

Power-State Variance in One WGPU Vector-Search Kernel: A Low-Cost Case Study with Qenlo

Akshat Singh Kushwaha

akshatsingh14372@outlook.com

Abstract. GPUs lower their clocks under light load, which is why careful kernel benchmarks lock clocks before timing. We measure what this does to a comparison between two GPU paths of one exact filtered vector-search engine, Qenlo, that return identical top- k results. A *light* path does device work proportional to the E eligible rows; a deliberately naive *heavy* path scans all N rows and keeps the GPU at boost. Across fresh processes on an RTX 4050 Laptop GPU, RTX 4090s (one archived host and five cloud pods) and an H100, the heavy kernel’s device time varied by at most 1% between processes on one machine and 7% across the five pods. The light kernel’s varied up to $4.2\times$ on the laptop (at $E = 3,000$) and $4.6\times$ across the pods; at $E = 10,000$, where the light path beats our single-threaded CPU scan on the laptop and pods, the spreads were $1.4\times$ and $4.6\times$. On the laptop and H100, telemetry shows the driver running the light path at low clocks. On the laptop, in an experiment whose protocol was pushed before its data, locking the clocks removed the variation ($2.8\times$ to $1.00\times$ across blocks) and a low lock reproduced the slow mode, although one of five predictions failed; a second such experiment that locked the SM and memory clocks separately traced the selection kernel’s slow mode mainly to the SM clock. Clocks could not be locked on the cloud hosts. There, a separate light GPU load made slow light-kernel runs rarer (clearly on the laptop, by a few runs on the pods and H100), but it also loaded the host, its planned end-to-end prediction failed, and sampled clocks did not track kernel time quantitatively, so the cross-host account is weaker. The cross-host comparison was exploratory. The magnitudes belong to this engine and these machines. The study ran on one laptop and about US\$6 of cloud time; protocols, raw data and telemetry are released.

1 Introduction

GPUs drop to lower clocks when they are lightly loaded. Kernel benchmarking practice knows this, and locks clocks or warms the device up before timing (NVIDIA, 2026b,c,d). Application-level benchmarks usually do neither. They compare implementations at the driver’s default clocks, and the implementations being compared can load the GPU very differently. A work-efficient kernel that leaves the GPU mostly idle can be clocked down; a wasteful one that keeps it busy is not. The measured gap between them then partly reflects the power state each one induced. We report one case of this and test it.

The case comes from Qenlo, an embedded vector store with two GPU paths for exact filtered top- k search that return identical results. A *light* path uploads the list of E eligible rows and does device work proportional to E . A *heavy* path evaluates the filter inside the kernel and scans all N rows; its selection kernel makes k passes over all N scores in one workgroup. The heavy path is not a competitive baseline. We keep it because it computes the same answer while holding the GPU at boost, which makes it a load contrast and a negative control. At $E = 10,000$ eligible rows out of $N = 100,000$, the light path is faster than both the heavy path and our single-threaded CPU scan on the laptop and the RTX 4090 pods (Appendix Table 8), so this is a regime where an application would choose it. (Part of that advantage comes from the CPU scan switching to slower FP64 accumulation above $E = 4,096$; Section 2.) There its kernel time varied

$1.4\times$ between fresh processes on the laptop and $4.6\times$ across five RTX 4090 cloud pods, while the heavy kernel’s varied by at most 1% on one machine and 7% across the pods (Section 4). At smaller E the CPU wins and the light path’s spread matters less in practice, although it is as large.

The direct test is to lock the clock. That needs administrator rights, which we had on the laptop. There, in an experiment whose protocol and predictions were pushed to the public repository before its first run (Section 7), locking the SM and memory clocks at boost removed the light kernel’s spread across processes, and locking them at the light path’s observed low state reproduced its slow mode. One of its five predictions failed: locked at boost, the light kernel took 32–50% less time than the best unlocked median, outside the 15% we had predicted.

Inside the cloud containers clocks could not be locked, so there we tried to raise the clock instead. A separate process, the *heater*, keeps the GPU lightly busy. If the variation came only from contention, the heater could only slow every path down; if it came from power state, it should speed up work that was running below its attainable clock. At the device level that is the direction we saw on all three GPU types: light-kernel runs more than $1.5\times$ slower than their best fell from 13 of 25 to 0 on the laptop, 7 to 4 of 25 on the RTX 4090 pods and 1 to 0 of 9 on the H100, and the heavy kernel never got faster (Figure 1c, Section 6). The heater is a crude instrument. It also uses host CPU, it slowed host-side work by 40–48% on the H100 pod at $E \leq 3,000$, and the end-to-end speed-up we had planned to see did not appear on any GPU.

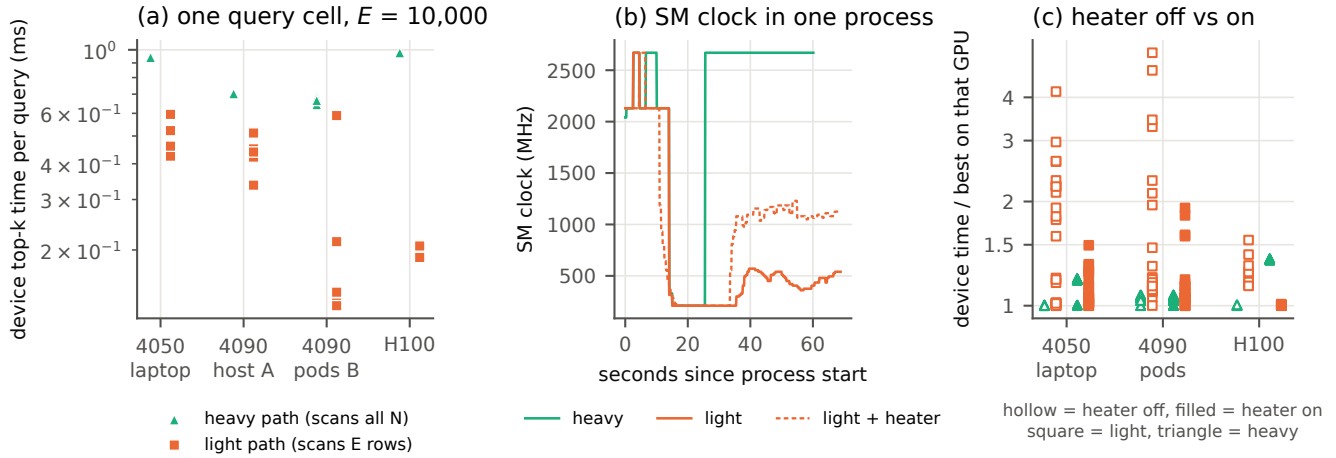


Figure 1: **On the laptop, the light kernel’s speed tracks the GPU clock.** (a) Device time of the top- k kernel for one query cell ($N = 100K$, $E = 10,000$, $D = 384$, $k = 10$, batch 1), one point per fresh process. At this E the light path is faster than the CPU scan on the laptop and pods (host A and the H100 had no CPU arm). The heavy path does $10\times$ more work and reproduces within 1% on a machine and 7% across the five RTX 4090 pods; the light path varies up to $1.5\times$ on a machine and $4.6\times$ across the pods. (b) SM clock sampled every 50 ms during one laptop process per condition ($E = 10,000$), including data loading. The heavy path stays at boost and the light path drops to about 500 MHz; a separate light GPU load (the heater) raises it to about 1.1 GHz, still below boost. (c) Light (squares) and heavy (triangles) device time relative to the best value observed for that GPU and E , heater off (hollow) and on (filled), one point per block and E . With the heater, the light kernel’s runs more than $1.5\times$ slow fall from 13 to 0 of 25 on the laptop, 7 to 4 of 25 on the pods and 1 to 0 of 9 on the H100, and the heavy kernel never gets faster.

Its protocols were written before the data but committed only afterwards (Appendix A). We treat the cross-host account as supported by telemetry and the heater, not as tested.

Why a small engine. Everything here ran on one laptop plus about US\$6 of rented GPU time. That was possible because Qenlo is small enough to instrument end to end: device timestamps on every call, a fresh process per measurement, continuous clock telemetry, and two code paths with identical outputs that load the GPU differently. With those in place, finding the effect, testing it by locking clocks and releasing every sample took days. We think small, fully instrumented engines are a cheap way to do this kind of systems measurement, and we offer Qenlo as one.

Contributions.

- **An observation in one engine.** For identical results, Qenlo’s work-efficient GPU path is the less reproducible one. Its kernel time varies up to $4.2\times$ between fresh processes on the laptop (at $E = 3,000$; $1.4\times$ at $E = 10,000$) and $4.6\times$ across five same-model cloud hosts, while the heavy path’s varies by at most 1% on one machine and 7% across those hosts (Section 4).
- **A locked-clock test on one laptop.** With its protocol committed before the data, locking the clocks removed the light kernel’s variation and a low lock reproduced the

slow mode. Locking the SM and memory clocks separately showed that the SM clock accounts for most of it; the memory clock matters only for the scoring kernel at the largest E (Section 7).

