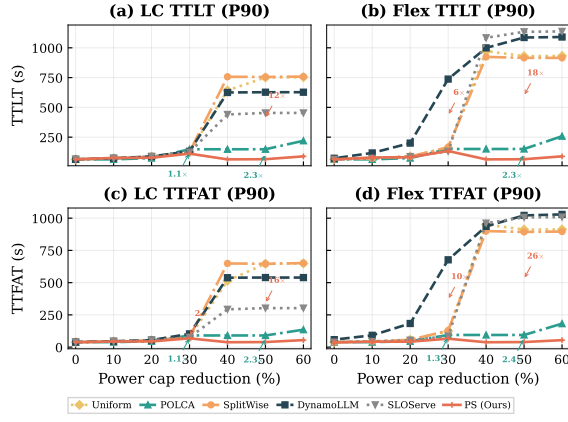


Fig. 8. P90 TTLT and TTFAT vs. power-cap reduction for the LC and Flex classes. POWERSLIDER preserves near-nominal latency up to a 60% power-cap reduction, while the best-performing baseline incurs up to $12\times$ higher LC latency and $18\times$ higher Flex latency at a 50% reduction. The Flex disparity is larger because baselines do not isolate execution stages and prioritize LC at Flex’s expense. Near-nominal tails are what keep demand-response participation compatible with latency SLOs.

harder because the cap binds alongside a second resource: thinking’s KV footprint limits decode concurrency (§III-C), and reallocating watts cannot create HBM capacity. The BE column exposes the mechanism behind the gap: baselines buy their remaining online goodput by zeroing BE, whereas POWERSLIDER funds the cap from contracted Flex slack, so shedding stays bounded and BE survives at 54%. In both regimes POWERSLIDER shifts constrained power toward the stages and classes closest to overload, while the Flex tier keeps local bottlenecks from spreading.

A time-varying-mix study (Figure 17, Appendix F) confirms the advantage from LC-dominant to Flex-dominant traffic: 95–100% online goodput as the Flex share drifts 5%→80% within one run (POLCA: 10%).

B. Q2: Tail Latency under Power-Cap Reductions

Figure 8 shows P90 TTLT and TTFAT for LC and Flex under the bursty workload. Even at a 60% power-cap reduction, POWERSLIDER keeps LC TTLT at 89s and TTFAT at 54s, only $1.1\times$ and $1.3\times$ above uncapped latency. The best-performing baseline, POLCA, reaches 220s LC TTLT and 182s Flex TTFAT, while Uniform, SplitWise, and SLOServe+ exceed 750s for LC and 900s for Flex. The separation is driven by queueing, not by slower token generation: uniform-DVFS baselines lose compute-bound prefill throughput in proportion to frequency (§III-B), so under a deep cap offered load exceeds capacity and queues, and with them the tails, grow without bound, while POLCA’s binary priority split protects its high tier only until the low tier is fully shed. POWERSLIDER keeps tails near nominal because it sheds watts where throughput is least affected and meters admission against the impact bound, so serving capacity stays ahead of admitted load and deep caps surface as bounded Flex degradation rather than queue growth, making these conclusions insensitive to the exact SLO thresholds chosen.

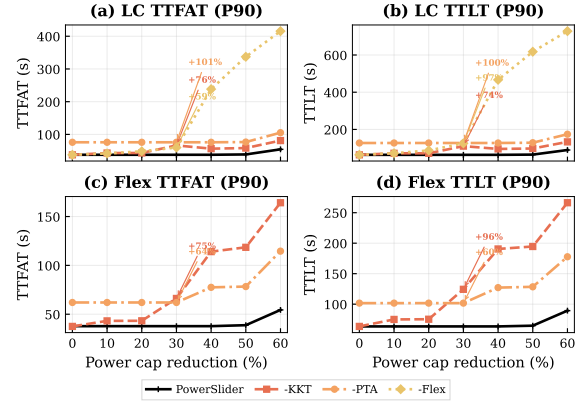


Fig. 9. Ablation of POWERSLIDER components ($-KKT$, $-PTA$, $-Flex$) for LC (top) and Flex (bottom) classes; TTFAT (left) and TTLT (right). The full design minimizes P90 latency across all regimes. Annotations indicate % latency increase over POWERSLIDER at a 30% power-cap reduction. Absorbing a cap needs all three mechanisms; no single knob suffices.

C. Q3: Sensitivity to the Flex Contract

Figure 16 (Appendix F) shows that POWERSLIDER is not sensitive to the contract parameters: varying α or ρ changes LC goodput only modestly at 20–40% power-cap reductions. Under deeper reductions, Flex differentiation becomes critical: at a 60% reduction, POWERSLIDER achieves 1.6 – $1.85\times$ the goodput of POLCA across a broad parameter range, while baselines stay near the POLCA level and Uniform drops to one-third. Returns diminish in α (raising it from 2 to 10 changes little compared with 1.1 to 2): a modest contract already captures most of the power headroom. The gain comes from the abstraction itself rather than from tuning: any reasonable (α_c, ρ_c) converts bounded user tolerance into a resource the solver can allocate (the impact bound δ_i of §IV-B), an interface the baselines lack; the $-Flex$ ablation (§VII-D) bounds the contract’s share of the end-to-end gain – and since the same parameters remain an enforced ceiling on degradation (§V), tuning moves how much headroom the operator gains, never whether the promise to users holds.

A contract-validation experiment (Figure 11) confirms $\Pr[L > \alpha_c L_c]$ stays inside the ρ_c budget at every cap depth (worst 27.5%, at a 60% reduction); the residual mass is dominated by requests shed via admission control (§V) rather than slow completions.

D. Q4: Component Ablation

POWERSLIDER’s gains come from the interaction of all three mechanisms. Figure 9 shows that KKT-based power allocation, PTA disaggregation, and Flex-aware scheduling each contribute, with roles that differ by regime: at low-to-moderate power-cap reductions, accurate power allocation and stage-aware coordination alone keep latency nearly flat; as the cap tightens, PTA isolation and Flex slack become increasingly important. The regime shift tracks which resource binds: a shallow cap is a power-allocation problem, which the solver handles alone; a deep cap becomes a memory-isolation and slack problem, where PTA and the Flex contract carry the load. **No single component dominates.** Each addresses a different

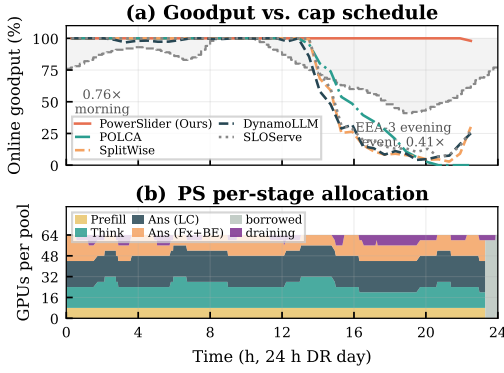


Fig. 10. **A real grid-emergency day replayed as the 24h DR day.** The cap schedule (gray) replays the Fig. 1 emergency day (Sep 6; trough $0.41\times$) [23]. (a) POWERSLIDER holds $\geq 98\%$ online goodput all day; every baseline collapses to 0–6.6% at the trough and none recovers in-trace (post-22.5h windows censored). (b) Per-stage GPU allocation under drain-before-reassign (43 reconfigurations).

bottleneck – KKT allocation distributes scarce power across groups, PTA prevents cross-stage interference, Flex converts application slack into power headroom – and removing any one at a 30% reduction raises LC TTLT by 74–100% and LC TTFAT by 59–101%.

E. Q5: Online Adaptation and Convergence

Richer DR dynamics. Figure 10 replays a real grid-emergency day rather than synthetic steps: the caps, derived from CAISO net load’s EEA3 day in Fig 1, ramp continuously, never start at full power. POWERSLIDER holds $\geq 98\%$ online goodput across the entire day (worst 60s window: 98.0%, at the deepest point of the trough). The shallow morning constraint ($0.76\times$) costs the baselines almost nothing, but the evening descent breaks all of them once the cap passes $\sim 0.8\times$: online goodput falls to 0–6.6% near the trough (POLCA 0%, SplitWise 2.8%, DynamoLLM 4.2%, SLOServe 6.6%), and because the day ends still capped at $0.77\times$, none re-enters the 95%-of-baseline band within the trace. Real cap days therefore punish the baselines harder than the stepped schedule: there is no full-power recovery window to drain their backlogs.

