Solve the Sudoku: Few-Shot Energy-Latency Map Profiling for Sustainable LLM Serving

KUNMING ZHANG, The Hong Kong University of Science and Technology (Guangzhou), China
XIANYI YUAN, The Hong Kong University of Science and Technology (Guangzhou), China
DEKE GUO, Sun Yat-sen University, China
GUOMING TANG*, The Hong Kong University of Science and Technology (Guangzhou), China

Energy- and carbon-aware LLM serving requires deployment-specific energy-latency maps, but exhaustively profiling every engine-GPU-model configuration is costly. We present KernelScale, a few-shot map-recovery framework that decomposes LLM inference into kernel families and learns engine-local workload scaling laws shared across GPU-model stacks. KernelScale pools sparse measurements to estimate reusable slopes and calibrates each deployment with a few measured configurations. Across 3 GPUs, 3 models, 2 serving engines, and 4,135 measured configurations, KernelScale recovers maps from only 3 shots per stack with 9.6% WAPE, reducing profiling cost by up to 15× compared to baselines. We also demonstrate that the shared slopes can be transferred, allowing a single target-side measurement to reduce prediction error on unseen GPUs or models from 45.5% and 80.2% to 16.5% and 15.8%, respectively.

CCS Concepts: • **Hardware → Power estimation and optimization**; • **Computing methodologies → Modeling and simulation**; • **Computer systems organization** → *Cloud computing*.

Additional Key Words and Phrases: LLM serving, GPU profiling, energy modeling, latency prediction, few-shot profiling, carbon-aware computing, kernel families

1 Introduction

Global data center energy consumption reached 415 TWh in 2024, driven predominantly by AI workloads, with LLM inference emerging as the fastest-growing contributor [9]. To mitigate this surge, sustainable LLM serving could play a pivotal role by balancing energy-latency trade-offs and enabling energy- or carbon-aware scheduling under strict performance constraints. Recent work has shown that such schedulers can substantially reduce operational energy and carbon emissions through dynamic request-time configuration optimization [14, 29].

All such schedulers implicitly require a detailed energy-latency landscape that specifies how energy and performance vary across the feasible configuration space [16, 22]. We formalize this per-configuration knowledge as an *energy-latency map*: for a given engine-GPU-model stack, the map records the energy cost and latency of every feasible inference configuration across workload axes (e.g., batch size, input length, output length). Before a scheduler can reduce energy or carbon, the operator must first pay the cost of building the map that the scheduler optimizes over. The map itself is not an operational carbon policy. Rather, it is the prerequisite and profiling substrate that enables downstream energy- or carbon-aware policies that can combine per-configuration energy information with queuing dynamics, fleet constraints, and time-varying grid carbon intensity.

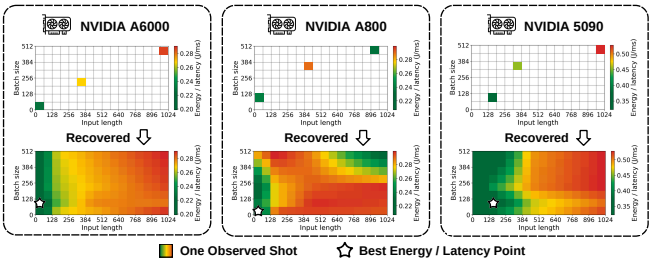


Fig. 1. Few-shot map recovery on three GPUs (with Mistral-7B, vLLM), like solving the *Sudoku* puzzle. Top: a sparse set of observed configurations. Bottom: the full energy-latency map recovered by KernelScale.

The difficulty is that the map is expensive to obtain and refresh. Exhaustively profiling a single engine-GPU-model stack over a representative workload grid requires hundreds of runs, each consuming significant GPU time and energy [15]. LLM inference energy is shaped by interacting deployment factors, including GPU architecture and runtime power behavior, model scale, request shape, batching, and serving-engine implementation [7, 8, 17, 22]. A black-box energy surface learned for one engine-GPU-model stack therefore rarely transfers directly to another. Existing energy-aware serving and modeling approaches commonly rely on dense profiling, stack-specific models, or calibrated primitive databases before they can guide deployment decisions [1, 3, 6, 25, 27]. A practical map-recovery method must therefore account for both prediction error and profiling cost.

This paper aims to address this cost barrier by studying few-shot energy-latency map recovery for sustainable LLM serving. We find that few-shot recovery is feasible because the map possesses exploitable structure at the kernel-family level. For example, as input length increases by 64×, attention's share of decode latency rises from 3% to 15%, while GEMM's share declines correspondingly. A conventional end-to-end predictor cannot distinguish the shift, whereas decomposing inference into kernel families isolates such composition shifts and captures the underlying feature. Furthermore, individual kernel families exhibit regular scaling behavior governed by their computation patterns, which can be captured by compact scaling laws whose workload slopes we observe to be shared across GPU-model stacks within a serving engine (due to the fundamental similarity of memory and compute patterns). Shared slopes can therefore be learned by pooling the sparse measurements, making few-shot map recovery feasible.

*Corresponding author.