- **Weaker evidence across hosts.** The heater’s direction is consistent with the same account on RTX 4090 and H100 hosts, but sampled clocks do not track kernel time there quantitatively, and the planned end-to-end prediction failed; we report both (Sections 5–6).
- **Reporting rules and data.** We consolidate existing rules for benchmarking latency-bound GPU work and add two (Section 8), and release protocols, raw samples, telemetry and hash manifests.

2 The query paths and why their load differs

We study Qenlo, an embedded Rust vector store that answers exact (exhaustive) filtered cosine top- k queries over a canonical collection of N normalized FP32 vectors of dimension D with per-row metadata. A predicate selects E eligible rows; the query returns the k eligible rows nearest to the query. Every execution path below is gated at FP64-oracle recall@10 ≥ 0.99 . In every call of every run reported here observed recall@10 was 1.0 and no filter was violated, except in the

September 24 host A archive, where one run’s evaluation recall was 0.99998. For these runs the choice of path therefore affects only latency.

CPU scan. A single-threaded AVX2 scan over the eligible rows. For $E \leq 4,096$ it scores in FP32 and rescores boundary candidates in FP64; above that it accumulates in FP64 throughout. The kernel switch is part of the implementation and applies equally to every condition we compare, but it favors the GPU in CPU/GPU comparisons: the CPU’s cost per eligible row rises about $3.3\times$ between $E = 3,000$ and $10,000$ (Appendix B). Like FAISS at batch 1, the scan uses one thread. It is a conventional baseline, not a latency-optimal one.

GPU paths (WGPU/WGSL). Vectors, identifiers and metadata are resident on the device. Each query uploads the query vector and an eligibility description, runs a scoring kernel and a top- k selection kernel, and blocks on a readback of k candidates, which the host merges. The paths differ only in how eligibility reaches the device:

- *Compact rows (the light path).* The host materializes the sorted list of E eligible row ids and uploads $4E$ bytes. Scoring and selection touch only those E rows, so device work is proportional to E .
- *Shader predicate (the heavy path).* The host uploads per-row liveness ($4N$ bytes) and the predicate bounds; the kernels run over all N rows and evaluate the predicate in the shader. Selection scans all N scores k times in one workgroup, so selection work is proportional to N regardless of E (scoring time still grows somewhat with E ; Table 5).

The heavy path therefore does up to N/E times more device work for the same answer ($1,000\times$ at $E = 100$, $N = 10^5$). This selection kernel is deliberately simple and far from the best available; a tuned top- k would narrow the gap. We use the heavy path as a load contrast and a negative control, not as a baseline: the two paths put very different loads on the GPU while computing identical results.

Timing boundary. Latency is measured from the host search call to host-visible ordered results, including eligibility preparation, upload, dispatch, synchronization, readback and merge. Device-side scoring and selection times come from GPU timestamp queries in the same calls; they overlap host phases and are not additive. Host row materialization is timed separately on the CPU.

3 Experimental design

Workload. The corpus is rows 0–99,999 of the all-MiniLM-L12-v2 AG News training-split embeddings ($D = 384$; Nielsen, 2023, CC BY 4.0); the queries are 5,000 later rows of the same split (101,000–105,999), disjoint from the

corpus but not deduplicated. The underlying AG corpus is distributed for non-commercial research use. Synthetic unique timestamps are attached to rows and a timestamp range selects exactly $E \in \{100, 1K, 3K, 10K, 30K\}$ eligible rows (0.1%–30%). Batch size is 1 and $k = 10$, the regime of an interactive retrieval or agent-memory lookup.

Unit of replication. Every condition runs in a fresh process: dataset load, 200 warm-up queries, then three passes over the 5,000 queries (15,000 timed calls). A process’s device time is the median over its 15,000 calls; its end-to-end P50 is the median of the three per-pass nearest-rank P50s. A *block* runs every condition once in a shuffled order; blocks are the independent replicates. Paired effects are computed within a block and summarized by the median over blocks of per-block log-ratios. We also report percentile-bootstrap intervals over blocks (20,000 draws), but with three to five blocks such an interval is just the range of the block ratios: it covers the population median with probability about 75% ($n = 3$) or 94% ($n = 5$), not 95%, and no exact sign test on five blocks can reach $p < 0.05$. We read these numbers descriptively. The 15,000 calls within a process are repeated measurements and are never treated as independent.

Telemetry. `nvidia-smi` samples SM clock, memory clock, performance state (P-state), utilization and power every 50 ms for the whole life of each process. At that interval short kernels fall between samples, and `nvidia-smi`’s power readings are themselves partial on some GPUs (Yang et al., 2024); we use the clocks to identify states, not to time kernels. We did not record clock-event (throttle) reasons or the driver’s power-management mode.

Intervention: a GPU heater. To move the GPU’s power state without touching the measured process, a separate process (the *heater*) repeatedly dispatches a small compute kernel (64 workgroups $\times$ 256 threads, 500 dependent FMAs each), waits for it, and sleeps 1 ms. Its own reported utilization is a few percent. On the GPU the heater can only compete with the measured process, so a speed-up it produces shows that the measured process was not capacity-bound. A lower power state is the most direct explanation; keeping the GPU out of deep idle (and its wake-up latency) or keeping driver state warm would act the same way. Keeping a GPU busy with dummy work is a known practitioner remedy for idle slowdowns (NVIDIA Developer Forums, 2022; Volkov, 2016); here we use it as an instrument. The heavy path, which already holds a high P-state, is the negative control. If contention were the only effect, the heater would slow both paths; if power state matters, it should speed up the light path and slow only the heavy one. The heater also uses host CPU, so it is not a clean GPU-only perturbation; its host cost shows up in end-to-end latency much more than in device timestamps (Section 6).

Hosts and protocols. (i) A Windows 11 laptop (Intel i5-13420H, RTX 4050 Laptop GPU, driver 616.56, Vulkan, AC power, “Balanced” power plan). It uses hybrid graphics (Optimus); the display is driven by the Intel iGPU. (ii) RTX 4090s on Linux with Vulkan: *host A*, an archive of five blocks per cell on one host from September 24 (at $E = 3,000$ the light path also has ten earlier blocks from the same archive), and *Pods B*, five pods on five different hosts from September 25 with one block each. The pods use two driver branches (570.195 and 580.126–580.173). Because each pod ran one block, host and process variance are confounded on the pods. (iii) One H100 SXM pod (Linux, Vulkan) running a reduced design. A pod whose host CPU was about $14\times$ slower than the others was replaced on that host-speed check (Appendix A). The laptop, pod and H100 experiments (D1, D1R, H100) follow written protocols with predictions, but those were committed to version control only after the data; we call them planned analyses. The two locked-clock protocols were pushed to the public repository before their first runs. Clock locking (`nvidia-smi -lgc`) requires administrator rights; it was available on the laptop (Section 7) and refused inside the cloud containers.

4 The heavy kernel reproduces; the light kernel does not

Table 1 reports the median device time of the top- k selection kernel per fresh process, for both paths at three selectivities on four GPUs. We use device timestamps rather than end-to-end latency because they isolate the GPU from host-side effects; Section 6 returns to end-to-end latency.

At the E values in Table 1 the heavy kernel reproduces within 1% between processes on each single machine (the laptop, 4090 host A and the H100), and within 7% across the five pod hosts, which span two driver branches (570 and 580). The light kernel does not. Its spread between processes reaches $2.6\times$ on 4090 host A and $4.2\times$ on the laptop (both at $E = 3,000$), and $2.1\text{--}4.6\times$ across the five 4090 pods ($4.3\text{--}4.6\times$ at $E \geq 3,000$). The per-process values look bimodal (Figure 1a), so a median of three to five processes mostly reports which mode most of them landed in. Host A’s and the pods’ ranges overlap at every E ; we read the difference between their medians as a difference in how often each landed in the slow mode, not as a stable property of the host. The light path is faster than the heavy path in every cell.

The retained archive shows the same pattern within one host over time. Across the ten earlier identical blocks at $E = 3,000$, the light path’s selection time spans $2.44\times$ and correlates with the SM clock in an `nvidia-smi` snapshot taken just after each process ended (Spearman $\rho = -0.83$, exact two-sided $p = 0.005$). That clock is a coarse, end-of-process proxy, the correlation rests largely on the two fast blocks, and the clock is partly endogenous (Section 5). The heavy kernels in the same archive vary by at most 0.01% at $E \leq 10,000$ and 0.7% at $E = 100,000$, where their clock was at boost, but by 7% at $E = 30,000$, where it was not.

5 Which power state each path gets

Table 2 reports the median SM and memory clocks while the GPU was active (samples with non-zero utilization) for each path. On every GPU the heavy path holds boost SM clocks: 2,670–2,820 MHz on the consumer GPUs and 1,980 MHz on the H100. The light path runs lower. At $E = 10,000$ the driver also cuts the memory clock to 810 MHz on the laptop (from 8,201 MHz; performance state P5) and on the 4090 pods (from 10,501 MHz). (The heavy path’s own memory clock sat at 7,001 MHz in the laptop’s $E = 3,000$ cell.) Here both clocks fall together, so telemetry alone cannot say which of them slows the kernels; Section 7 separates them. On the H100 the light path’s SM clock falls from 1,605 to 645 MHz between $E = 1,000$ and 10,000. On the laptop, sampled clocks predict the heater’s effect on device time well. On the cloud hosts they do not: the pods’ median light-path SM clock is 79–89% of boost, which cannot by itself produce a $4.6\times$ spread, and on the H100 at $E = 10,000$ the heater raised the median clock $3.1\times$ but sped the kernel only $1.23\times$. Telemetry at this resolution supports the power-state account on the laptop and is only consistent with it elsewhere.