Per-stage allocation. Figure 10(b) shows the allocation timeline behind this behavior. Pool moves are prefill $\leftrightarrow$ think $\leftrightarrow$ answer reassignments; the drain lag is visible as the band between pools, and reallocation churn strands no capacity (no pool starved). The cadence validates the two-time-scale split of §V: one drain-bounded pool move per ~ 33 min suffices because the DVFS layer absorbs the 5-minute cap wiggles in between, so slow reallocation plus fast frequency control cover the grid’s time scales without churn. As the cap steps from 90% to 60% of nominal on a 512-GPU cluster (Figure 15, Appendix F1), POWERSLIDER keeps traffic stable with bounded BE latency, whereas most baselines face a brittle choice between dropping requests and instability.

Power-tracking accuracy. Figure 11 shows per-second delivered power against the binding cap over a replayed CAISO emergency day (trough $0.40\times$): Delivered power stays under the binding cap through every event, including the trough: DVFS bottoms out on static (idle) power, and the

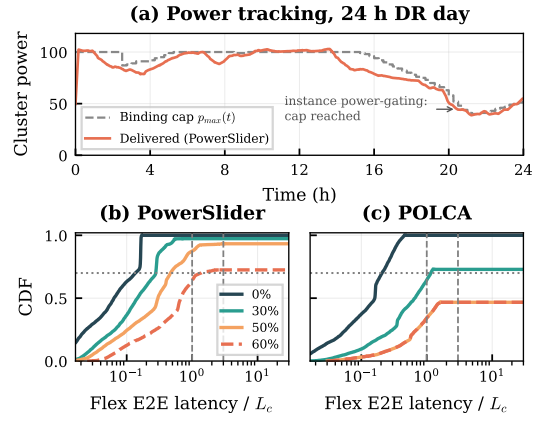


Fig. 11. **Power tracking (a) and Flex contract validation (b, c) on the compressed 24h DR day.** (a) Delivered power on the replayed day in Fig. 1: POWERSLIDER stays under the binding cap throughout; (b) Flex E2E latency CDFs (0–60% reductions; $L_c=128.9$ s) stay inside the $(\alpha_c=3, \rho_c=0.3)$ envelope at every depth; (c) POLCA breaches it beyond 30% power reduction

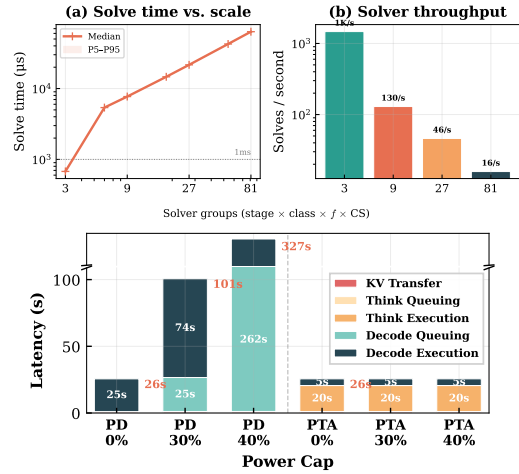


Fig. 12. **Top:** PSOpt solve time scales near-linearly with group count, staying inside one 100ms DVFS interval. **Bottom:** per-phase P50 latency, PD vs. PTA: under a 40% cap PD develops decode queueing while PTA keeps it near zero ($12.5\times$ lower end-to-end latency). Millisecond re-solves track the moving cap; stage isolation keeps it from becoming queueing collapse.

consolidation fail-safe (§V) power-gates drained instances to shed the remainder. The drain-bounded actuation transient exceeds the lead only on the steepest descent (4% of event seconds at 0.5 % of power), where the NVML backstop enforces the cap in deployment.

SLO-mix drift is equally benign: as the Flex share sweeps 5%→80% within one run dynamically, POWERSLIDER holds 95–100% online goodput where POLCA collapses to 10% (Figure 17).

F. Q6: Overheads and Failure Regimes

Solver scaling. PSOpt assigns one decision variable per group (a stage-class pair sharing a frequency and chunk size); the per-group GPU count enters the capacity model as an input rather than being enumerated, so solve cost is *independent of cluster size and request rate*: scaling from 64 to 512 GPUs changes the values fed to the solver, not the size of the optimization. Each solve runs 20 bisection

iterations at 80 decision levels per group; in the paper’s 9-group setting this takes 7.7 ms (130 solves/s), and even at 81 groups (9 hardware types $\times$ 3 classes $\times$ 3 stages) the solve takes 63 ms, inside a single 100 ms DVFS interval (Figure 12). This is also why POWERSLIDER needs no spatial control hierarchy (cf. PowerGrad’s multi-level controllers [52]): with solve cost independent of cluster size and per-GPU daemons acting locally on the solution, POWERSLIDER’s hierarchy is temporal (Table III), not spatial.

KV-transfer overhead, and when PTA hurts. Figure 12 (bottom) breaks down per-phase P50 latency, including queueing and KV transfer, under long reasoning workloads (SIK, Magpie). At nominal power, PD and PTA have similar end-to-end latency: the extra think→answer handoff is hidden by pipelining, so PTA’s overhead is negligible. Under power-cap reductions the structural benefit dominates: PD develops large decode queueing as capped decode workers thrash shared KV capacity, while PTA keeps decode queueing near zero – a $12.5\times$ lower end-to-end latency at a 40% power-cap reduction. PD, mixed, and full PTA are one continuum that POWERSLIDER traverses online by retagging instances (Appendix C); under a drifting reasoning/non-reasoning mix, the think pool tracks the traffic share within one reallocation window.

G. Q7: Generality and the Economics of Participation

Hardware generalization is discussed in Appendix F: the cubic power-model form fitted on A100 also fits public H100 power curves, and porting POWERSLIDER requires only refitting one power cubic per hardware type.

Priced at ERCOT-ERS rates (Appendix F), the baselines’ revenue loss overtakes the DR payment at 27–34% depth; POWERSLIDER stays net-positive throughout, serving the full load at better energy per token than uncapped (Figure 18(a)); no baseline reaches 90% goodput beyond a 30% cap.

H. Discussion: Implications for Future Hardware Design

Two hardware directions follow from our experience. (1) **Faster, finer-grained frequency domains.** POWERSLIDER’s control latency is dominated by actuation, not the solver: the frequency-lock call takes tens to hundreds of milliseconds and applies to the whole GPU, forcing vote-commit batching (\$V) and limiting fast grid programs. Per-partition domains with microsecond transitions (under 0.5% die area [37]) would let power follow *per-request* stage boundaries and deepen AGC-speed enrollment. (2) **Hardware power telemetry and enforcement.** Grid participation requires trustworthy caps, but today’s enforcement loop is firmware-mediated and opaque; on-package capping with bounded response time and per-SM/HBM attribution would enable contractual exceedance bounds and compute-granularity power management.

VIII. RELATED WORK

Power and latency control in datacenters. Pegasus [46], Heracles [47], Adrenaline [29], PARTIES [8], and Caladan [19] established power and frequency as runtime actuators under latency SLOs, and Dynamo [72], Thunderbolt [41],

and Flex [79] manage fleet-scale budgets under QoS — on CPUs, for inferred request-level slack; POWERSLIDER carries the premise to GPU LLM inference, where phase-dependent boundedness shapes the actuator and the slack is *contractual*.

Fixed-envelope power management for AI hardware. PowerWeave [37] (spatial per-partition DVFS under hard SLOs), Power Sloshing [10] (CPU↔GPU redistribution of a fixed server cap), and PowerGrad [52] (hierarchical steering of a cluster budget), with earlier hierarchical capping [41], [72], each divide a *fixed* envelope more efficiently but stop below the grid interface: none decides which stages, service classes, and memory configurations absorb a cap reduction. None is a *distinct* moving-cap baseline either: PowerWeave takes no budget input, server-local sloshing reduces to uniform capping under a cluster-wide cut, and PowerGrad’s gradient steering degenerates to proportional capping under stage disaggregation; and POWERSLIDER supplies the grid-facing layer such controllers could enforce.

DVFS hardware technology. Silicon already offers fine, fast control: per-core regulators [33], integrated VRs [6], digital LDOs [54], GPU DVFS prototypes [39], [63], memory DVFS [12], and sub-0.5%-area per-SM domains [37]; shipping GPUs expose one coarse frequency domain per die. POWERSLIDER is built for that interface with a granularity-agnostic solver: finer domains would deepen fast grid participation.