Authors' Contact Information: Kunming Zhang, kzhang519@connect.hkust-gz.edu.cn, The Hong Kong University of Science and Technology (Guangzhou), China; Xianyi Yuan, yuanxianyi1018@outlook.com, The Hong Kong University of Science and Technology (Guangzhou), China; Deke Guo, guodk@mail.sysu.edu.cn, Sun Yat-sen University, China; Guoming Tang, guomingtang@hkust-gz.edu.cn, The Hong Kong University of Science and Technology (Guangzhou), China.

We present KernelScale in this work, a kernel-family scaling-law framework that recovers the full map with only a few measurements (by exploiting structural regularities) in the map, much like solving a *Sudoku* puzzle, as illustrated in Fig. 1. For a fixed serving engine, KernelScale collects a minimum (of three) shots per GPU-model stack, pools these to fit shared workload slopes across kernel families, and then calibrates each stack's deployment-specific intercept using its own shots. For a new GPU or model under the same engine, KernelScale can reuse the learned slopes. In the minimum setting, one target shot suffices to anchor the intercept of an unseen deployment.

Contributions. This paper makes three contributions.

- We decouple energy-latency map profiling from downstream scheduling and identify two structural properties that enable few-shot recovery. First, as input length grows, the latency and energy composition across kernel families shifts substantially, making it necessary to model the map at the kernel-family level rather than as a monolithic end-to-end function. Second, each kernel family exhibits computation patterns that are shared across GPUs and models within a given serving engine.
- We design KernelScale, a kernel-family scaling-law framework that decomposes inference into per-family scaling models to isolate composition shifts, applies scaling-law structure to capture the underlying computation patterns and pools few-shot measurements across stacks to learn shared workload slopes.
- Across 3 GPU types, 3 models, and 2 serving engines, KernelScale achieves 9.6% mean WAPE from only 3 shots per stack, whereas a strong black-box predictor requires 15× more profiling cost to reach comparable accuracy. For unseen GPUs or models, a single target-side measurement reduces prediction error from 45% and 80% to about 16%. The recovered maps also preserve latency-constrained energy selection, reducing the feasible energy gap from 33.2% to 8.7%.

2 Methodology

KernelScale recovers deployment-specific energy-latency maps by exploiting two structural properties of LLM inference. First, as input length changes, the latency and energy composition across kernel families shifts substantially. Second, individual kernel families exhibit regular scaling behavior whose workload slopes are shared across GPU-model stacks within a serving engine. These properties enable a three-step approach: decompose inference into kernel families to isolate composition shifts (§2.1), fit compact scaling laws with shared slopes to capture computation patterns (§2.2), and calibrate each deployment stack's intercept (§2.3).

KernelScale recovers a map over the following configuration tuple $c = (e, g, m, \mathrm{BS}, \mathrm{IL}, \mathrm{OL})$, where e is the serving engine, g is the GPU, m is the model, BS is batch size, IL is input length, and OL is output length. The prediction targets are stage-level latency and energy for both prefill and decode, where latency is the summed CUDA kernel execution time and energy is integrated from a reconstructed 1 ms power trace calibrated against NVML [20] samples.

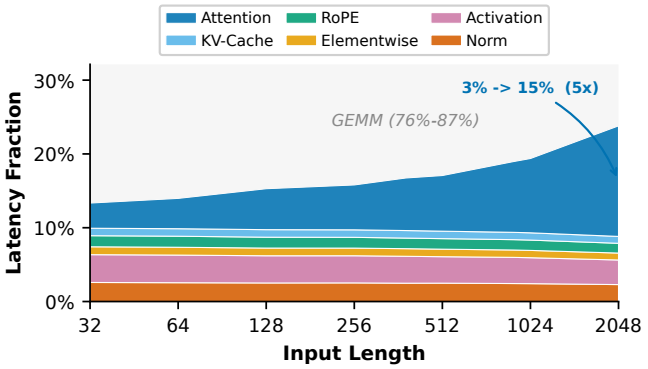


Fig. 2. Kernel-family composition shift during decode (vLLM, Mistral-7B, A800): attention's share grows with input length while GEMM falls.

2.1 Kernel-Family Decomposition

A direct end-to-end predictor learns $\hat{Y} = f(\mathrm{BS}, \mathrm{IL}, \mathrm{OL})$ for a whole inference stage. This couples two distinct signals: how each kernel family scales with workload parameters and how the mixture of family contributions changes across operating points. Fig. 2 illustrates this effect during decode. As input length grows from 32 to 2048 tokens, attention's latency share increases by about 5× while GEMM changes much more slowly, shifting attention's latency share from 3% to 15%. A direct model must explain both the within-family scaling and the cross-family composition shift with one function.

Kernel-family decomposition separates these signals by construction. For a target $q \in \{\text{latency}, \text{energy}\}$, KernelScale predicts a stage-level quantity as the sum of family components:

$$\hat{Y}_s^{(q)}(c) = \sum_{f \in \mathcal{F}} \hat{y}_{f,s}^{(q)}(c),$$

where s denotes the deployment stack and f denotes a kernel family. Each component $\hat{y}_{f,s}^{(q)}$ is directly supervised by the corresponding family-level measurement. This structure gives composition-aware generalization: when a workload shifts more runtime or energy into attention or KV-cache movement, the affected family component changes without forcing the model to relearn unrelated families.