The governor’s inputs are undocumented, and we do not try to model them; we report what we observe. At default clocks the heavy path spends 52–84% of each call inside its two kernels and always holds boost; the light path spends 13–43% and does not (`research/scripts/busy_fraction.py`). Within the light path the clock *falls* as E grows on the laptop and the H100, even though per-call device work grows. That default-clock share is endogenous, because a lower clock lengthens the kernels that define it. At locked boost on the laptop (Tables 4 and 5 over the call times of Appendix Table 8) the kernels fill 13.5%, 10.4% and 8.7% of a call at $E = 1,000$, 3,000 and 10,000, because host-side preparation of the E ids grows faster than device work. A governor that reacts to GPU utilization would therefore lower the clock as E grows, as we observe. We did not test this directly, for example by varying the host gap between queries at a fixed E . The NVIDIA policy is undocumented (Spaan et al., 2026), and our claims concern its observable consequences.

6 Raising the load where clocks could not be locked

Each experiment ran every condition (engine $\times E \times$ heater on/off) once per block, in shuffled order and fresh processes (Section 3). Table 3 gives the most direct test. For every block and E , it divides each path’s device time by the best value observed for that GPU and E in either arm. Both arms share the same reference, so a heater-off run that happened to be slow cannot flatter the heater through regression to the mean.

Device time: the planned interaction appears on every GPU, clearly only on the laptop. Without the heater, the light kernel runs a median $1.15\text{--}1.59\times$ slower than its best,

Table 1: Top- k kernel device time per query (ms): median over fresh heater-off processes (min–max), and max/min across processes. Heavy = shader predicate over all $N = 100K$ rows; light = compact eligible rows. RTX 4090 host A is the retained September 24 archive (one host, 5 blocks; at $E = 3,000$ the light column also includes ten earlier blocks from the same archive, hence $n = 5/15$); pods B are five September 25 pods on five different hosts, one block each. Identical binary source, dataset bytes and queries.

E	GPU	n	Heavy (ms)	max/min	Light (ms)	max/min
1,000	RTX 4050 Laptop	5	0.950 (0.949–0.954)	1.01	0.030 (0.030–0.030)	1.00
1,000	RTX 4090 host A	5	0.711 (0.711–0.711)	1.00	0.053 (0.051–0.065)	1.27
1,000	RTX 4090 pods B	5	0.681 (0.650–0.695)	1.07	0.028 (0.027–0.056)	2.08
1,000	H100 SXM	3	0.987 (0.982–0.987)	1.00	0.039 (0.039–0.039)	1.02
3,000	RTX 4050 Laptop	5	0.952 (0.951–0.956)	1.01	0.106 (0.055–0.230)	4.16
3,000	RTX 4090 host A	5/15	0.711 (0.711–0.711)	1.00	0.210 (0.098–0.251)	2.56
3,000	RTX 4090 pods B	5	0.685 (0.652–0.700)	1.07	0.055 (0.050–0.216)	4.30
3,000	H100 SXM	3	0.988 (0.988–0.988)	1.00	0.094 (0.088–0.104)	1.18
10,000	RTX 4050 Laptop	5	0.941 (0.940–0.941)	1.00	0.461 (0.425–0.595)	1.40
10,000	RTX 4090 host A	5	0.703 (0.703–0.703)	1.00	0.440 (0.337–0.512)	1.52
10,000	RTX 4090 pods B	5	0.673 (0.643–0.687)	1.07	0.142 (0.128–0.589)	4.61
10,000	H100 SXM	3	0.977 (0.972–0.977)	1.01	0.205 (0.188–0.206)	1.10

Table 2: SM / memory clock (MHz) while the GPU is active: median over blocks of each process’s median over its active samples (utilization > 0 , sampled every 50 ms). Pooling all samples instead tells a harsher story on the pods, where the light path often sat at 810 MHz memory (e.g. 1,245/810 at $E = 10,000$); a median process can hide a slow minority.

GPU	E	Heavy path	Light path	Light path + heater
RTX 4050 Laptop	1,000	2670 / 8201	2130 / 8201	2130 / 8201
RTX 4050 Laptop	3,000	2670 / 7001	1080 / 7001	1912 / 8201
RTX 4050 Laptop	10,000	2670 / 8201	585 / 810	1215 / 8201
RTX 4090 pods B	1,000	2820 / 10501	2520 / 10501	2550 / 10501
RTX 4090 pods B	3,000	2820 / 10501	2505 / 10501	2550 / 10501
RTX 4090 pods B	10,000	2820 / 10501	2235 / 810	2520 / 10501
H100 SXM	1,000	1980 / 2619	1605 / 2619	1980 / 2619
H100 SXM	3,000	1980 / 2619	1275 / 2619	1980 / 2619
H100 SXM	10,000	1980 / 2619	645 / 2619	1980 / 2619

with a 90th percentile reaching $3.3\times$. With the heater, its median is 1.00–1.07. Runs more than $1.5\times$ slow fall from 13 of 25 to 0 on the laptop, from 7 to 4 of 25 on the 4090 pods and from 1 to 0 of 9 on the H100; only the laptop’s change is more than a few runs. The heavy kernel has no tail to remove. With the heater it becomes slower on the laptop (1.18 – $1.19\times$ at $E \leq 10,000$, unchanged at 30,000) and the H100 ($1.36\times$), the cost of sharing the GPU, and stays unchanged on the 4090 pods. Paired ratios with block-bootstrap intervals are in Appendix Table 7.

End to end: small and host-dependent. The heater also competes for GPU submission and readback, and on some hosts for the CPU: on the H100 pod it slowed host row materialization by 40–48% at $E \leq 3,000$. End to end, the light path’s median call latency with the heater was 0.94–0.96 of its heater-off value on the 4090 pods at $E = 3,000$ – $10,000$ (1.07 at 30,000) and 0.95–0.97 on the laptop at $E \geq 3,000$, with intervals that mostly include 1. On the H100 it was 1.10–1.24. Heavy calls were 6–9% slower on the consumer GPUs and

25–27% slower on the H100. Under the same added load the light path fared better than the heavy path in 10 of 13 cells, but whether it gained or lost overall depends on how much contention the host adds. The $1.9\times$ end-to-end speed-up in our laptop pilot did not replicate reliably across blocks.

Scorecard against the planned predictions (Appendix Table 6).

- P1 predicted a light-path end-to-end speed-up of at least 10% at $E \leq 3,000$. It *failed* on every GPU.
- P2 predicted no heavy-path speed-up. It held in every cell.
- P3 predicted that the light path’s ratio would fall below the heavy path’s. It held in every cell for device time (on the pods at $E = 30,000$ the two ratios tie at two decimals) and in 10 of 13 cells end to end.
- P4 predicted that light device time falls while host materialization stays within 10%. It held in 6 of 13 cells and

Table 3: Device time relative to the best observed for that GPU and E : median, 90th percentile, and share of runs more than $1.5\times$ slower than best, heater off versus on. One observation per block and E .

Path	GPU	n	Heater off			Heater on		
			median	p90	$> 1.5\times$	median	p90	$> 1.5\times$
Light	RTX 4050 Laptop	25	1.59	2.61	52%	1.07	1.27	0%
Light	RTX 4090 pods	25	1.15	3.29	28%	1.06	1.59	16%
Light	H100 SXM	9	1.24	1.41	11%	1.00	1.01	0%
Heavy	RTX 4050 Laptop	25	1.00	1.00	0%	1.19	1.20	0%
Heavy	RTX 4090 pods	25	1.05	1.07	0%	1.05	1.07	0%
Heavy	H100 SXM	9	1.00	1.01	0%	1.36	1.37	0%

failed where the heater also slowed the host.

The device-level predictions held. P1, the end-to-end prediction and the one that mattered most for an application, did not.

7 Locking the clock

On the laptop we ran the direct test. Each condition ran in a fresh process with the driver’s default clocks (U), with SM and memory locked at boost (SM 2,670 MHz, the heavy path’s observed value; memory 8,001 MHz, which `nvidia-smi` then reports as 8,201; Lhi), or locked at the light path’s low state (585 and 810 MHz; Llo). The heater was crossed with U and Lhi. The design was 6 cells $\times$ 5 conditions $\times$ 5 blocks = 150 processes, with its protocol pushed to the public repository 11 minutes before the first run (Appendix A). No run was excluded, the sampled SM clock matched the lock in every locked run, and the GPU never exceeded 50 °C. Six heater-on, locked-high light runs held memory at 7,001 rather than 8,001 MHz; the protocol’s exclusion rule covered only the SM clock, and their times equal those of the other locked-high runs.