Multi-class QoS and interference management. Paragon [13], Quasar [14], Bubble-Up/Flux [48], [76], and their Tarcil [15], [35], [58], [73] deliver graceful per-class degradation for cache, cores, and memory bandwidth; POLCA [57] is a binary two-class instance for GPU power, and POWERSLIDER’s (α_c, ρ_c) contracts and stage-aware allocation provide the missing analogue under varying caps.

GPU and ML energy-performance tradeoffs. Zeus [77] and Perseus [11] trace energy-time Pareto frontiers for training, the view our power-goodput frontier builds on; μ -Serve, DynamoLLM, TAPAS, VoltanaLLM, and EcoServe [42], [61], [65], [66], [78] apply DVFS or auto-scaling to inference under a fixed SLO or static priority split, and throttLL’eM [31] adds predictive throttling with instance autoscaling to minimize energy under SLOs – none takes a power cap as input or expresses per-class degradation under a time-varying cap.

Datacenter demand response and carbon-aware scheduling. Grid-interactive datacenters [43]–[45], [70] and carbon-aware batch systems [20], [21], [26], [64], [71] shift *deferrable* work at job granularity; Carbon Explorer [1] shows that the ability to *follow* a grid signal unlocks carbon-aware operation. CarbonScaler [25] allocates time-varying capacity using per-job marginal-utility curves, but latency-bound serving offers no such curve; POWERSLIDER’s Flex contract and power-goodput frontier provide exactly that elasticity profile, while every request still meets its SLO as the cap moves.

IX. CONCLUSION

Dynamic power caps turn LLM inference into an architecture-aware runtime control problem: pipeline stages

differ in compute intensity, memory pressure, DVFS sensitivity, and SLO slack, so a uniform watt costs far more than a well-placed one. POWERSLIDER answers with the Flex contract, PTA disaggregation, and a KKT-guided online solver: $1.64\times$ the best baseline’s goodput at a 30% cap, LC tails within $1.3\times$ of nominal. Phase asymmetry, not uniform capping, is the foundation for grid-interactive serving.

ACKNOWLEDGMENTS

Yueying Li was supported by the NSF under grant CCF-2118709. Yueying Li acknowledges Anvil AI and GPU allocations through allocation CIS230253 from the Advanced Cyberinfrastructure Coordination Ecosystem: Services & Support (ACCESS) program, which is supported by U.S. National Science Foundation grants #2138259, #2138286, #2138307, #2137603, and #2138296. Udit Gupta and Leo Han were supported by NSF Grants CCF-232660 and CCF-2326608, and acknowledge support from Google and Amazon.

REFERENCES

- [1] B. Acun, B. Lee, F. Kazhamiaka, K. Maeng, U. Gupta, M. Chakkaravarthy, D. Brooks, and C.-J. Wu, “Carbon explorer: A holistic framework for designing carbon aware datacenters,” in *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, 2023, pp. 118–132.
- [2] A. Agrawal, N. Kedia, A. Panwar, J. Mohan, N. Kwatra, B. Gulavani, R. Ramjee, and A. Tumanov, “Taming throughput-latency tradeoff in llm inference with sarathi-serve,” in *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*. USENIX Association, 2024, pp. 1–18.
- [3] Anthropic, “Speed up responses with fast mode in claude,” <https://code.claude.com/docs/en/fast-mode>, Feb. 2026, accessed: 2026-02-13.
- [4] R. Bianchini, C. Belady, and A. Sivasubramaniam, “Datacenter power and energy management: Past, present, and future,” *IEEE Micro*, vol. 44, no. 5, pp. 30–36, 2024.
- [5] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge University Press, 2004.
- [6] E. A. Burton, G. Schrom, F. Paillet, J. Douglas, W. J. Lambert, K. Radhakrishnan, and M. J. Hill, “FIVR—fully integrated voltage regulators on 4th generation Intel Core SoCs,” in *Proceedings of the 29th Annual IEEE Applied Power Electronics Conference and Exposition (APEC)*, 2014, pp. 432–439.
- [7] J. A. Butts and G. S. Sohi, “A static power model for architects,” in *Proceedings of the 33rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2000, pp. 191–201.
- [8] S. Chen, C. Delimitrou, and J. F. Martínez, “Parties: QoS-aware resource partitioning for multiple interactive services,” in *Proceedings of the 24th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2019)*. Providence, RI: ACM, 2019, pp. 107–120.
- [9] S. Chen, Z. Jia, S. Khan, A. Krishnamurthy, and P. B. Gibbons, “SLOs-Serve: Optimized serving of multi-SLO LLMs,” *arXiv preprint arXiv:2504.08784*, 2025.
- [10] A. Cho, J. Stojkovic, L. Piga, A. Dhanotia, S. M. Sajal, G. Zuo, K. Malladi, D. Akers, K. Subramanian, S. Kanaujia, and A. Daglis, “Power sloshing in compound servers for large-scale ai inference workloads,” in *Proceedings of the 52nd Annual International Symposium on Computer Architecture (ISCA)*, 2025.
- [11] J.-W. Chung, J. You, Y. Yoo, R. Ramjee, and M. Chowdhury, “Perseus: Removing energy bloat from large model training,” in *Proceedings of the 30th ACM Symposium on Operating Systems Principles*, 2024, pp. 1–18.
- [12] H. David, C. Fallin, E. Gorbato, U. R. Hanebutte, and O. Mutlu, “Memory power management via dynamic voltage/frequency scaling,” in *Proceedings of the 8th ACM International Conference on Autonomic Computing (ICAC)*, 2011, pp. 31–40.
- [13] C. Delimitrou and C. Kozyrakis, “Paragon: QoS-aware scheduling for heterogeneous datacenters,” in *Proceedings of the 18th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2013, pp. 77–88.
- [14] C. Delimitrou and C. Kozyrakis, “Quasar: Resource-efficient and QoS-aware cluster management,” in *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2014, pp. 127–144.
- [15] C. Delimitrou, D. Sanchez, and C. Kozyrakis, “Tarcil: Reconciling scheduling speed and quality in large shared clusters,” in *Proceedings of the 6th ACM Symposium on Cloud Computing (SoCC)*, 2015, pp. 97–110.
- [16] EPRI, “Utility experiences and trends regarding data centers: 2024 survey,” Electric Power Research Institute, Tech. Rep., 2024. [Online]. Available: <https://www.epri.com/research/products/000000003002030643>
- [17] ERCOT, “Load resource participation in the ERCOT markets,” <https://www.ercot.com/services/programs/load/laar>, 2025, controllable Load Resources follow 5-minute SCED base points; accessed 2026-07.
- [18] Federal Energy Regulatory Commission, “Order no. 825: Settlement intervals and shortage pricing in markets operated by regional transmission organizations and independent system operators,” <https://www.ferc.gov/sites/default/files/2020-05/settlement825.pdf>, 2016, docket RM15-24-000; accessed 2026-07.
- [19] J. Fried, Z. Ruan, A. Ousterhout, and A. Belay, “Caladan: Mitigating interference at microsecond timescales,” in *Proceedings of the 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2020, pp. 281–297.
- [20] Í. Goiri, K. Le, M. E. Haque, R. Beachea, T. D. Nguyen, J. Guitart, J. Torres, and R. Bianchini, “Greenslot: Scheduling energy consumption in green datacenters,” in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, 2011.
- [21] Í. Goiri, K. Le, T. D. Nguyen, J. Guitart, J. Torres, and R. Bianchini, “Greenhadoop: Leveraging green energy in data-processing frameworks,” in *Proceedings of the 7th ACM European Conference on Computer Systems (EuroSys)*, 2012, pp. 57–70.
- [22] Google, “Flex inference with the Gemini API,” <https://ai.google.dev/gemini-api/docs/flex-inference>, 2025, accessed: 2026-06-11.
- [23] gridstatus developers, “gridstatus: standardized Python API to electricity grid data (CAISO, ERCOT, MISO, and others),” <https://opensource.gridstatus.io>, 2026, accessed 2026-07.
- [24] D. Guo, D. Yang, H. Zhang, J. Song, R. Zhang, R. Xu, Q. Zhu, S. Ma, P. Wang, X. Bi *et al.*, “Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning,” *arXiv preprint arXiv:2501.12948*, 2025.
- [25] W. A. Hanafy, Q. Liang, N. Bashir, D. Irwin, and P. Shenoy, “CarbonScaler: Leveraging cloud workload elasticity for optimizing carbon-efficiency,” *Proceedings of the ACM on Measurement and Analysis of Computing Systems (SIGMETRICS)*, vol. 7, no. 3, pp. 57:1–57:28, 2023.
- [26] W. A. Hanafy, Q. Liang, N. Bashir, D. Irwin, and P. Shenoy, “The war of the efficiencies: Understanding the tension between carbon and energy optimization,” in *Proceedings of the 2nd Workshop on Sustainable Computer Systems (HotCarbon)*, 2023.
- [27] S. Hong and H. Kim, “An integrated GPU power and performance model,” in *Proceedings of the 37th Annual International Symposium on Computer Architecture (ISCA)*, 2010, pp. 280–289.
- [28] M. Horowitz, “Computing’s energy problem (and what we can do about it),” in *Proceedings of the IEEE International Solid-State Circuits Conference (ISSCC)*, 2014, pp. 10–14.
- [29] C.-H. Hsu, Y. Zhang, M. A. Laurenzano, D. Meisner, T. Wenisch, J. Mars, L. Tang, and R. G. Dreslinski, “Adrenaline: Pinpointing and reining in tail queries with quick voltage boosting,” in *Proceedings of the 21st IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2015, pp. 271–282.
- [30] International Energy Agency (IEA), “Demand response,” 2023. [Online]. Available: <https://www.iea.org/energy-system/energy-efficiency-and-demand/demand-response>
- [31] A. K. Kakolyris, D. Masouros, P. Varvaroutsos, S. Xydis, and D. Soudris, “throttLL’eM: Predictive GPU throttling for energy efficient LLM inference serving,” in *Proceedings of the 31st IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2025.