KernelScale groups kernels into seven canonical families: attention, GEMM, KV-cache management, normalization, activation, elementwise operations, and rotary embedding. Families are defined by deterministic name-pattern rules applied uniformly across GPUs, models, and serving engines. This decomposition yields four prediction tasks per engine, prefill/decode × latency/energy, all sharing the same family-level structure.

2.2 Scaling-Law Model

With inference decomposed into kernel families, each family can be modeled independently because its cost is governed by a specific computation pattern that determines how it scales with workload parameters. Compute-bound families scale with arithmetic work. During prefill, attention performs full self-attention over the prompt, so its FLOPs grow quadratically with input length [5]. GEMM FLOPs grow linearly with the token count times hidden dimension [2]. During decode, each generated token triggers a round of token-local

Table 1. Family-level workload features. The model uses compact log-space features whose terms match the dominant execution pattern of each family.

Stage	Family group	Workload features
Decode	Attention	$\log W_{\text{attn}}$, $\log \text{BS}$, $(\log \text{BS})^2$, $\log \text{BS} \cdot \log(\text{IL} + \text{OL})$
Decode	KV-cache	$\log \text{OL}$, $\log \text{BS}$, $(\log \text{BS})^2$, $\log \text{IL}$
Decode	Other families	$\log \text{OL}$, $\log \text{BS}$, $(\log \text{BS})^2$
Prefill	Attention	$\log \text{IL}$, $\log \text{BS}$, $\log \text{IL} \cdot \log \text{BS}$, $(\log \text{IL})^2$, $(\log \text{IL})^3$
Prefill	GEMM	$\log \text{IL}$, $\log \text{BS}$, $\log \text{IL} \cdot \log \text{BS}$, $\log \text{BS} \cdot \log(\text{IL})/\text{IL}$
Prefill	Other families	$\log \text{IL}$, $\log \text{BS}$, $\log \text{IL} \cdot \log \text{BS}$

Kernel families: attention, gemm, norm, rope, activation, elementwise, kv_cache

work across all families. Attention additionally scans the growing KV history, so it has cumulative work growth. Memory-bound families, such as activation, are dominated by data movement and scale nearly linearly with the token-loop repetition count [18], but exhibit batch-size curvature because batch size influences not only total work but also parallel efficiency, memory coalescing, and overhead amortization [26].

These computation patterns produce multiplicative scaling relationships: doubling the workload parameter multiplies cost by a hardware- and family-specific factor [10]. Taking logarithms converts these relationships into an additive model, enabling compact log-linear scaling laws. Thus, each family model takes the compact log-linear form

$$\log \hat{y}^{(q)}_{f,e,g,m} = \underbrace{\alpha^{(q)}_{f,e,0} + \alpha^{(q)}_{f,e,g} + \alpha^{(q)}_{f,e,m}}_{\text{engine-local deployment intercept}} + \boldsymbol{\theta}^{(q)\top}_{f,e} \boldsymbol{\varphi}_f(\text{BS}, \text{IL}, \text{OL}). \quad (1)$$

For a fixed serving engine e, the slope vector $\boldsymbol{\theta}_{f,e}$ captures the workload scaling of family f and is shared across GPU-model stacks. All terms in Eq. 1 are learned from measured profiling shots using per-family ordinary least squares in log space. Source-stack shots are pooled to fit the family workload slopes and the observed GPU/model fixed-effect intercepts: $\alpha^{(q)}_{f,e,0}$ is the engine-family base intercept, $\alpha^{(q)}_{f,e,g}$ is the GPU fixed effect, and $\alpha^{(q)}_{f,e,m}$ is the model fixed effect. These intercepts capture deployment scale rather than acting as opaque GPU-model-pair identifiers. For an unseen GPU or model, KernelScale freezes the learned workload slopes and estimates the missing low-dimensional deployment scale from the target-side shots, rather than relearning the full workload-scaling law. No intercept term is assumed from hardware specifications.

Family-specific workload features. KernelScale uses different feature skeletons $\boldsymbol{\varphi}_f$ for decode and prefill because the two stages execute different computation patterns. Decode is a token loop while prefill processes the full prompt in one forward pass. Table 1 summarizes the resulting feature sets. The same feature skeleton is used for latency and energy, but the coefficients are learned separately for each target. The feature-skeleton terms are chosen from the dominant loop structure of each family rather than from held-out test error.

For decode attention,

$$W_{\text{attn}} = \sum_{t=1}^{OL} (IL + t) = OL \cdot IL + \frac{OL(OL+1)}{2}, \quad (2)$$

which approximates the cumulative context scanned over a decode window. Batch size is kept outside W_{attn} because it changes not only the amount of work, but also occupancy and memory behavior. KernelScale therefore models batch size with separate first-order and curvature terms; the $\log \text{BS} \cdot \log(\text{IL} + \text{OL})$ interaction captures the additional memory pressure when long context and large batch co-occur. KV-cache management includes an IL term because the pre-existing context determines the initial cache volume.