The variation disappears at fixed clocks. Unlocked, the light kernel’s device time varied by $2.84\times$, $2.11\times$ and $1.51\times$ across blocks at $E = 1,000$, 3,000 and 10,000. Locked at boost, it varied by $1.00\times$ in all three cells (prediction K1, ≤ 1.15). The heater had cut unlocked light time to 0.57, 0.37 and 0.40 of its heater-off value; with the clock locked it changed nothing (ratio 1.00 in every cell; K4). The heavy kernel was already stable and unchanged by the lock (Lhi/U = 1.000, 1.000, 1.002; K5).

The low state alone reproduces the slow mode. Locked at 585/810 MHz, the light kernel at $E = 10,000$ took 0.444 ms, within 4% of the 0.461 ms that D1 measured on the same laptop without locking (K3, which was registered against D1). Against E-LOCK’s own unlocked arm (0.567 ms) the low lock is 22% faster, so it reproduces the slow level rather than bounding it. A lock this low may itself put the GPU in an idle-like P-state (Adamek et al., 2021), so Llo is a close match to the driver’s slow mode, not necessarily the same state. The

same lock slowed the heavy kernel by $3.8\times$ (0.95 to 3.6 ms; Table 5, LL, reproduces it), so the slow state is costly for both paths. The driver put the light path there and kept the heavy path out of it.

One prediction failed. K2 predicted that locking high would match the best unlocked speed to within 15%. Locked-high light time was 0.50–0.68 of the best unlocked median (0.63–0.81 of the best single unlocked run), outside that tolerance, so K2 fails. After the fact, we note that the lock beat every unlocked configuration, including unlocked with the heater. That suggests the driver’s best unlocked state for light work is still below boost, but we did not predict it.

The unlocked baseline drifted. E-LOCK’s unlocked light medians (0.069, 0.176 and 0.567 ms) are $1.2\text{--}2.3\times$ slower than D1’s on the same laptop three and a half weeks earlier (0.030, 0.106 and 0.461 ms; Table 1). D1 saw no spread at $E = 1,000$ and no heater effect there; E-LOCK saw $2.84\times$ and 0.57. We did not record what changed between the two dates (driver updates, Windows build, power settings). The unlocked state of this laptop is not stable over weeks, which is itself an argument for locking, and the locked medians are what we compare across experiments.

What this does and does not show. On this laptop, fixing the clocks removes the light kernel’s process-to-process variation and sets its level, so the clock the driver chooses accounts for the variation we measured. We could not repeat the test on the cloud hosts, so the cross-host spread there is attributed to power state by the heater and telemetry evidence only.

Which clock matters. E-LOCK moved the SM and memory clocks together. A follow-up on the same laptop, E-SPLIT, crossed them: SM at 2,670 or 585 MHz, memory at 8,001 or 810 MHz, both paths, the same three E , five blocks, 120 fresh processes. Its protocol and analysis script were committed and pushed before its data. The first attempt used a binary built without the GPU backend and failed at startup; the rebuild was recorded as an amendment before any valid run (Appendix A). Re-running E-LOCK’s two

Table 4: Locked-clock experiment on the laptop: median top- k device time per query (ms) over five blocks, and max/min across blocks. Light = compact rows, heavy = shader predicate; heater off. U = unlocked, Lhi = locked at 2,670/8,001 MHz, Llo = locked at 585/810 MHz.

E	Light						Heavy, Lhi	
	U		Lhi		Llo		ms	max/min
	ms	max/min	ms	max/min	ms	max/min		
1,000	0.069	2.84	0.024	1.00	0.096	1.00	0.948	1.00
3,000	0.176	2.11	0.045	1.00	0.177	1.00	0.951	1.00
10,000	0.567	1.51	0.113	1.00	0.444	1.00	0.941	1.00

corners reproduced its medians within 0.2%. Table 5 reports the medians and, for each kernel, the memory share $m = \ln(t_{HL}/t_{HH})/\ln(t_{LL}/t_{HH})$, where $m = 0$ means lowering only the memory clock costs nothing and $m = 1$ means it accounts for the whole slow mode.

The slow mode is the SM clock. For the selection kernel, which dominates device time and is the metric in every other table, lowering only the memory clock changed nothing at $E \leq 3,000$ and slowed the light path by 10% at $E = 10,000$ ($m \leq 0.07$ everywhere; prediction S2, $m < 0.3$). Lowering only the SM clock reproduced nearly all of the slow mode. The scoring kernel was also SM-driven at $E \leq 3,000$. At $E = 10,000$ the memory clock did matter for scoring: lowering it alone doubled light-path scoring time (0.034 to 0.069 ms; $m = 0.48$, just inside the registered limit of 0.5) and raised heavy-path scoring by 44%. There the two clocks do not act independently ($m + s = 1.35$ for light scoring, where s is the same share for the SM clock). The memory cut we saw in telemetry at $E = 10,000$ therefore costs something in the scoring kernel, but the variation in Table 1 comes mainly from the SM clock. A memory-bound explanation of the slow mode does not hold at these sizes.

8 Consequences

On the laptop, unlocked timing understated the efficient kernel. At fixed boost the light kernel ran $2.9\text{--}5.0\times$ faster than E-LOCK’s unlocked arm, $1.3\text{--}4.1\times$ faster than D1’s unlocked medians and $1.5\text{--}2.0\times$ faster than the best unlocked median, with the heater (K2). How much the unlocked number understates it depends on which unlocked run is the reference, but every reference understates it. A benchmark that compares a work-efficient GPU kernel with a less efficient one at low load runs the efficient one at a lower clock. Here the light path won every cell regardless. But at $E = 10,000$ its device-time advantage over the heavy path is $1.6\times$ on 4090 host A, $2.0\times$ on the laptop, $4.7\times$ on the 4090 pods and $4.8\times$ on the H100 (medians of three to five processes, from Table 1), for identical code. On the laptop with both paths locked at boost it is $8.3\times$ (Table 4). We expect, but did not test, that optimizations which remove GPU work (tighter filtering, smaller kernels, early exit) are measured at a similar discount, and that the discount differs between machines. Our heavy path is a naive comparator; a tuned selection kernel

would load the GPU differently.

Kernel-level results did not transfer between machines with the same GPU. On RTX 4090s (host A and the five pods), light-kernel times for the same cell at $E = 10,000$ ran from 0.128 to 0.589 ms across the five pods’ processes, a $4.6\times$ span (host A’s fall inside it). Any threshold or cost model calibrated on one of them, for example a CPU/GPU router, partly encodes that host’s power behavior. The study that preceded this one fitted a scalar CPU/GPU routing threshold with zero errors on 31 development pairs. On a held-out suite it chose the slower route in 8 of 16 workloads, two of them within 3.2% regret, well inside the process-to-process variation measured here (Appendix C). In our data the heater never changed a CPU/GPU winner (Appendix Table 8). We therefore do not claim that power state moves this system’s crossover, only that it moves the margins a router is calibrated on.

Isolated benchmarks may misstate co-located latency. In deployment a retrieval kernel often shares the GPU with a model; our heater is a crude stand-in for that load. Co-location is known to slow latency-critical GPU work through interference (Han et al., 2022; Strati et al., 2024); the opposite effect, a co-runner raising a shared clock, has been observed for MIG partitions (Patel et al., 2023). On an idle GPU the light path is exposed to deep power states; under load, to contention. The heavy path is exposed only to contention. Latency measured on an idle GPU may therefore be a poor proxy for latency under co-location, and the error can differ between implementations. We did not test co-location with a real model.

What to report. For latency-bound GPU work we would follow the rules below. Rules (i)–(iii) and the first half of (iv) restate existing guidance (Kalibera and Jones, 2013; Hoefer and Belli, 2015; NVIDIA, 2026d,c,b); the load-raising control with a heavy negative control in (iv), and rule (v), are to our search new.

- (i) Replicate across fresh processes and report the spread between them; repetitions inside one process share its power state.
- (ii) Log clocks, performance state and clock-event reasons continuously, not just before and after a run, and record

Table 5: E-SPLIT on the laptop: median device time per query (ms) over blocks, by locked SM / memory clock (H = 2,670 / 8,001 MHz, L = 585 / 810 MHz; first letter SM, second memory), and memory share m . Heater off. Five blocks per cell, except light LH at $E = 1,000$ ($n = 1$) and $E = 3,000$ ($n = 4$), where the driver held memory at 7,001 MHz instead of 8,001 and the runs were excluded as registered.

Path	E	Top- k selection					Scoring				
		HH	HL	LH	LL	m	HH	HL	LH	LL	m
Light	1,000	0.024	0.024	0.096	0.096	0.00	0.007	0.007	0.023	0.023	0.00
Light	3,000	0.044	0.045	0.177	0.177	0.00	0.012	0.012	0.044	0.044	0.00
Light	10,000	0.113	0.124	0.434	0.444	0.07	0.034	0.069	0.123	0.149	0.48
Heavy	1,000	0.947	0.947	3.615	3.615	0.00	0.126	0.126	0.541	0.541	0.00
Heavy	3,000	0.951	0.951	3.624	3.624	0.00	0.140	0.140	0.593	0.593	0.00
Heavy	10,000	0.940	0.965	3.567	3.593	0.02	0.196	0.282	0.763	0.876	0.24

the driver’s power-management mode.