- [32] W. Karush, "Minima of functions of several variables with inequalities as side constraints," Master's thesis, Department of Mathematics, University of Chicago, 1939.
- [33] W. Kim, M. S. Gupta, G.-Y. Wei, and D. Brooks, "System level analysis of fast, per-core DVFS using on-chip switching regulators," in *Proceedings of the 14th IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2008, pp. 123–134.
- [34] H. W. Kuhn and A. W. Tucker, "Nonlinear programming," in *Proceedings of the Second Berkeley Symposium on Mathematical Statistics and Probability*. University of California Press, 1951, pp. 481–492.
- [35] N. Kulkarni, G. Gonzalez-Pumariega, A. Khurana, C. A. Shoemaker, C. Delimitrou, and D. H. Albonesi, "Cutletsys: Data-driven resource management for interactive services on reconfigurable multicores," in *Proceedings of the 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2020, pp. 650–664.
- [36] W. Kwon, Z. Li, S. Zhuang, Y. Sheng, L. Zheng, C. H. Yu, J. E. Gonzalez, H. Zhang, and I. Stoica, "Efficient memory management for large language model serving with pagedattention," in *Proceedings of the 29th Symposium on Operating Systems Principles*, 2023, pp. 611–626.
- [37] V. Kypriotis, E. Dubberstein, P. H. Coppock, E. H. Solomon, R. Zamir, T. Srimani, and D. Skarlatos, "Powerweave: Unlocking energy-efficient ml on gpus with os-level spatial power management," in *Proceedings of the 53rd Annual International Symposium on Computer Architecture (ISCA)*, 2026.
- [38] Lawrence Berkeley National Laboratory, "2024 report on U.S. data center energy use," Lawrence Berkeley National Laboratory, Tech. Rep., 2024, funded by the DOE Industrial Efficiency and Decarbonization Office. [Online]. Available: <https://eta-publications.lbl.gov/sites/default/files/2024-12/lbnl-2024-united-states-data-center-energy-usage-report.pdf>
- [39] J. Lee, V. Sathisha, M. Schulte, K. Compton, and N. S. Kim, "Improving throughput of power-constrained GPUs using dynamic voltage/frequency and core scaling," in *Proceedings of the 20th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2011, pp. 111–120.
- [40] J. Leng, T. Hetherington, A. ElTantawy, S. Gilani, N. S. Kim, T. M. Aamodt, and V. J. Reddi, "GPUWatch: Enabling energy optimizations in GPGPUs," in *Proceedings of the 40th Annual International Symposium on Computer Architecture (ISCA)*, 2013, pp. 487–498.
- [41] S. Li, R. S. Acker, X. Zhang, W. Dargie, E. W. Knightly, and H. Hoffmann, "Thunderbolt: Throughput-optimized, quality-of-service-aware power capping at scale," in *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, 2020, pp. 1–18.
- [42] Y. Li, Z. Hu, E. Choukse, R. Fonseca, G. E. Suh, and U. Gupta, "Ecoserve: Designing carbon-aware ai inference systems," *arXiv preprint arXiv:2502.05043*, 2025.
- [43] M. Lin, A. Wierman, L. L. H. Andrew, and E. Thereska, "Dynamic right-sizing for power-proportional data centers," in *Proceedings of IEEE INFOCOM*, 2011, pp. 1098–1106.
- [44] Z. Liu, M. Lin, A. Wierman, S. H. Low, and L. L. H. Andrew, "Greening geographical load balancing," in *Proceedings of the ACM SIGMETRICS Joint International Conference on Measurement and Modeling of Computer Systems*, 2011, pp. 233–244.
- [45] Z. Liu, I. Liu, S. Low, and A. Wierman, "Pricing data center demand response," in *Proceedings of the ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, 2014, pp. 111–123.
- [46] D. Lo, L. Cheng, R. Govindaraju, L. A. Barroso, and C. Kozyrakis, "Towards energy proportionality for large-scale latency-critical workloads," in *Proceedings of the 41st Annual International Symposium on Computer Architecture (ISCA)*, 2014, pp. 301–312.
- [47] D. Lo, L. Cheng, R. Govindaraju, P. Ranganathan, and C. Kozyrakis, "Heracles: Improving resource efficiency at scale," in *Proceedings of the 42nd Annual International Symposium on Computer Architecture (ISCA)*, 2015, pp. 450–462.
- [48] J. Mars, L. Tang, R. Hundt, K. Skadron, and M. L. Soffa, "Bubble-up: Increasing utilization in modern warehouse scale computers via sensible co-locations," in *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2011, pp. 248–259.
- [49] Microsoft Azure, "Azure public dataset," <https://github.com/Azure/AzurePublicDataset>, 2024, microsoft Azure Public Dataset Repository. [Online]. Available: <https://github.com/Azure/AzurePublicDataset>
- [50] T. Mudge, "Power: A first-class architectural design constraint," *IEEE Computer*, vol. 34, no. 4, pp. 52–58, 2001.
- [51] N. Muennighoff, Z. Yang, W. Shi, X. L. Li, L. Fei-Fei, H. Hajishirzi, L. Zettlemoyer, P. Liang, E. Candès, and T. Hashimoto, "s1k-1.1," <https://huggingface.co/datasets/simplescaling/s1k-1.1>, 2025, hugging Face dataset.
- [52] H. Nam, R. P. Pothukuchi, A. Buyuktosunoglu, A. Amarnath, P. Bose, and J. Torrellas, "Powergrad: Hierarchical power management for power-limited ml inference clusters," in *Proceedings of the 53rd Annual International Symposium on Computer Architecture (ISCA)*, 2026.
- [53] T. Norris, N. M. Frick, E. Pattison, and D. Patino-Echeverri, "Rethinking load growth: Assessing the potential for integration of large flexible loads in us power systems," Duke University, Tech. Rep., 2025.
- [54] Y. Okuma, K. Ishida, Y. Ryu, X. Zhang, P.-H. Chen, K. Watanabe, M. Takamiya, and T. Sakurai, "0.5-V input digital LDO with 98.7% current efficiency and 2.7- μ a quiescent current in 65nm CMOS," in *Proceedings of the IEEE Custom Integrated Circuits Conference (CICC)*, 2010.
- [55] Pacific Gas and Electric Company, "Base interruptible program (BIP)," 2025. [Online]. Available: <https://www.pge.com/en/save-energy-and-money/energy-saving-programs/demand-response-programs/business-programs.html>
- [56] P. Patel, E. Choukse, C. Zhang, Í. Goiri, A. Shah, S. Maleki, and R. Bianchini, "Splitwise: Efficient generative llm inference using phase splitting," *arXiv preprint arXiv:2311.18677*, 2023.
- [57] P. Patel, E. Choukse, C. Zhang, Í. Goiri, B. Warriar, N. Mahalingam, and 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*, 2024, pp. 207–222.
- [58] V. Petrucci, M. A. Laurenzano, J. Doherty, Y. Zhang, D. Mossé, J. Mars, and L. Tang, "Octopus-man: QoS-driven task management for heterogeneous multicores in warehouse-scale computers," in *Proceedings of the 21st IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2015, pp. 246–258.
- [59] PJM Interconnection, "PJM manual 11: Energy & ancillary services market operations," <https://www.pjm.com/-/media/documents/manuals/m11.ashx>, 2025, synchronized Reserve resources, including demand response, must deliver within 10 minutes; accessed 2026-07.
- [60] R. Qin, Z. Li, W. He, J. Cui, F. Ren, M. Zhang, Y. Wu, W. Zheng, and X. Xu, "Mooncake: Trading more storage for less computation — a KVCache-centric architecture for serving LLM chatbot," in *23rd USENIX Conference on File and Storage Technologies (FAST 25)*. Santa Clara, CA: USENIX Association, Feb. 2025, pp. 155–170. [Online]. Available: <https://www.usenix.org/conference/fast25/presentation/qin>
- [61] H. Qiu, W. Mao, A. Patke, S. Cui, S. Jha, C. Wang, H. Franke, Z. Kalbarczyk, T. Başar, and R. K. Iyer, "Power-aware deep learning model serving with mu-Serve," in *2024 USENIX Annual Technical Conference (USENIX ATC 24)*. Santa Clara, CA: USENIX Association, Jul. 2024, pp. 75–93. [Online]. Available: <https://www.usenix.org/conference/atc24/presentation/qiu>
- [62] J. Rand, N. Manderlink, S. Zhang, C. Talley, W. Gorman, R. H. Wiser, J. Seel, J. Mulvaney Kemp, S. Jeong, and F. Kahril, "Queued up: 2025 edition, characteristics of power plants seeking transmission interconnection as of the end of 2024," Lawrence Berkeley National Laboratory, Tech. Rep., 2025. [Online]. Available: <https://emp.lbl.gov/queues>
- [63] A. Sethia and S. Mahlke, "Equalizer: Dynamic tuning of GPU resources for efficient execution," in *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2014, pp. 647–658.
- [64] A. Souza, N. Bashir, J. Murillo, W. Hanafy, Q. Liang, D. Irwin, and P. Shenoy, "Ecovisor: A virtual energy system for carbon-efficient applications," in *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, 2023, pp. 252–265.
- [65] J. Stojkovic, C. Zhang, Í. Goiri, J. Torrellas, and E. Choukse, "Dynamollm: Designing llm inference clusters for performance and energy efficiency," *arXiv preprint arXiv:2408.00741*, 2024.
- [66] J. Stojkovic, C. Zhao, and A. Klimovic, "Tapas: Thermal-and power-aware scheduling for llm inference in cloud platforms," in *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2025.