Prefill processes the full prompt in one forward pass, so IL and BS are the main workload axes. Attention receives sequence-length curvature terms to capture long-context tiling and memory-hierarchy effects. GEMM receives a finite-size correction based on $\log(\text{IL})/\text{IL}$, which captures the short-sequence regime where fixed overhead and low utilization are not yet amortized.

2.3 Few-shot Calibration

The key structural property of Eq. 1 is engine-local transfer. For a fixed serving engine, the workload slopes $\boldsymbol{\theta}_{f,e}$ are shared across GPU-model stacks, while the effective deployment intercept must be calibrated for each engine-GPU-model stack. KernelScale observes k profiling runs per GPU-model stack, pools these shots to estimate shared family-level workload slopes, and then calibrates the intercept for each deployment using its own k shots.

The same mechanism handles an unseen GPU or model under a supported serving engine. KernelScale reuses the learned workload slopes and re-estimates the target stack's intercepts from a small number of target-side measurements. Zero-shot transfer is not expected to be reliable because absolute latency and energy scale are deployment-specific. A single target configuration can nevertheless anchor the intercept when the workload slopes have already been learned from source stacks, as evaluated in §3.

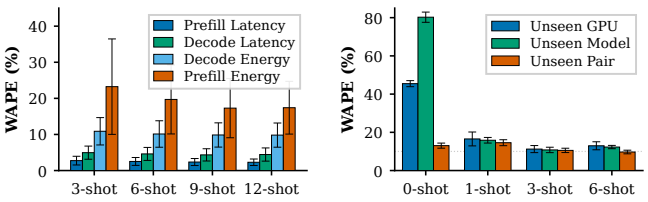
3 Evaluation

Experimental setup. We profile three types of GPUs (RTX 5090, A6000, A800) spanning two microarchitecture generations (Blackwell, Ampere) and a 3× memory-bandwidth range, 3 models (Llama-3.2-3B, Mistral-7B, Qwen2.5-7B) with distinct attention implementations [11, 21], and 2 serving engines (vLLM, SGLang) [13, 30], collecting all traces with NVIDIA Nsight Systems (`nsys`) [19]. The measurement grid contains 4,135 profiling runs in total and covers batch sizes 1–64, input lengths 32–2048, and output lengths 32–512. In the minimum few-shot setting, KernelScale observes only three configurations from each evaluated stack, yielding 54 profiling runs in total, or 1.31% of the grid.

We evaluate prediction error using weighted absolute percentage error (WAPE). Latency and energy span orders of magnitude across the profiling grid, and small short-window values can make per-point percentage errors unstable [12], while WAPE measures aggregate absolute error relative to aggregate measured cost. All evaluations are estimated over 10 random seeds.

3.1 Can few shots recover a deployment map?

This subsection quantifies the minimum profiling budget required for acceptable recovery.



(a) Within source pool. (b) Outside the source pool.

Fig. 3. Performance of few-shot map recovery. Within source pool, prediction error decreases rapidly with profiling budget across all four tasks. Outside the source pool, zero-shot is unreliable for unseen GPUs/models, but a single target configuration calibrates the intercept.

Few-shot protocol. We define one shot as one profiled workload point on one stack: a single `nsys` run at fixed $(e, g, m, \text{BS}, \text{IL}, \text{OL})$. The same run yields prefill and decode labels for both latency and energy, so the profiling budget is shared across all four prediction tasks. For k shots, KernelScale samples k workload points for each deployment stack $d = (e, g, m)$. The 3-shot setting uses one low-, one mid-, and one high-load point, where load is approximated by BS×IL for prefill and BS×IL×OL for decode. For each engine, KernelScale pools the selected shots from its 3×3 GPU-model source pool to fit shared workload slopes, then uses each stack's own k shots to fit its intercept. The selected shots serve as anchors for a workload map. Once the family slopes and deployment intercept are fitted, KernelScale evaluates Eq. 1 at any configuration in the calibrated workload domain.

Three shots recover source-pool stacks. Averaged over the four tasks shown in Fig. 3a, kernel-family models achieve 9.6% mean WAPE with only 3 shots. Increasing the budget to 6 shots reduces error to 9.0%, after which the returns diminish. Prefill energy remains the hardest metric and is the main contributor to the high-error tail in Fig. 3a. Our diagnostic suggests that short-window energy attribution is a major contributor to this tail, rather than workload scaling alone. Prefill stages are short, so stage energy depends on assigning lagged and low-rate power samples to fine-grained execution windows. In our measurement audit, the p95 gap between raw NVML-integrated energy and reconstructed energy reaches 62.0% for vLLM prefill, compared with 10.4% for decode. Fig. 3a further suggests that the kernel-family workload model captures prefill scaling reasonably well, while the remaining energy error is more consistent with short-window power-attribution noise. We therefore treat prefill energy as the main remaining hard case: it affects the per-task WAPE tail, although prefill accounts for a median 12.3% of total energy in our clean grid, and improving it likely requires higher-resolution power sampling.