- (iii) Report device-timestamp kernel time alongside end-to-end latency, to separate power-state effects from host effects.
- (iv) Lock clocks where possible, and report results at the locked clock alongside the default. Where clocks cannot be locked, add a load-raising control such as the heater, together with a heavy negative control.
- (v) When comparing implementations, state whether they ran in the same power state.

9 Related work

Power management versus latency. On CPU servers, low-power states and slow frequency governors are known to inflate the latency of latency-critical services at low load (Meisner et al., 2011; Kanev et al., 2014), motivating latency-aware controllers (Lo et al., 2014; Kasture et al., 2015; Hsu et al., 2015; Chou et al., 2019). Mobile governors mispredict concurrent work and run uncoordinated (Pathania et al., 2014; Lin et al., 2023; Zhang et al., 2025), and at low locked clocks NVIDIA GPUs (V100, Titan, P4) can drop into an idle-like P-state (Adamek et al., 2021). GPU DVFS work for machine learning treats frequency as a knob to *lower* for energy (Tang et al., 2019; You et al., 2023; Chung et al., 2024; Qiu et al., 2024; Stojkovic et al., 2025; Kakolyris et al., 2025; Maliakel et al., 2026; Spaan et al., 2026). We study the opposite case, in which the default policy under-clocks latency-bound work that the application would prefer to run fast.

The GPU version of this is known to practitioners. NVIDIA’s documentation warns that floating clocks make timing-based choices between kernel implementations non-deterministic and that a kernel’s measured duration depends on the clock state earlier kernels left behind (NVIDIA, 2026d,b); forum answers describe short, sparse kernels that never raise the clock and recommend locking clocks or keeping the GPU busy with dummy kernels (NVIDIA Developer Forums, 2022). Volkov (2016) ran a compute kernel to raise the clock before measuring, in MIG partitions co-located work forces a light partition onto a higher shared clock (Patel

et al., 2023), and idle gaps between timing trials have corrupted a measured kernel speed-up (Agarwal et al., 2026). Chu and Li (2026) note that a lower-power optimized kernel can run at a higher sustained clock and inflate its apparent speed-up, the same class of confound in the power-capped direction. We have not found a controlled study showing that a work-efficient implementation is systematically disadvantaged because the lighter load it places on the GPU induces a lower power state, nor a measurement of a separate low-utilization process speeding up a foreground kernel’s device time across hosts.

Benchmark validity. Innocuous setup details bias measured speed-ups (Mytkowicz et al., 2009); variation across executions must be sampled and reported (Kalibera and Jones, 2013; Maricq et al., 2018; Hoeffer and Belli, 2015). Same-SKU datacenter GPUs vary by 8% on average (up to 22%) under sustained compute-heavy load, from power management and manufacturing variability (Sinha et al., 2022); GEMM timings vary 1.6–14% across a fleet and 1–6.5% across repeated runs on one GPU, partly from thermally driven frequency changes (Golden et al., 2025). Kernel benchmarking tools therefore lock clocks and detect throttling (NVIDIA, 2026b,c; Oliaro et al., 2026), and `nvidia-smi` power readings cover only part of each update period on some GPUs (Yang et al., 2024). Our spread (up to $4.6\times$) is larger than that same-model variability under sustained load, which measures a different regime; it is comparable to informal reports of $2\text{--}3\times$ idle-state slowdowns of short GPU work (NVIDIA Developer Forums, 2022), and on the pods it confounds host, process and driver.

Filtered vector search and CPU/GPU placement. Filtered nearest-neighbor search is an active area (Gollapudi et al., 2023; Patel et al., 2024; Iff et al., 2026; Shi et al., 2025; Li et al., 2025; Amanbayev et al., 2026), including filtered search on GPUs (Xi et al., 2025); production GPU exact search accepts bitset and bitmap prefilters (NVIDIA, 2026a), and GPU k -selection has well-studied fast designs (Johnson et al., 2021; Zhang et al., 2023) that our single-workgroup selection kernel does not use. Whether to run vector search on a GPU at all depends on data movement and batch size (Gregg

and Hazelwood, 2011; Mageirakos et al., 2026); those studies concern transfer-dominated, large-batch regimes, while ours is a resident corpus at batch 1, where per-dispatch overhead is tens of microseconds on Vulkan (Maczan, 2026). Compile-time CPU/GPU operator placement is fragile when estimates or co-processor resources are wrong (Breß et al., 2016; Karnagel et al., 2017; Rosenfeld et al., 2022). The choice between an eligible-id list and a full-width filter resembles selection vectors versus bitmaps in vectorized CPU databases (Ngom et al., 2021), although our two GPU paths also differ in their selection work. We add that the measured margins between such alternatives can depend on the GPU’s power state, which the alternatives themselves influence.

10 Limitations

One engine, one kernel pair. Both paths come from Qenlo, with a simple hand-written WGSL selection kernel, and the heavy path is deliberately naive. Tuned library kernels such as FAISS or cuVS may load the GPU differently and run in different states; testing them, and more hosts per GPU model, needs compute we did not have. We do not expect these magnitudes to carry over. *Clock locking on one machine.* Locking was possible only on the laptop and was refused in the cloud containers, so the cross-host account rests on telemetry and the heater, which also loads the host. The locked-clock test covers one GPU, one driver and one operating system. *Laptop platform.* The laptop uses hybrid graphics (Optimus): the display runs on the Intel iGPU and the RTX 4050 renders nothing else during the runs. It ran on AC power under the Windows “Balanced” plan; we did not test other plans. *Telemetry.* `nvidia-smi` samples every 50 ms; clocks are reported for active samples of a whole process, including warm-up, so they cannot show the clock during a sub-millisecond kernel. We did not record clock-event reasons or the NVIDIA power-management mode, and we did not test whether “prefer maximum performance” (which needs no administrator rights) reproduces the lock. *Timestamps.* Device times come from pass-level timestamp queries, so after an idle gap they may include wake-up latency as well as slower execution. *Drift.* The laptop’s unlocked behavior changed between D1 and E-LOCK (Section 7). *CPU baseline.* The CPU scan is single-threaded and switches to FP64 above $E = 4,096$; a stronger CPU path could move the crossover. *Small n and confounding.* Five blocks on the laptop, five RTX 4090 pods with one block each on five different hosts, and three H100 blocks. On the pods, host and process variance cannot be separated, and with three to five blocks the bootstrap intervals are block ranges, not 95% intervals. About 65 cell-level verdicts were scored without multiplicity control. The pods also span two driver branches. One pod was replaced before analysis because its host CPU was about $14\times$ slower on a single-threaded test; its partial block is archived but excluded. *Host coupling.* On the laptop and the H100 the heater also changed host-side timings, so end-to-end effects there cannot be attributed to the GPU clock alone. *API.*

All runs use Vulkan through WGPU; CUDA runtimes may be managed differently. *Driver versions.* 616.56 (Windows) and 570.195–580.173 (Linux).

11 Conclusion

In Qenlo, the faster GPU path for a light query was also the one whose kernel time varied most between processes and machines. On the laptop the variation tracked the clock the driver chose, and fixing the clock removed it. On the cloud hosts we could only raise the GPU’s load, which made the light kernel’s slow runs rarer at the device level but did not make calls faster end to end. On the laptop the heavy path held a high clock, so it looked more stable than it would at equal clocks, and its gap to the light path looked smaller ($2.0\times$ unlocked against $8.3\times$ locked at $E = 10,000$). For benchmarks of light GPU work, we would record the power state, replicate across fresh processes, and report device time separately from end-to-end latency. A small engine with this much instrumentation can test whether similar effects appear in other kernels for little money, which is the use we have in mind for it.

Reproducibility

Each experiment’s protocol and predictions were written before its data were collected, by our records, but only the two locked-clock protocols (E-LOCK and E-SPLIT) were pushed to the public repository before their data; GitHub’s push times show the order (Appendix A). The others were committed the same evening, after collection. Every raw archive and the E-LOCK/E-SPLIT trees have SHA-256 manifests, and the runner logs are committed. Raw per-call samples, 50 ms telemetry, analysis scripts and the heater are in the repository: `research/data/raw/2026-09-25-dvfs-*`, `research/scripts/analyze_dvfs_d1.py`, `research/scripts/cross_host.py`, `research/scripts/pool_hosts.py` and `crates/qenlo/examples/gpu_heater.rs`. Tables 1, 2 and 6–8 come from `paper/v2/scripts/make_tables.py` over CSVs that `research/scripts/cross_host.py` and `pool_hosts.py` produce from the extracted archives; Tables 3, 4 and 5 come from `tail_table.py`, `analyze_dvfs_lock.py` and `analyze_dvfs_split.py`. An independent re-implementation by an outside audit reproduced every table cell. Cloud compute for the new runs cost about US\$6.