- [67] U.S. Department of Energy, “DOE releases new report evaluating increase in electricity demand from data centers,” <https://www.energy.gov/articles/doe-releases-new-report-evaluating-increase-electricity-demand-data-centers>, Dec. 2024, press release, accessed April 15, 2026.
- [68] U.S. Energy Information Administration, “Solar, battery storage to lead new U.S. generating capacity additions in 2025,” <https://www.eia.gov/todayinenergy/detail.php?id=64586>, 2025, today in Energy, Feb. 2025; accessed 2026-07-09.
- [69] U.S. Energy Information Administration, “Wind and solar generated a record 17% of U.S. electricity in 2025,” <https://www.eia.gov/todayinenergy/detail.php?id=67367>, 2026, today in Energy; accessed 2026-07-09.
- [70] A. Wierman, Z. Liu, I. Liu, and H. Mohsenian-Rad, “Opportunities and challenges for data center demand response,” in *Proceedings of the International Green Computing Conference (IGCC)*, 2014, pp. 1–10.
- [71] P. Wiesner, I. Behnke, D. Scheinert, K. Gontarska, and L. Thamsen, “Let’s wait awhile: How temporal workload shifting can reduce carbon emissions in the cloud,” in *Proceedings of the 22nd International Middleware Conference*, 2021, pp. 260–272.
- [72] Q. Wu, Q. Deng, L. Ganesh, C.-H. Hsu, Y. Jin, S. Kumar, B. Li, J. Meza, and Y. J. Song, “Dynamo: Facebook’s data center-wide power management system,” *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 469–480, 2016.
- [73] Y. Xu, Z. Musgrave, B. Noble, and M. Bailey, “Bobtail: Avoiding long tails in the cloud,” in *Proceedings of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2013, pp. 329–341.
- [74] Z. Xu, F. Jiang, L. Niu, Y. Deng, R. Poovendran, Y. Choi, and B. Y. Lin, “Magpie: Alignment data synthesis from scratch by prompting aligned llms with nothing,” *arXiv preprint arXiv:2406.08464*, 2024.
- [75] Z. Xu, F. Jiang, L. Niu, Y. Deng, R. Poovendran, Y. Choi, and B. Y. Lin, “Magpie-reasoning-v2-250k-cot-deepseek-r1-llama-70b,” <https://huggingface.co/datasets/Magpie-Align/Magpie-Reasoning-V2-250K-CoT-Deepseek-R1-Llama-70B>, 2025, hugging Face dataset. Instructions generated with Llama-3.1/3.3-70B-Instruct; responses generated with DeepSeek-R1-Distill-Llama-70B.
- [76] H. Yang, A. Breslow, J. Mars, and L. Tang, “Bubble-flux: Precise online QoS management for increased utilization in warehouse scale computers,” in *Proceedings of the 40th Annual International Symposium on Computer Architecture (ISCA)*, 2013, pp. 607–618.
- [77] J. You, J.-W. Chung, and M. Chowdhury, “Zeus: Understanding and optimizing GPU energy consumption of DNN training,” in *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. Boston, MA: USENIX Association, Apr. 2023, pp. 119–139. [Online]. Available: <https://www.usenix.org/conference/nsdi23/presentation/you>
- [78] J. Yu, A. Taneja, J. Lin, and M. Zhang, “Voltanallm: Feedback-driven frequency control and state-space routing for energy-efficient llm serving,” 2025. [Online]. Available: <https://arxiv.org/abs/2509.04827>
- [79] C. Zhang, S. Li, H. Wang, and H. Hoffmann, “Flex: High-availability datacenters with zero reserved power,” in *Proceedings of the 48th Annual International Symposium on Computer Architecture*, 2021, pp. 319–332.
- [80] L. Zheng, L. Yin, Z. Xie, J. Huang, C. Sun, C. H. Yu, S. Cao, C. Kozyrakis, I. Stoica, J. E. Gonzalez, C. Barrett, and Y. Sheng, “Sglang: Efficient execution of structured language model programs,” in *Advances in Neural Information Processing Systems*, vol. 37, 2024.
- [81] Y. Zhong, S. Liu, J. Chen, J. Hu, Y. Zhu, X. Liu, X. Jin, and H. Zhang, “Distserve: Disaggregating prefill and decoding for goodput-optimized large language model serving,” *arXiv preprint arXiv:2401.09670*, 2024.
- [82] K. Zhu, H. Shi, L. Xu, J. Shan, A. Krishnamurthy, B. Kasikci, and L. Xie, “PolyServe: Efficient multi-SLO serving at scale,” *arXiv preprint arXiv:2507.17769*, 2025.

APPENDIX

A. AI Usage

This section outlines the usage of AI in the generation of this submission. We used an LLM-based assistant to shorten and polish writing, to provide review-style feedbacks on drafts, and to help with small coding tasks (plotting and build scripts). All technical content, experiments, and claims were authored and verified by the authors.