One selected target shot anchors deployment scale. For a target deployment outside the source GPU-model pool while still within serving engines, KernelScale reuses the learned workload slopes and re-estimates only the target intercept. We evaluate leave-one-GPU (Unseen GPU) and leave-one-model (Unseen Model) holdouts, which remove all source stacks containing the target GPU or model, and a leave-one-pair (Unseen Pair) holdout, which removes only one GPU-model combination while keeping both levels observed through other stacks. Reported numbers are averaged over the two engines.

Table 2. Sensitivity of out-of-source-pool transfer to the load bucket used for one-shot target calibration. Values are mean WAPE (%) ± standard deviation across 10 random seeds, averaged over engines.

Holdout	0-shot	Low	Mid	High
Unseen GPU	45.5±1.6	22.6±6.5	16.5±3.6	14.1±2.2
Unseen model	80.2±2.7	20.1±4.6	15.8±1.5	13.3±2.2

Zero-shot transfer is unreliable for fully unseen levels, with 45.5% WAPE for an unseen GPU and 80.2% for an unseen model (Fig. 3b). One target shot sampled from the middle load bucket reduces WAPE to 16.5% and 15.8%, respectively, capturing 84% and 92% of the improvement achieved with six target shots. We also test whether this single target shot is sensitive to the load region. Table 2 shows that any one-shot calibration substantially improves unseen-GPU and unseen-model transfer over zero-shot, but the exact load region matters: low-load shots are least reliable, likely because fixed overhead dominates small configurations. The middle bucket is therefore a balanced default that avoids the low-load overhead regime while not relying on an extreme high-load point.

The leave-one-pair case is intentionally easier than leave-one-GPU or leave-one-model because the marginal GPU and model scales are still observed. It removes the exact GPU-model pair from source training while keeping the target GPU visible with other models and the target model visible on other GPUs under the same engine. Thus zero-shot prediction cannot use a pair-specific intercept, but it can compose the learned GPU intercept, the learned model intercept, and the shared workload slopes. KernelScale achieves 13.0% zero-shot WAPE. This lower error is expected and supports the intended interpretation of the intercept terms as reusable scale components rather than pair identifiers.

3.2 Why does three-shot recovery work?

Scaling laws, not pooling alone, make three-shot recovery work. We isolate the mechanism using the same three physical shots for all methods. Local direct-total fitting is not identifiable with only three configurations per stack, because the local design matrix is rank deficient under this budget. However, pooling alone is not sufficient: a pooled XGBoost regressor [4] reaches 38.8% mean WAPE. The main gain comes from the workload scaling law, since a pooled Direct-Total scaling-law model reaches 11.7% WAPE without kernel-family decomposition. Family-specific scaling further reduces WAPE to 9.6%. Thus, three-shot recovery works because pooling identifies compact scaling laws, and kernel families add a smaller but stable composition-aware gain.

The learned structure reflects physical regularity, not local fit. The preceding ablation shows what enables few-shot recovery under a fixed budget. We next ask whether the scaling-law structure genuinely captures workload physics or merely overfits the sampled points. To separate structural benefit from few-shot sampling effects, we compare methods trained on the full grid. XGBoost achieves the lowest interpolation WAPE (4.3%) but collapses to 36.2% under

Table 3. Profiling cost to match KernelScale's three-shot map recovery. #Shots is the number of profiling runs per deployment stack. #Configs counts total number of profiling runs. The whole-map block evaluates our target engine-GPU-model maps with variable batch size, input length, and output length. The fixed-BS scope evaluates a fixed-batch slice closer to Wilkins et al.'s modeling setting.

Scope	Method	#Shots	#Configs	Energy (kJ)	WAPE (%)
Whole-map	XGBoost	44	792	960	9.2
	Wilkins OLS	128	2,304	2,815	34.6
	KernelScale	3	54	64	9.6
Fixed BS	XGBoost	12	216	314	12.2
	Wilkins OLS	7	126	170	4.6
	KernelScale	3	54	79	4.7

boundary-frontier extrapolation, where parameter-space extremes are held out entirely. Direct-Total, a single stage-level scaling-law model, interpolates at 8.6% and degrades only to 14.0%, confirming that the scaling-law form preserves the global trend needed for stable extrapolation. KernelScale further improves over Direct-Total in both regimes (7.7% / 11.8% vs. 8.6% / 14.0%). Direct-Total error correlates strongly with composition shift ($r = 0.50$–0.56), whereas KernelScale is nearly uncorrelated ($r = 0.03$–0.13), suggesting that per-family decomposition isolates the kernel-mix shifts illustrated in Fig. 2 that a monolithic model conflates.

3.3 How much profiling cost does KernelScale avoid?

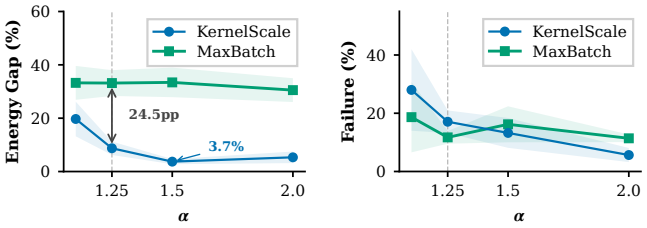
The preceding results show that KernelScale can recover deployment maps from only three profiled configurations per deployment stack. This subsection quantifies the profiling-cost and energy savings achieved by KernelScale over recent workload-level energy models and black-box predictors.