References

Karel Adamek, Jan Novotny, Jeyarajan Thiyyagalingam, and Wesley Armour. Efficiency near the edge: Increasing the energy efficiency of FFTs on GPUs for real-time edge computing. *IEEE Access*, 9:18167–18182, 2021. ISSN 2169-

3536. doi: 10.1109/access.2021.3053409. URL <http://dx.doi.org/10.1109/ACCESS.2021.3053409>.
- Gaurav Agarwal, Ashish Garg, and Isha Singhal. How much of a real workload can LLM-generated GPU kernels actually reach?, 2026. URL <https://arxiv.org/abs/2609.21058>. Preprint.
- Abylay Amanbayev, Brian Tsan, Tri Dang, and Florin Rusu. Filtered approximate nearest neighbor search in vector databases: System design and performance analysis. *arXiv preprint arXiv:2602.11443*, 2026. URL <https://arxiv.org/abs/2602.11443>.
- Sebastian Breß, Henning Funke, and Jens Teubner. Robust query processing in co-processor-accelerated databases. In *Proceedings of the 2016 International Conference on Management of Data, SIGMOD/PODS’16*, pages 1891–1906. ACM, June 2016. doi: 10.1145/2882903.2882936. URL <http://dx.doi.org/10.1145/2882903.2882936>.
- Chih-Hsun Chou, Laxmi N. Bhuyan, and Daniel Wong. μ DPM: Dynamic power management for the microsecond era. In *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 120–132. IEEE, February 2019. doi: 10.1109/hpca.2019.00032. URL <http://dx.doi.org/10.1109/HPCA.2019.00032>.
- Logan Chu and Dong Li. Hardware-attributed operator profiling for PyTorch, 2026. URL <https://arxiv.org/abs/2609.11938>. Preprint, v2.
- Jae-Won Chung, Yile Gu, Insu Jang, Luoxi Meng, Nikhil Bansal, and Mosharaf Chowdhury. Reducing energy bloat in large model training. In *Proceedings of the ACM SIGOPS 30th Symposium on Operating Systems Principles, SOSP ’24*, pages 144–159. ACM, November 2024. doi: 10.1145/3694715.3695970. URL <http://dx.doi.org/10.1145/3694715.3695970>.
- Alicia Golden, Michael Kuchnik, Samuel Hsia, Zachary DeVito, Gu-Yeon Wei, David Brooks, and Carole-Jean Wu. PRISM: Probabilistic runtime insights and scalable performance modeling for large-scale distributed training, 2025. URL <https://arxiv.org/abs/2510.15596>. Preprint, v3.
- Siddharth Gollapudi, Neel Karia, Varun Sivashankar, Ravishankar Krishnaswamy, Nikit Begwani, Swapnil Raz, Yiyong Lin, Yin Zhang, Neelam Mahapatro, Premkumar Srinivasan, Amit Singh, and Harsha Vardhan Simhadri. Filtered-DiskANN: Graph algorithms for approximate nearest neighbor search with filters. In *Proceedings of the ACM Web Conference 2023*, pages 3406–3416. Association for Computing Machinery, 2023. doi: 10.1145/3543507.3583552. URL <https://doi.org/10.1145/3543507.3583552>.
- Chris Gregg and Kim Hazelwood. Where is the data? why you cannot debate CPU vs. GPU performance without the answer. In *(IEEE ISPASS) IEEE INTERNATIONAL SYMPOSIUM ON PERFORMANCE ANALYSIS OF SYSTEMS AND SOFTWARE*, pages 134–144. IEEE, April 2011. doi: 10.1109/ispass.2011.5762730. URL <http://dx.doi.org/10.1109/ISPASS.2011.5762730>.
- Mingcong Han, Hanze Zhang, Rong Chen, and Haibo Chen. Microsecond-scale preemption for concurrent GPU-accelerated DNN inferences. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, 2022.
- Torsten Hoeftler and Roberto Belli. Scientific benchmarking of parallel computing systems: Twelve ways to tell the masses when reporting performance results. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC ’15)*, 2015. doi: 10.1145/2807591.2807644.
- Chang-Hong Hsu, Yunqi Zhang, Michael A. Laurenzano, David Meisner, Thomas Wenisch, Jason Mars, Lingjia Tang, and Ronald G. Dreslinski. Adrenaline: Pinpointing and reining in tail queries with quick voltage boosting. In *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*, pages 271–282. IEEE, February 2015. doi: 10.1109/hpca.2015.7056039. URL <http://dx.doi.org/10.1109/HPCA.2015.7056039>.
- Patrick Iff, Paul Brügger, Marcin Chrapek, David Kochergin, Maciej Besta, and Torsten Hoeftler. Benchmarking filtered approximate nearest neighbor search algorithms on transformer-based embedding vectors. In *Proceedings of the 49th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 610–622, 2026. doi: 10.1145/3805712.3809731. URL <https://doi.org/10.1145/3805712.3809731>.
- Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data*, 7(3):535–547, 2021. doi: 10.1109/TBDATA.2019.2921572. URL <https://doi.org/10.1109/TBDATA.2019.2921572>.
- Andreas Kosmas Kakolyris, Dimosthenis Masouros, Petros Vavaroutsos, Sotirios Xydis, and Dimitrios Soudris. throttLL’eM: Predictive GPU throttling for energy efficient LLM inference serving. In *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 1363–1378. IEEE, March 2025. doi: 10.1109/hpca61900.2025.00103. URL <http://dx.doi.org/10.1109/HPCA61900.2025.00103>.
- Tomas Kalibera and Richard Jones. Rigorous benchmarking in reasonable time. In *Proceedings of the 2013 international symposium on memory management, ISMM ’13*, pages 63–74. ACM, June 2013. doi: 10.1145/

- 2464157.2464160. URL <http://dx.doi.org/10.1145/2464157.2464160>.
- Svilen Kanev, Kim Hazelwood, Gu-Yeon Wei, and David Brooks. Tradeoffs between power management and tail latency in warehouse-scale applications. In *2014 IEEE International Symposium on Workload Characterization (IISWC)*, pages 31–40. IEEE, October 2014. doi: 10.1109/iiswc.2014.6983037. URL <http://dx.doi.org/10.1109/IISWC.2014.6983037>.
- Tomas Karnagel, Dirk Habich, and Wolfgang Lehner. Adaptive work placement for query processing on heterogeneous computing resources. *Proceedings of the VLDB Endowment*, 10(7):733–744, 2017. doi: 10.14778/3067421.3067423.
- Harshad Kasture, Davide B. Bartolini, Nathan Beckmann, and Daniel Sanchez. Rubik: fast analytical power management for latency-critical systems. In *Proceedings of the 48th International Symposium on Microarchitecture, MICRO-48*, pages 598–610. ACM, December 2015. doi: 10.1145/2830772.2830797. URL <http://dx.doi.org/10.1145/2830772.2830797>.
- Mocheng Li, Xiao Yan, Baotong Lu, Yue Zhang, James Cheng, and Chenhao Ma. Attribute filtering in approximate nearest neighbor search: An in-depth experimental study. *Proceedings of the ACM on Management of Data*, 3(6):298:1–298:26, 2025. doi: 10.1145/3769763. URL <https://doi.org/10.1145/3769763>.
- Chengdong Lin, Kun Wang, Zhenjiang Li, and Yu Pu. A workload-aware DVFS robust to concurrent tasks for mobile devices. In *Proceedings of the 29th Annual International Conference on Mobile Computing and Networking, MobiCom ’23*, pages 1–16. ACM, July 2023. doi: 10.1145/3570361.3592524. URL <http://dx.doi.org/10.1145/3570361.3592524>.
- David Lo, Liqun Cheng, Rama Govindaraju, Luiz Andre Barroso, and Christos Kozyrakis. Towards energy proportionality for large-scale latency-critical workloads. In *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*, pages 301–312. IEEE, June 2014. doi: 10.1109/isca.2014.6853237. URL <http://dx.doi.org/10.1109/ISCA.2014.6853237>.
- Jędrzej Maczan. Characterizing WebGPU dispatch overhead for LLM inference across four GPU vendors, three backends, and three browsers, 2026. URL <https://arxiv.org/abs/2604.02344>. Preprint.
- Vasilis Mageirakos, Joel André, Marko Kabić, Bowen Wu, Yannis Chronis, and Gustavo Alonso. To GPU or not to GPU: Vector search in relational engines, 2026. URL <https://arxiv.org/abs/2605.15957>. Preprint.
- Paul Joe Maliakel, Shashikant Ilager, and Ivona Brandic. Characterizing LLM inference energy-performance trade-offs across workloads and GPU scaling. In *2026 IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, pages 33–43, 2026. doi: 10.1109/CCGrid68966.2026.00013. Preprint: arXiv:2501.08219.
- Aleksander Maricq, Dmitry Duplyakin, Ivo Jimenez, Carlos Maltzahn, Ryan Stutsman, and Robert Ricci. Taming performance variability. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pages 409–425, Carlsbad, CA, October 2018. USENIX Association. ISBN 978-1-939133-08-3. URL <https://www.usenix.org/conference/osdi18/presentation/maricq>.
- David Meisner, Christopher M. Sadler, Luiz André Barroso, Wolf-Dietrich Weber, and Thomas F. Wenisch. Power management of online data-intensive services. In *Proceedings of the 38th annual international symposium on Computer architecture, ISCA ’11*, pages 319–330. ACM, June 2011. doi: 10.1145/2000064.2000103. URL <http://dx.doi.org/10.1145/2000064.2000103>.
- Todd Mytkowicz, Amer Diwan, Matthias Hauswirth, and Peter F. Sweeney. Producing wrong data without doing anything obviously wrong! In *Proceedings of the 14th international conference on Architectural support for programming languages and operating systems, ASPLOS09*, pages 265–276. ACM, March 2009. doi: 10.1145/1508244.1508275. URL <http://dx.doi.org/10.1145/1508244.1508275>.
- Amadou Ngom, Prashanth Menon, Matthew Butrovich, Lin Ma, Wan Shen Lim, Todd C. Mowry, and Andrew Pavlo. Filter representation in vectorized query execution. In *Proceedings of the 17th International Workshop on Data Management on New Hardware (DaMoN ’21)*, pages 1–7, 2021. doi: 10.1145/3465998.3466009. Article 6.
- Beatrix Miranda Ginn Nielsen. Pretrained sentence BERT models AG News embeddings. Dataset, Technical University of Denmark, version 1, CC BY 4.0, 2023. URL <https://doi.org/10.11583/DTU.21286923.v1>.
- NVIDIA. cuVS brute-force index: filtered search with bitset and bitmap prefilters. <https://docs.nvidia.com/cuvs/user-guide/api-guides/indexing-guide/brute-force>, 2026a. Accessed 2026-10-25.
- NVIDIA. Nsight compute profiling guide: Clock control. <https://docs.nvidia.com/nsight-compute/ProfilingGuide/>, 2026b. Accessed 2026-10-25.
- NVIDIA. NVBench: CUDA kernel benchmarking library (command-line options: clock locking, throttle detection). <https://github.com/NVIDIA/nvbench>, 2026c. Accessed 2026-10-25.