B. Proofs and Supplementary Derivations

1) *KKT Derivation for Joint Optimization*: Note in this section, we expand the objective to be involving power in the first term for generality, but it’s not necessary. The Lagrangian of the joint optimization (1) with multipliers $\mu \geq 0$ (power cap) and $\eta_i \geq 0$ (impact bound per stage-class i) is:

$$\mathcal{L} = \Delta t \cdot P(\mathbf{k}, \mathbf{f}) + \sum_i w_i I_i + \mu (P(\mathbf{k}, \mathbf{f}) - p_{\max}) + \sum_i \eta_i (I_i(\text{cap}_i) - \delta_i). \quad (3)$$

a) *Frequency stationarity*.: Setting $\partial \mathcal{L} / \partial f_i = 0$:

$$\underbrace{(\Delta t + \mu) \cdot \frac{\partial P}{\partial f_i}}_{\text{power cost of raising } f_i} = - \underbrace{(w_i + \eta_i) \cdot \frac{\partial I_i}{\partial \text{cap}_i} \cdot \frac{\partial \text{cap}_i}{\partial f_i}}_{\text{impact benefit of raising } f_i}. \quad (4)$$

Under the per-GPU power model $P(f) = c_2 f^3 + c_1 f + c_0$ aggregated over k_i active GPUs, the marginal power cost on the left side is $k_i (3c_2 f_i^2 + c_1)$. The capacity gradient with respect to frequency is:

$$\frac{\partial \text{cap}_i}{\partial f_i} = \frac{k_i^{\beta_i} \gamma_i}{f_{\max}}, \quad (5)$$

which is proportional to the compute-bound fraction γ_i .

b) *Complementary slackness*.: $\eta_i^* \geq 0$ and $\eta_i^* (\delta_i - I_i^*) = 0$: the multiplier η_i is positive only when the impact bound is tight ($I_i = \delta_i$), so stage-classes with SLO headroom ($I_i < \delta_i$) have $\eta_i = 0$ and are degraded first. $\square$

2) *Deriving the Compute-Bound Fraction γ_i* :

a) *Per-layer FLOPs and bytes*.: For a standard transformer layer ($d_{\text{ff}}=4d$) processing a batch of B decode tokens with context length S , the total floating-point operations and memory bytes are:

$$F = \underbrace{24Bd^2}_{\text{MLP + projections}} + \underbrace{4BSd}_{\text{attention}} = 4Bd(6d + S), \quad (6)$$

$$M = \underbrace{12d^2 \cdot b_w}_{\text{weights}} + \underbrace{2BSd \cdot b_w}_{\text{KV cache}} = 2d(6d + BS) b_w, \quad (7)$$

where b_w is bytes per parameter (e.g., 2 for bf16).

b) *Arithmetic intensity and γ_i* .: The overall arithmetic intensity of the layer is:

$$\text{AI} = \frac{F}{M} = \frac{2B(6d + S)}{(6d + BS) b_w}. \quad (8)$$

The hardware balance point is $I^* = C/W$ (peak FLOP/s over HBM bandwidth). The layer is fully compute-bound when $\text{AI} \geq I^*$ and fully memory-bound when $\text{AI} \ll I^*$. The compute-bound fraction is:

$$\gamma_i = \min\left(1, \frac{\text{AI}}{I^*}\right) = \min\left(1, \frac{2BW(6d + S)}{(6d + BS) b_w C}\right). \quad (9)$$

c) *Implications*.: For prefill ($B = \text{prompt length}$, large), AI grows with B and $\gamma_i \rightarrow 1$. For decode with small B , $\text{AI} \approx 2B/b_w \ll I^*$, so $\gamma_i \approx 0$. Longer context S increases the KV-cache memory term in the denominator, further reducing γ_i , confirming that decode stages with long reasoning traces are predominantly memory-bound and amenable to frequency reduction with minimal throughput loss.

3) *Throughput, Power, Latency, and Energy Models:* PSOpt’s analytical solver relies on the four per-stage models profiled in §IV-C and visualized in Figure 4. All four are fit per-model on A100-80GB at TP-4 across the batch-size breakpoints reported in §IV-C.

a) *Latency.*: Per-batch latency is fit as a piecewise-linear function of f_i with a knee at a batch-size-dependent saturation frequency $f^*(BS)$:

$$L(f_i; BS) = \begin{cases} a_1(BS) - b_1(BS) f_i, & f_i \leq f^*(BS) \\ a_2(BS) - b_2(BS) f_i, & f_i > f^*(BS) \end{cases} \quad (10)$$

with $b_2 \ll b_1$ in the memory-bandwidth-bound regime (low γ_i , e.g. answer decode at small batch), collapsing to a single linear segment when the stage is compute-bound across the full frequency range (high γ_i , e.g. prefill). This regime-dependent slope is what makes stage-aware DVFS effective: dropping f_i on low- γ_i stages beyond f^* cuts power with negligible latency cost.

b) *Throughput.*: Per-instance throughput follows directly from Eq. 10 as $BS/L(f_i; BS)$. Figure 4 confirms the regime split implied by the two slopes: at small batch size throughput saturates above ~ 810 MHz, while at large batch size it scales nearly linearly up to 1,410 MHz.

c) *Power.*: Per-GPU power follows the cubic model $P(f) = c_2 f^3 + c_1 f + c_0$ from §IV-C, with coefficients (c_2, c_1, c_0) profiled at each batch-size breakpoint; aggregate stage power is $k_i P(f_i)$. The R^2 of the fit exceeds 0.95 across all profiled points.

d) *Energy.*: Per-token energy is $P(f_i) \cdot L(f_i; BS)/BS$. In the compute-bound regime latency drops roughly as $1/f_i$ while power grows as f_i^3 , so energy-per-token has a minimum at an intermediate frequency; in the memory-bound regime latency is nearly flat beyond f^* so energy-per-token falls monotonically as f_i is reduced, justifying the aggressive frequency reduction PSOpt selects for answer decode.

C. From Static Provisioning to Dynamic Runtime Routing

The runtime operates in two layers. An *allocator* provisions Prefill, Think, and Answer pools once at start-up, fixing the per-stage parallelism configuration and tagging each instance with its home stage. A *dispatcher* then routes every arriving request across those pools and, under skewed load, re-tags a small number of instances so that the effective pool boundary tracks the offered workload between PSOpt epochs.

a) *Static provisioning.*: Given per-stage pool sizes and parallelism degrees from PSOpt, the allocator partitions the cluster into three contiguous blocks, one per stage, and spawns instances at tensor-parallel granularity within each block. The result is three sets of stage-tagged instances that define the initial k in Eq. 1; no further static capacity planning occurs until the next PSOpt epoch.

b) *Dynamic dispatch.*: At arrival, each request is classified as two-phase (Prefill then Answer) or three-phase (Prefill, Think, then Answer) based on whether the model emits a reasoning trace. For each hop, the dispatcher selects the least-loaded instance in the corresponding pool under a load key ϕ .

Three variants of ϕ are supported – memory reservation (JSQ), pending-token count (TokenJSQ), and a convex combination of the two (Weighted) – expressing the same selection rule under different notions of “load.” Reservations are updated before the dispatcher returns so that subsequent arrivals observe the new load immediately; KV-cache transfers between selected instances contend on the destination’s inbound bandwidth.

c) *Online pool rebalancing.*: Strict stage pools are efficient when the offered mix matches the provisioned split but waste capacity when one stage saturates while another is idle, a common situation for reasoning workloads, whose think-to-answer ratio varies with prompt difficulty. POWERSLIDER absorbs this mismatch without waiting for the next PSOpt epoch: when a pool has no instance satisfying its load and SLO predicates, the dispatcher *borrow*s an underutilized instance from a sibling pool, re-tags it for the borrowing stage, and places it in a shared “mixed” pool visible to both stages. When a borrowed instance drains its in-flight work, it is released back to its home pool. This borrow-and-return mechanism turns the static P/T/A boundary into a soft one at the per-request timescale, while PSOpt remains responsible for coarse re-provisioning at epoch boundaries. Class-visibility filters (a request of class c considers only instances whose reserved load comes from classes at priority $\geq c$) interact with rebalancing without further modification: LC requests see the whole cluster, Flex and BE requests see progressively smaller slices, preserving the SLO-tier ordering from §IV-B.

Algorithm 2 distills the two-layer runtime: a one-shot PROVISION step at epoch boundaries and a per-request DISPATCH step that invokes REBALANCE on demand.

Algorithm 2 POWERSLIDER static provisioning and dynamic routing.

```

1: function PROVISION( $k$  from PSOpt, per-stage parallelism)
2:   partition cluster into blocks of size  $k_\pi$ , one per stage  $\pi \in \{P, T, A\}$ 
3:   spawn tensor-parallel instances in each block; tag each by home stage
4: end function
5: function DISPATCH( $R$  of class  $c$ , load key  $\phi$ , SLO  $\tau_c$ )
6:    $\mathcal{H} \leftarrow$  stage sequence for  $R$            {2-phase or 3-phase}
7:   for each stage  $\pi \in \mathcal{H}$  do
8:      $i_\pi^* \leftarrow \arg \min_{i \in \text{pool}(\pi), \text{class-visible to } c} \phi(i)$ 
9:     if no  $i_\pi^*$  admits  $R$  within  $\tau_c$  then
10:       $i_\pi^* \leftarrow \text{REBALANCE}(\pi)$            {borrow from sibling pool}
11:     end if
12:   end for
13:   reserve load on each  $i_\pi^*$  and dispatch  $R$  along  $\mathcal{H}$ 
14:   return  $(i_\pi^*)_{\pi \in \mathcal{H}}$ 
15: end function
16: function REBALANCE( $\pi$ )
17:   pick underutilized instance  $j$  from a sibling pool; re-tag  $j$  to  $\pi$ 
18:   place  $j$  in a mixed pool visible to both stages
19:   on completion of  $j$ ’s in-flight work, release  $j$  to its home stage
20: return  $j$ 
21: end function

```

TABLE IV
WORKLOADS IN EVALUATION AND LATENCY REQUIREMENTS.