We compare KernelScale against two representative baselines that capture prior workload-level modeling and data-driven black-box prediction. The closest prior baseline is Wilkins' workload-based model (Wilkins OLS), adapted from Wilkins et al. [24]. Their model fits total energy and runtime from input tokens, output tokens, and their interaction:

$$\hat{y} = \beta_0 + \beta_1 \tau_{\text{in}} + \beta_2 \tau_{\text{out}} + \beta_3 \tau_{\text{in}} \tau_{\text{out}},$$

where $\hat{y}$ is either latency or energy. In the original fixed-batch setting, τ_{in} and τ_{out} denote the input and output token counts, and the model is fit as a direct-total workload model for a model/system combination. To apply this baseline to our broader map-recovery task, we preserve its model form and expose only the variable workload axes: for each engine-GPU-model deployment stack, we set $\tau_{\text{in}} = \text{BS} \cdot \text{IL}$ and $\tau_{\text{out}} = \text{BS} \cdot \text{OL}$, then fit separate direct-total OLS models for latency and energy. As a second baseline, we train XGBoost on the same workload axes as a strong black-box predictor, enabling us to quantify the profiling-energy savings of KernelScale relative to a flexible data-driven model [28]. All methods use the same sampling policy, and all costs are computed from the corresponding physical profiling runs.

Table 3 reports results under two evaluation scopes. The upper block corresponds to our target setting: recovering an engine-GPU-model map over variable batch size, input length, and output length.



(a) Feasible energy gap. (b) Constraint failure rate.

Fig. 4. Decision quality vs. latency headroom α. Lower energy gap indicates better energy-aware selection. Constraint failure captures the remaining latency-risk cost of prediction error.

In this setting, KernelScale achieves 9.6% WAPE with only 54 profiling runs. Wilkins OLS fails to reach comparable accuracy across the scanned budgets: even at the largest budget, $k = 128$, it consumes 2,815 kJ of profiling energy while still yielding 34.6% WAPE. XGBoost first reaches comparable accuracy to KernelScale's three-shot result at $k = 44$, but requires 960 kJ of profiling energy. Thus, compared with a strong black-box predictor, KernelScale achieves comparable map-recovery accuracy while using only 7% of the profiling energy.

The lower block evaluates a narrower fixed-batch slice closer to the setting of Wilkins et al. In this regime, Wilkins OLS attains comparable overall WAPE only at $k = 7$, after consuming 170 kJ of profiling energy. Even under this fixed-batch setup, KernelScale requires fewer profiling runs and reduces profiling energy by 54%.

3.4 From energy maps to energy-aware selection

We next test whether recovered maps preserve deployment decisions. For each workload bucket, the measured oracle selects the lowest-energy configuration whose measured latency is within a headroom factor α of the fastest measured configuration. Under the same 3-shot budget used in §3.1, the predicted map applies the same rule using predicted latency and energy. We report measured energy gap and constraint failure under the same three-shot budget.

Fig. 4 shows that energy visibility matters. MaxBatch uses KernelScale's predicted latency map but selects the largest feasible batch size without energy prediction, so the comparison isolates the energy visibility rather than confounding it with latency-prediction quality. At $\alpha = 1.25$, it incurs a 33.2% feasible energy gap, while KernelScale reduces the gap to 8.7%. At $\alpha = 1.50$, KernelScale reaches a 3.7% gap and 13.2% constraint failure. The result is not a full scheduling claim. It shows that the recovered map preserves the energy ranking needed by latency-constrained and carbon-aware deployment policies.

4 Related Work

Recent systems have explored energy- and carbon-aware LLM serving through scheduling, batching, disaggregation, and GPU power management [3, 16, 23]. These systems demonstrate that inference energy can be substantially reduced through scheduling and runtime adaptation, but generally assume that accurate deployment-specific energy-performance information is already available.

To characterize LLM inference cost, prior works model latency and energy using workload-level regression, profiled measurements, analytical inference modeling, or operator-level decomposition [6, 24]. Several studies further perform kernel- or primitive-level energy modeling by decomposing inference into operations such as GEMM, attention, communication, and memory movement [25]. While these approaches improve modeling fidelity and deployment exploration, most still require dense per-stack profiling, detailed analytical specifications, or stack-specific calibration before deployment maps can be constructed.

In contrast, KernelScale targets the map-construction problem itself. Rather than relying on exhaustive profiling or deployment-specific analytical modeling, KernelScale recovers the maps from a few measured configurations by exploiting transferable kernel-family workload-scaling structure.