- NVIDIA. TensorRT documentation: Hardware/software environment for performance measurements. <https://docs.nvidia.com/deeplearning/tensorrt/10.x.x/performance/hw-sw-environment.html>, 2026d. Accessed 2026-10-25.
- NVIDIA Developer Forums. Why kernel calculate speed got slower after waiting for a while? (thread 221059, 2022); avoiding performance mode slowdown (thread 54181, 2017). <https://forums.developer.nvidia.com/t/221059>, 2022. Practitioner reports; not peer reviewed.
- Gabriele Oliaro, Yichao Fu, May Jiang, Owen Lu, Junli Wang, Zhihao Jia, Hao Zhang, and Samyam Rajbhandari. FastKernels: Benchmarking GPU kernel generation in production, 2026. URL <https://arxiv.org/abs/2605.23215>. Preprint, v1.
- Liana Patel, Peter Kraft, Carlos Guestrin, and Matei Zaharia. ACORN: Performant and predicate-agnostic search over vector embeddings and structured data. *Proceedings of the ACM on Management of Data*, 2(3):120:1–120:27, 2024. doi: 10.1145/3654923. URL <https://doi.org/10.1145/3654923>.
- Pratyush Patel, Zibo Gong, Syeda Rizvi, Esha Choukse, Pulkit Misra, Thomas Anderson, and Akshitha Sriraman. Towards improved power management in cloud GPUs. *IEEE Computer Architecture Letters*, 22(2):141–144, 2023. doi: 10.1109/LCA.2023.3278652.
- Anuj Pathania, Qing Jiao, Alok Prakash, and Tulika Mitra. Integrated CPU-GPU power management for 3d mobile games. In *Proceedings of the 51st Annual Design Automation Conference, DAC '14*, pages 1–6. ACM, June 2014. doi: 10.1145/2593069.2593151. URL <http://dx.doi.org/10.1145/2593069.2593151>.
- Haoran Qiu, Weichao Mao, Archit Patke, Shengkun Cui, Saurabh Jha, Chen Wang, Hubertus Franke, Zbigniew Kalbarczyk, Tamer Başar, and Ravishankar K. Iyer. Power-aware deep learning model serving with μ -Serve. In *2024 USENIX Annual Technical Conference (USENIX ATC 24)*, pages 75–93, Santa Clara, CA, July 2024. USENIX Association. ISBN 978-1-939133-41-0. URL <https://www.usenix.org/conference/atc24/presentation/qiu>.
- Viktor Rosenfeld, Sebastian Breß, and Volker Markl. Query processing on heterogeneous CPU/GPU systems. *ACM Computing Surveys*, 55(1):11:1–11:38, 2022. doi: 10.1145/3485126.
- Jiayang Shi, Yuzheng Cai, and Weiguo Zheng. Filtered approximate nearest neighbor search: A unified benchmark and systematic experimental study [experiment, analysis & benchmark], 2025. URL <https://arxiv.org/abs/2509.07789>. Preprint.
- Prasoon Sinha, Akhil Guliani, Rutwik Jain, Brandon Tran, Matthew D. Sinclair, and Shivaram Venkataraman. Not all GPUs are created equal: Characterizing variability in large-scale, accelerator-rich systems. In *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–15, 2022. doi: 10.1109/SC41404.2022.00070. arXiv:2208.11035.
- Jeffrey Spaan, Kuan-Hsun Chen, and Ana-Lucia Varbanescu. Reducing compute waste in LLMs through kernel-level DVFS, 2026. URL <https://arxiv.org/abs/2601.08539>. Preprint.
- Jovan Stojkovic, Chaojie Zhang, Íñigo Goiri, Josep Torrellas, and Esha Choukse. DynamoLLM: Designing LLM inference clusters for performance and energy efficiency. In *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 1348–1362. IEEE, March 2025. doi: 10.1109/hpca61900.2025.00102. URL <http://dx.doi.org/10.1109/HPCA61900.2025.00102>.
- Foteini Strati, Xianzhe Ma, and Ana Klimovic. Orion: Interference-aware, fine-grained GPU sharing for ML applications. In *Proceedings of the Nineteenth European Conference on Computer Systems (EuroSys '24)*, pages 1075–1092, 2024. doi: 10.1145/3627703.3629578.
- Zhenheng Tang, Yuxin Wang, Qiang Wang, and Xiaowen Chu. The impact of GPU DVFS on the energy and performance of deep learning: an empirical study. In *Proceedings of the Tenth ACM International Conference on Future Energy Systems, e-Energy '19*, pages 315–325. ACM, June 2019. doi: 10.1145/3307772.3328315. URL <http://dx.doi.org/10.1145/3307772.3328315>.
- Vasily Volkov. Understanding latency hiding on GPUs. Technical Report UCB/EECS-2016-143, EECS Department, University of California, Berkeley, 2016.
- Jingyi Xi, Chenghao Mo, Ben Karsin, Artem Chirkin, Mingqin Li, and Minjia Zhang. VecFlow: A high-performance vector data management system for filtered-search on GPUs. *Proceedings of the ACM on Management of Data*, 3(4):271:1–271:27, 2025. doi: 10.1145/3749189.
- Zeyu Yang, Karel Adamek, and Wesley Armour. Accurate and convenient energy measurements for GPUs: A detailed study of NVIDIA GPU’s built-in power sensor. In *SC24: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–17, 2024. doi: 10.1109/SC41406.2024.00028. Preprint: arXiv:2312.02741, “Part-time Power Measurements: nvidia-smi’s Lack of Attention”.
- Jie You, Jae-Won Chung, and Mosharaf Chowdhury. Zeus: Understanding and optimizing GPU energy consumption of DNN training. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 119–139, Boston, MA,

April 2023. USENIX Association. ISBN 978-1-939133-33-5. URL <https://www.usenix.org/conference/nsdi23/presentation/you>.

Jingrong Zhang, Akira Naruse, Xipeng Li, and Yong Wang. Parallel top-K algorithms on GPU: A comprehensive study and new methods. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '23)*, 2023. doi: 10.1145/3581784.3607062.

Zongpu Zhang, Pranab Dash, Y. Charlie Hu, Qiang Xu, Jian Li, and Haibing Guan. Dissecting the impact of mobile DVFS governors on LLM inference performance and energy efficiency, 2025. URL <https://arxiv.org/abs/2507.02135>. Preprint.