Model	TTF(A)/T	TTLT/TBT	Dataset
CodeLlama-34B	72.0s	115.5s	Magpie
QWen-32B	36.0s	121.0s	S1K
DS-R1-Llama-70B	5.0s	0.50s	Azure-Code
QWen-14B	0.30s	0.1s	Azure-Chat

TABLE V
BASELINE CONFIGURATIONS.

Name	Arch	DVFS Policy	Scheduler	Multi-SLO
B1 Uniform	PD disagg	Uniform	Overlap-KV JSQ	Single class
B2 POLCA	Collocated	Priority-aware	Mixed-JSQ	HP/LP binary
B3 SplitWise+	PD disagg	Stage-aware	Overlap-KV JSQ	+Priority
B4 DynamoLLM+	Collocated	+Uniform	Prediction rtr.	SLO-aware
B5 SLOs-Serve+	Collocated	+Uniform	ProMax	LC/BE priority
POWERSLIDER	PTA disagg	KKT solver	Multi-SLO JSQ	LC/Flex/BE

D. Experimental Setup Details

Table IV lists the workloads and their latency targets; Table V summarizes the five baselines, all using the same cluster and power-aware routing.

E. Additional Motivation Data

1) *Prediction-Based Routing under Reasoning Workloads:* Figure 13 details the misrouting cascade summarized in §III-C. A DistBert-style length classifier trained on the reasoning distribution of Figure 5 yields a 37.1% overall misprediction rate with a severe asymmetry: 26.6% of Medium and 16.3% of Long requests are misrouted to the Short pool, inflating its load by 25%; Medium requests are hardest to classify, with only 49.9% routed correctly even after recalibrating length-class thresholds to equalize class sizes. At low request rates the impact is negligible, but as the system approaches saturation (the regime a power-cap reduction induces), TTFT rises sharply: mean latency doubles and P99 increases by over an order of magnitude, while per-token metrics stay below 10%, confirming that the bottleneck is routing-induced queuing rather than per-token compute. The behavior is not specific to DynamoLLM; it applies to any pre-sized-pool design whenever decode-length variance across request classes is high.

2) *PTA Benefit across Workload Distributions:* Figure 14 sweeps the think-to-answer ratio and prefill length behind the Observation 3 numbers (§III-C), showing the PD-vs-PTA gap across the workload-distribution space.

F. Additional Experiments

1) *Per-Request Transient Behavior:* Figure 15 shows the per-request scatter behind the Q5 summary (§VII-E): response times for every completed request in all three classes, for POWERSLIDER and the three strongest baselines, across the cap step.

This appendix collects additional results referenced from §VII, workload-mix sensitivity (Figure 17), economic break-even and the energy-goodput Pareto (Figure 18), and grid-program feasibility (Table VI).

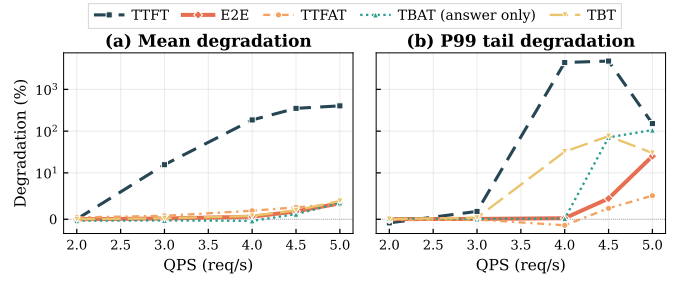


Fig. 13. Latency degradation from prediction-based misrouting under increasing per-instance offered load (128-instance cluster). At low load (≤ 2 rps per instance) misrouting has negligible impact; near saturation (~ 5 rps per instance), TTFT mean degrades $>100\%$ and P99 exceeds 1,000%, while end-to-end and TTFAT remain buffered until saturation.

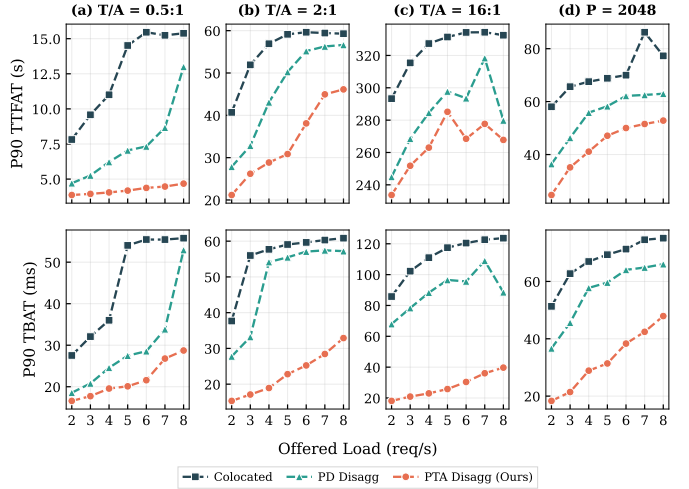


Fig. 14. Robustness of the PTA benefit across workload distributions. Panels (a)–(c) increase the T/A ratio from 0.5:1 to 16:1 (average prefill 512); panel (d) uses average prefill 2048. The advantage grows with T/A ratio (49% TBAT reduction at 0.5:1 up to 71% at 16:1 at high load) because longer thinking chains cause more severe batch interference under PD and colocated serving, which PTA eliminates by isolating answer decode onto dedicated workers.

Workload-mix sensitivity. Figure 17 quantifies how much of the Q1 advantage depends on the traffic mix, using a mix that drifts within one run rather than separate static runs. POWERSLIDER is essentially mix-insensitive: it sustains 95–100% online goodput over the whole 5%→80% Flex drift at a 30% power-cap reduction (92–100% at 50%), shedding BE admissions rather than violating online SLOs when the cap makes full service infeasible, and 93–100% at every mix extreme. Baselines degrade as the Flex share grows because they lack an interface to exploit contractual slack: at a 30% reduction, POLCA falls from 100% at $\leq 40\%$ Flex to 58% at 60% and 10% at 80% Flex; at the Flex-dominant extreme (10/80/10) it retains 14% (30% reduction) and 0% (50%) vs. POWERSLIDER’s 93–99%.

a) *Generalization beyond A100.:* The cubic DVFS power model fitted on GH200 ($R^2=0.98$) also fits public H100 power: the same cubic form holds with refitted coefficients. Porting POWERSLIDER to a new hardware type therefore requires only that per-hardware refit of the power cubic and

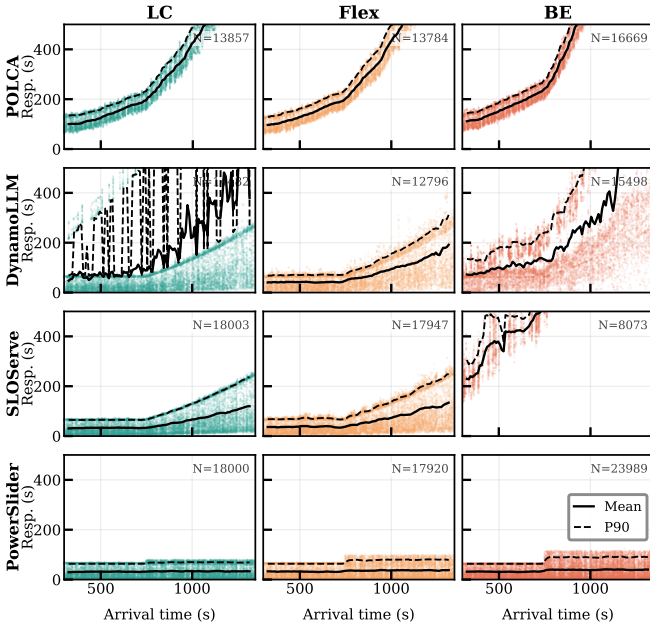


Fig. 15. Per-request response times for LC, Flex, and BE classes as the power cap steps from 90% to 60% of nominal at $t = 750$ s on a 512-GPU cluster. POWERSLIDER completes the most requests across all classes with stable response times; baselines either shed BE traffic or accumulate unbounded queuing.