5 Discussion

KernelScale is currently validated on single-node decoder-only serving. The reusable component is the engine-local kernel-family workload scaling law: workload slopes are largely determined by computation patterns such as attention, GEMM, and memory movement, while GPU microarchitecture, model scale, and runtime overhead are absorbed by low-dimensional deployment intercepts. New execution structures require extending the family set rather than changing the recovery principle. Multi-node serving would add communication and synchronization families, such as tensor-parallel collectives and interconnect-dependent all-reduce/all-to-all costs. MoE models would add routing-dependent expert GEMM, load imbalance, and possibly all-to-all transfer. Multimodal models would add modality-specific encoder and cross-attention families. We leave empirical validation of these families to future work.

Latency recovery is well captured by workload scaling laws, while energy recovery remains harder because per-stage energy estimation requires attributing coarse, lagged power samples to fine-grained execution windows. Because prefill stages are short, their energy labels are more sensitive to sampling and alignment noise. This indicates that energy-map fidelity is also limited by measurement resolution, rather than solely by the expressiveness of the workload-scaling model.

6 Conclusion

KernelScale shows that energy-latency maps for LLM inference can be recovered with few-shot profiling. By decomposing inference runtime into recurring kernel families and modeling them with scaling law structure, KernelScale learns reusable workload slopes from pooled few-shot measurements and calibrates each GPU-model stack with a small deployment intercept. Across 3 GPUs, 3 models, 2 serving engines, and 4,135 measured configurations, three profiling configurations per stack recover maps with 9.6% mean WAPE. The XGBoost regressor requires 15× more profiling runs and energy cost to match this accuracy. The results position KernelScale as a low-cost profiling substrate for energy-aware LLM serving. Future work will extend this validation to MoE, multimodal, and multi-node deployments, and investigate higher-resolution power sampling for short-window prefill-energy attribution.

Acknowledgments

We thank the anonymous reviewers for their constructive feedback. This work was supported in part by the Startup Fund of The Hong Kong University of Science and Technology (Guangzhou), the Guangdong Basic and Applied Basic Research Foundation (Nansha Joint Fund), the Guangdong Major Project of Basic Research (Grant No. 2026B0303000007), and the National Natural Science Foundation of China (Grant No. U23B2004).

References

[1] Amey Agrawal, Nitin Kedia, Jayashree Mohan, Ashish Panwar, Nipun Kwatra, Bhargav S. Gulavani, Ramachandran Ramjee, and Alexey Tumanov. 2024. Vidur: A Large-Scale Simulation Framework for LLM Inference. Proceedings of Machine Learning and Systems (MLSys). https://proceedings.mlsys.org/paper_files/paper/2024/hash/b74a8de47d2b3c928360e0a011f48351-Abstract-Conference.html

[2] Amey Agrawal, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav S Gulavani, and Ramachandran Ramjee. 2023. Sarathi: Efficient llm inference by piggybacking decodes with chunked prefills. *arXiv preprint arXiv:2308.16369* (2023).

[3] Omar Basit, Yunzhao Liu, Z. Jonny Kong, and Y. Charlie Hu. 2026. DualScale: Energy-Efficient Disaggregated LLM Serving via Phase-Aware Placement and DVFS. arXiv:2602.18755 doi:10.48550/arXiv.2602.18755

[4] Tianqi Chen and Carlos Guestrin. 2016. XGBoost: A Scalable Tree Boosting System. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 785–794. doi:10.1145/2939672.2939785

[5] Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness. *Advances in neural information processing systems* 35 (2022), 16344–16359.

[6] Anurag Dutt, Young Won Choi, Avirup Sil, Anshul Gandhi, Aruna Balasubramanian, and Niranjan Balasubramanian. 2025. Fine-Grained Energy Prediction for Parallelized LLM Inference with PIE-P. arXiv:2512.12801 https://arxiv.org/abs/2512.12801

[7] Mauricio Fadel Argerich, Jonathan Furst, and Marta Patino-Martinez. 2026. Watt Counts: Energy-Aware Benchmark for Sustainable LLM Inference on Heterogeneous GPU Architectures. arXiv:2604.09048 https://arxiv.org/abs/2604.09048

[8] Md. Monzurul Amin Ifath and Israat Haque. 2026. Characterizing Performance-Energy Trade-offs of Large Language Models in Multi-Request Workflows. arXiv:2604.09611 https://arxiv.org/abs/2604.09611

[9] International Energy Agency. 2025. Energy and AI. https://www.iea.org/reports/energy-and-ai IEA, Paris.

[10] Aaron Jarmusch and Sunita Chandrasekaran. 2026. Microbenchmark-Driven Analytical Performance Modeling Across Modern GPU Architectures. arXiv:2605.04178 https://arxiv.org/abs/2605.04178

[11] Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023. Mistral 7B. arXiv:2310.06825 [cs.CL] https://arxiv.org/abs/2310.06825

[12] Stephan Kolassa and Wolfgang Schütz. 2007. Advantages of the MAD/Mean Ratio over the MAPE. *Foresight: The International Journal of Applied Forecasting* 6 (2007), 40–43.