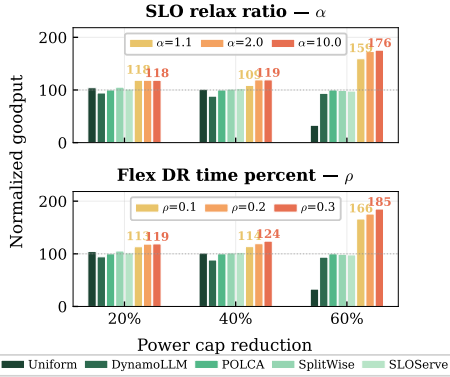


Fig. 16. Sensitivity of POWERSLIDER's Flex SLO parameters on LC goodput, normalized to POLCA. **Top:** varying the relaxation ratio α with $\rho = 0.2$. **Bottom:** varying the violation-budget fraction ρ with $\alpha = 3.0$. At a 60% power-cap reduction, POWERSLIDER delivers up to $1.85\times$ the goodput of POLCA and remains robust over a wide range of α and ρ . The contract that funds POWERSLIDER's power headroom does not need careful tuning to deliver it.

throughput points; the simulator already carries an H100 profile scaled from the measured A100 coefficients, with the wider 210–1980 MHz frequency range and 700 W TDP. We expect the wider DVFS range to increase goodput retention under deep caps; a full H100 evaluation is left to future work.

Energy per useful token. Figure 18(a) recasts the goodput results in energy terms. At every cap depth POWERSLIDER's energy per generated token stays within 0.34–0.44 J: at a 60% cap reduction it still serves 97.9% of online load at 0.42 J/token, equal-or-better than its own uncapped 0.44 J, because stage-aware DVFS slows memory-bound pools where throughput is nearly frequency-insensitive. The baselines' en-

TABLE VI
GRID-PROGRAM FEASIBILITY GIVEN POWERSLIDER'S ACTUATION PATHS (7.7 MS SOLVE; <1 S DVFS BAND; 5 MIN REALLOCATION EPOCH).

Program	Resp. req.	POWERSLIDER path	Feasible
Freq. regulation (AGC)	seconds	DVFS band (<1 s)	✓ ($\leq 20\%$)
Spinning reserve	10 min	reallocation epoch	✓ (full depth)
Emergency DR	30 min	reallocation epoch	✓ (full depth)
Day-ahead / economic	hours	capacity planning	✓
Stacked (reg. + reserve)	both	both bands	✓

ergy per token degrades $\approx 3\times$ at deep caps not because they draw more power but because most of that energy goes to requests that miss their SLOs or never complete; POLCA avoids the worst of this by shedding load, plateauing at 0.63 J/token with 35% goodput. Joules per useful token is also the quantity operational-carbon models consume.

Economic break-even. Figure 18(b) converts measured goodput into dollars: DR enrollment pays for curtailed energy, while cap-induced goodput loss forfeits serving revenue. We assume scarcity-event compensation of \$1.75/kWh of committed curtailment (emergency-response-service pricing; economic-DR prices outside scarcity events are orders of magnitude lower) and \$2.50/GPU-hr serving revenue. Because every system receives the same payment at a given depth, the ranking below is price-independent; the absolute break-even depths scale with the curtailment-to-revenue price ratio. Under these prices, every baseline's revenue loss overtakes the DR payment at moderate depths – POLCA first at $\approx 27\%$, Uniform at $\approx 33\%$, the remaining baselines at 32–34% – consistent with operators' current reluctance to enroll inference clusters in DR programs. Because POWERSLIDER holds online goodput at 97.9–100% at every depth, its revenue loss never exceeds \$4 per event and net benefit grows monotonically with depth, reaching \$103 per 2 h event at a 60% reduction under scarcity-priced events.

b) Multi-tier grid-program participation. Table VI assesses which formal grid programs POWERSLIDER's actuation latencies can serve. Two response paths exist: the DVFS-only band (up to a ~ 15 –20% power-cap reduction) actuates in under a second (7.7 ms solve + 100 ms DVFS interval), fast enough for AGC-driven frequency regulation; deeper responses require one 5 min drain-bounded reallocation epoch, comfortably inside spinning-reserve (10 min) and emergency-DR (30 min) windows. Because the two bands actuate independently, they can be *stacked*: enroll the DVFS band in regulation while committing the reallocation band to reserves, monetizing both fast and deep flexibility from the same cluster.

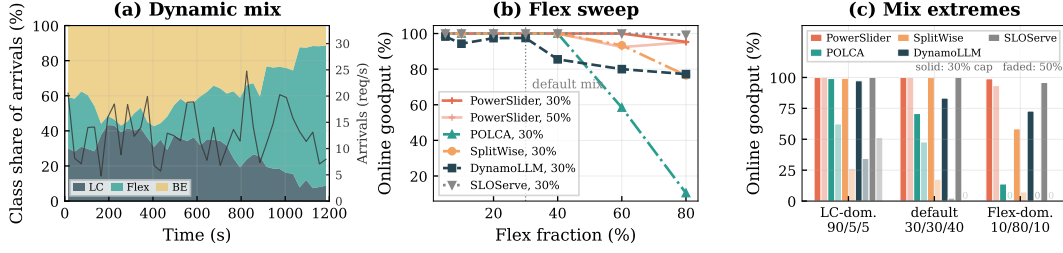


Fig. 17. **Workload-mix sensitivity under a dynamic user mix (measured).** (a) The LC/Flex/BE arrival mix drifts from 5% to 80% Flex *within a single run* (QPS 14, CoV 7.5 bursty arrivals, 64 GPUs); the gray line is the arrival rate. (b) Online goodput per mix segment at a 30% power-cap reduction (POWERSLIDER also shown at 50%): POWERSLIDER holds 95–100% goodput across the entire drift (95.3% at 80% Flex, shedding BE admissions when the cap makes full service infeasible), while POLCA collapses to 10% at 80% Flex. (c) Mix extremes – LC-dominant (90/5/5), default (30/30/40), Flex-dominant (10/80/10) – at 30% (solid) and 50% (faded) reductions.

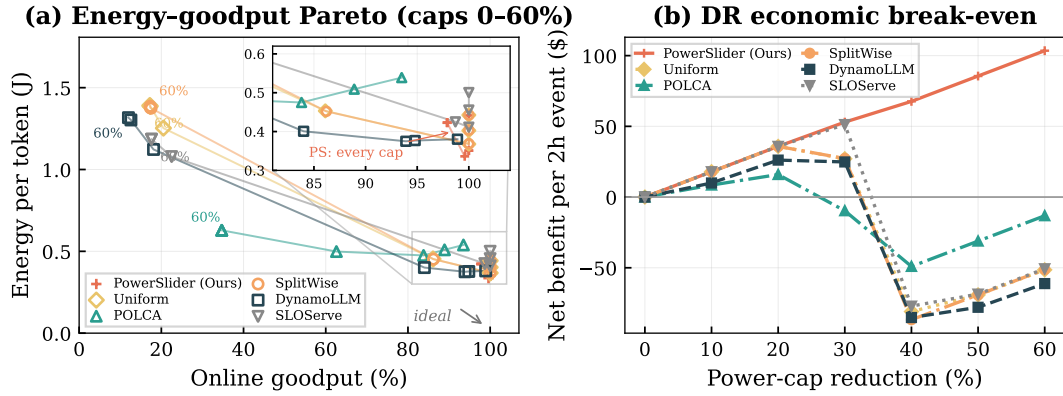


Fig. 18. **Energy-goodput Pareto and economic break-even under power caps (measured).** (a) Cluster energy per generated token vs. online goodput, one point per system and cap depth (0–60%; ideal is bottom-right). POWERSLIDER alone holds the ideal corner at every depth (0.34–0.44 J/token at 97.9–100% goodput; inset), while at a 60% cap the baselines pay $\approx 3\times$ the energy per *useful* token at 12–23% goodput. (b) Net benefit per 2h DR event vs. cap depth; POWERSLIDER is net-positive at every depth where the baselines turn negative (§VII-G).