[13] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient Memory Management for Large Language Model Serving with PagedAttention. arXiv:2309.06180 [cs.LG] https://arxiv.org/abs/2309.06180

[14] Baolin Li, Siddharth Samsi, Vijay Gadepally, and Devesh Tiwari. 2023. Clover: Toward Sustainable AI with Carbon-Aware Machine Learning Inference Service. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*. doi:10.1145/3581784.3607034

[15] Dimitrios Liakopoulos, Tianrui Hu, Prasoon Sinha, and Neeraja J. Yadwadkar. 2025. iServe: An Intent-based Serving System for LLMs. arXiv:2501.13111 [cs.SE] https://arxiv.org/abs/2501.13111

[16] Qunyou Liu, Darong Huang, Marina Zapater, and David Atienza. 2025. GreenLLM: SLO-Aware Dynamic Frequency Scaling for Energy-Efficient LLM Serving. arXiv:2508.16449 doi:10.48550/arXiv.2508.16449

[17] Paul Joe Maliakel, Shashikant Ilager, and Ivona Brandic. 2025. Characterizing LLM Inference Energy-Performance Tradeoffs across Workloads and GPU Scaling. arXiv:2501.08219 https://arxiv.org/abs/2501.08219

[18] NVIDIA Corporation. 2026. *CUDA C++ Best Practices Guide*. https://docs.nvidia.com/cuda/cuda-c-best-practices-guide/index.html Accessed 2026-05-19.

[19] NVIDIA Corporation. 2026. *NVIDIA Nsight Systems User Guide.* https://docs.nvidia.com/nsight-systems/UserGuide/index.html Accessed 2026-05-18.

[20] NVIDIA Corporation. 2026. *NVML API Reference Guide.* https://docs.nvidia.com/deploy/nvml-api/index.html Accessed 2026-05-18.

[21] Qwen Team. 2025. Qwen2.5 Technical Report. arXiv:2412.15115 [cs.CL] https://arxiv.org/abs/2412.15115

[22] Jovan Stojkovic, Esha Choukse, Chaojie Zhang, Inigo Goiri, and Josep Torrellas. 2024. Towards Greener LLMs: Bringing Energy-Efficiency to the Forefront of LLM Inference. arXiv:2403.20306 https://www.microsoft.com/en-us/research/publication/towards-greener-llms-bringing-energy-efficiency-to-the-forefront-of-llm-inference/

[23] Jovan Stojkovic, Chaojie Zhang, Íñigo Goiri, Josep Torrellas, and Esha Choukse. 2024. DynamoLLM: Designing LLM Inference Clusters for Performance and Energy Efficiency. arXiv:2408.00741 doi:10.48550/arXiv.2408.00741 Related DOI: 10.1109/HPCA61900.2025.00102.

[24] Grant Wilkins, Srinivasan Keshav, and Richard Mortier. 2024. Offline Energy-Optimal LLM Serving: Workload-Based Energy Models for LLM Inference on Heterogeneous Systems. In *Proceedings of the 3rd Workshop on Sustainable Computer Systems (HotCarbon).* https://hotcarbon.org/assets/2024/pdf/hotcarbon24-final30.pdf

[25] Tianhao Xu, Yiming Liu, Xianglong Lu, Yijia Zhao, Xuting Zhou, Aichen Feng, Yiyi Chen, Yi Shen, Qin Zhou, Xumeng Chen, Ilya Sherstyuk, Haorui Li, Rishi Thakkar, Ben Hamm, Yuanzhe Li, Xue Huang, Wenpeng Wu, Anish Shanbhag, Harry Kim, Chuan Chen, and Junjie Lai. 2026. AIConfigurator: Lightning-Fast Configuration Optimization for Multi-Framework LLM Serving. arXiv:2601.06288 https://arxiv.org/abs/2601.06288

[26] Zihao Ye, Lequn Chen, Ruihang Lai, Wuwei Lin, Yineng Zhang, Stephanie Wang, Tianqi Chen, Baris Kasikci, Vinod Grover, Arvind Krishnamurthy, et al. 2025. Flashinfer: Efficient and customizable attention engine for llm inference serving. *Proceedings of Machine Learning and Systems* 7 (2025).

[27] Kaixuan Zhang, Yunfan Cui, Shuhao Zhang, Chutong Ding, Shiyou Qian, Luping Wang, Jian Cao, Guangtao Xue, Cheng Huang, Guodong Yang, and Liping Zhang. 2026. PipeWeave: Synergizing Analytical and Learning Models for Unified GPU Performance Prediction. arXiv:2601.14910 doi:10.48550/arXiv.2601.14910 Accepted to ISCA 2026.

[28] Yijia Zhang, Zhihong Gou, Shijie Cao, Weigang Feng, Sicheng Zhang, Guohao Dai, and Ningyi Xu. 2024. Automating Energy-Efficient GPU Kernel Generation: A Fast Search-Based Compilation Approach. arXiv:2411.18873 doi:10.48550/arXiv.2411.18873

[29] Haiying Zhao, Cheng Li, Shenggui Lin, and You Yang. 2025. EcoServe: Designing Carbon-Aware AI Inference Systems. arXiv:2503.00657 https://arxiv.org/abs/2503.00657

[30] Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. 2024. SGLang: Efficient Execution of Structured Language Model Programs. arXiv:2312.07104 [cs.AI] https://arxiv.org/abs/2312.07104