A Protocols, predictions and deviations

Three protocols were written, according to our records, before their data existed, but they were committed to version control and hashed only after the data for D1, D1R and the H100 arm had been collected (same evening, 2026-09-25), so no independent timestamp shows the order. We therefore call their predictions planned, not pre-registered. Their timing rests on the protocol files’ own statements and the run logs. The protocols are: the laptop experiment D1 (`research/experiments/dvfs-d1/protocol.md`), the RTX 4090 pod replication D1R (`dvfs-d1r/`) and the H100 arm (`dvfs-h100/`). A protocol for an API and batch-size follow-up (D2, `dvfs-d2/`) was written but not run and is not part of this paper. (An early commit message calls these three protocols pre-registered; they were not.) The locked-clock experiment E-LOCK (`research/experiments/dvfs-lock/`) ran on the laptop only. GitHub’s server-side push record, which an author cannot backdate, shows its protocol pushed at 05:09:15 UTC on 2026-10-19; our runner logged the first run starting at 05:20:13. Its analysis script was written after the data. E-SPLIT (`research/experiments/dvfs-split/`), which crosses the SM and memory locks, had its protocol pushed at 07:25:49 UTC on 2026-10-25, its analysis script at 07:33:29 and its amendment at 07:37:55; the first valid run started at 07:38:32. Its predictions were written after seeing E-LOCK’s medians, as the protocol states. Run start times come from our own runner and are not externally timestamped. Neither protocol was filed with an independent registry such as OSF, so we call their predictions committed in advance rather than pre-registered in the registry sense. One exploratory pilot preceded them (`research/data/raw/2026-09-25-dvfs-pilot/`) and is used only to motivate the design. D1 predicted, for batch-1 queries: P1, the heater lowers light-path end-to-end P50 by at least 10% (ratio < 0.9 , interval excluding 1) at $E \leq 3,000$; P2, the heater does not speed up the heavy path (ratio ≥ 0.95); P3,

the light path’s ratio is below the heavy path’s; P4, the heater lowers light-path device time while host row materialization stays within $\pm 10\%$. The same rules were applied unchanged to every host.

P1 fails everywhere at the block level; the pilot’s $1.9\times$ end-to-end speed-up on the laptop did not replicate reliably across blocks. P2 and the device-time half of P3 hold in every cell. P4 fails where the heater also slowed host-side work. On the H100 pod, host row materialization became 40–48% slower with the heater running, which shows that the heater process took host CPU time as well as GPU time. The end-to-end slowdowns on that host therefore say little about the power-state account; they show that our heater is not a clean GPU-only perturbation.

Deviations. (i) The clock summary in the planned analysis script took the median over the whole process, including data loading; we added a GPU-active median (utilization > 0) before inspecting any cross-host comparison and report both in the released CSVs. (ii) Two pod setup attempts failed because of script bugs (a missing dataset copy, a mis-specified pod) and were re-run before any cell completed; no completed cell was discarded. (iii) The pod first assigned to block 3 ran host-side benchmark work roughly $10\times$ slower than the others (about $14\times$ on a single-threaded Python test). We re-ran block 3 from scratch on a new pod (Ryzen 9 7950X); by our records the decision was made on host speed before inspecting its results, but 14 of its cells had already completed. They are retained (`d1r-b3-partial.tar.gz`) and excluded from the tables. Keeping them would have widened the pods’ spread, not narrowed it: the slow pod’s light kernel took 1.084 ms at $E = 10,000$, against 0.128–0.589 ms on the other pods. A stray setup process ran for about 30 s on the block 4 pod during three cells; dropping those cells changes the affected light-path ratio from 0.967 to 0.966. (iv) The cross-host table (Table 1) and the device-time framing were not planned; they were suggested by the retained September 24 archive and are therefore partly exploratory, although the pods and H100 supplied new data for them.

E-LOCK predictions. K1 (light max/min ≤ 1.15 locked high), K3 (locked-low light time within 25% of D1’s slow mode at $E = 10,000$), K4 (no heater effect locked high) and K5 (heavy within 3% of unlocked) held in every tested cell. K2 (locked-high light time within 15% of the best unlocked) failed: locked-high light time was 32–50% lower than the best unlocked median. No run was excluded and the run followed the protocol. Six heater-on, locked-high light runs held memory at 7,001 rather than the requested 8,001 MHz although the lock command succeeded; the exclusion rule covered only the SM clock, and their times match the other locked-high runs, so K4 is unaffected. Analysis: `research/scripts/analyze_dvfs_lock.py`.

E-SPLIT predictions and deviations. S0 (re-run E-LOCK corners within 10% of E-LOCK), S1 (light max/min ≤ 1.15

Table 6: Planned predictions by GPU and E . P3 is shown for end-to-end P50 / device time. “n/a”: P1 was registered only for $E \leq 3,000$.

GPU	E	P1 call < 0.9	P2 heavy ≥ 0.95	P3 light < heavy	P4 device ↓, host flat
RTX 4050 Laptop	100	no	yes	no / yes	no
RTX 4050 Laptop	1,000	no	yes	no / yes	no
RTX 4050 Laptop	3,000	no	yes	yes / yes	no
RTX 4050 Laptop	10,000	n/a	yes	yes / yes	no
RTX 4050 Laptop	30,000	n/a	yes	yes / yes	yes
RTX 4090 pods B	100	no	yes	yes / yes	no
RTX 4090 pods B	1,000	no	yes	yes / yes	yes
RTX 4090 pods B	3,000	no	yes	yes / yes	yes
RTX 4090 pods B	10,000	n/a	yes	yes / yes	yes
RTX 4090 pods B	30,000	n/a	yes	no / yes	yes
H100 SXM	1,000	no	yes	yes / yes	no
H100 SXM	3,000	no	yes	yes / yes	no
H100 SXM	10,000	n/a	yes	yes / yes	yes

Table 7: Paired heater-on / heater-off ratios (median over blocks of per-block ratios, percentile block-bootstrap interval, which with $n \leq 5$ is the range of the block ratios and covers the median with probability only about 75% ($n = 3$) to 94% ($n = 5$), and with the H100’s $n = 3$ there are only ten distinct resamples). Below 1 = faster with the heater. Device = top- k kernel time; call = end-to-end P50. n = blocks.

GPU	E	n	Light device	Heavy device	Light call	Heavy call
RTX 4050 Laptop	100	5	0.91 [0.90, 1.12]	1.19 [1.00, 1.20]	1.11 [0.78, 1.80]	1.09 [1.02, 1.12]
RTX 4050 Laptop	1,000	5	1.00 [0.98, 1.01]	1.18 [1.18, 1.19]	1.13 [1.05, 1.64]	1.06 [1.04, 1.09]
RTX 4050 Laptop	3,000	5	0.65 [0.26, 1.25]	1.19 [1.18, 1.19]	0.96 [0.48, 1.56]	1.07 [1.02, 1.09]
RTX 4050 Laptop	10,000	5	0.46 [0.39, 0.54]	1.19 [1.19, 1.20]	0.97 [0.79, 1.10]	1.08 [1.06, 1.17]
RTX 4050 Laptop	30,000	5	0.59 [0.57, 0.68]	1.00 [1.00, 1.00]	0.95 [0.92, 1.01]	1.07 [1.02, 1.11]
RTX 4090 pods B	100	5	0.92 [0.91, 1.01]	1.00 [1.00, 1.00]	0.97 [0.95, 1.11]	1.07 [0.95, 1.08]
RTX 4090 pods B	1,000	5	0.90 [0.70, 0.92]	1.00 [1.00, 1.00]	0.97 [0.94, 1.03]	1.07 [1.06, 1.28]
RTX 4090 pods B	3,000	5	0.87 [0.35, 0.94]	1.00 [1.00, 1.00]	0.96 [0.54, 0.99]	1.07 [1.02, 1.07]
RTX 4090 pods B	10,000	5	0.82 [0.34, 0.91]	1.00 [1.00, 1.01]	0.94 [0.64, 1.00]	1.07 [1.07, 1.37]
RTX 4090 pods B	30,000	5	1.00 [0.46, 1.00]	1.00 [1.00, 1.00]	1.07 [0.57, 1.09]	1.07 [1.03, 1.16]
H100 SXM	1,000	3	0.86 [0.86, 0.88]	1.36 [1.35, 1.36]	1.24 [1.21, 1.25]	1.26 [1.26, 1.28]
H100 SXM	3,000	3	0.71 [0.65, 0.77]	1.35 [1.35, 1.35]	1.10 [1.03, 1.17]	1.27 [1.26, 1.28]
H100 SXM	10,000	3	0.81 [0.80, 0.88]	1.36 [1.36, 1.37]	1.18 [1.14, 1.48]	1.25 [1.19, 1.26]

under every lock), S2 (selection memory share $m < 0.3$) and S3 (scoring $m < 0.5$) held in every testable cell; S3’s margin was small for light scoring at $E = 10,000$ ($m = 0.48$). One amendment was made before any valid data: the registered binary had been built without the `gpu-wgpu` feature, so the first attempt failed every process at startup with no measurements. It was rebuilt from the same commit with the feature, the new hash was committed as `amendment-1.md`, and the failed attempt is archived unanalyzed. Five light-path runs locked at SM 585 MHz and memory 8,001 MHz ran with memory at 7,001 MHz and were excluded under the registered 5% rule, leaving $n = 1$ at $E = 1,000$ and $n = 4$ at $E = 3,000$ for that condition. Analysis: `research/scripts/analyze_dvfs_split.py`.

B CPU/GPU crossover

Table 8 gives, per GPU, E and arm, the median over blocks of each block’s end-to-end P50. The CPU’s cost per eligible row is 83–103 ns on the laptop and 44–80 ns on the pods at $E \leq 3,000$, then 306–333 ns and 149–164 ns at $E \geq 10,000$. The jump coincides with the CPU scan’s switch from certified FP32 to FP64 accumulation above $E = 4,096$ (Section 2),

so the crossover between $E = 3,000$ and 10,000 is partly a property of our CPU kernel. Extrapolating the FP32 rate would put the laptop’s CPU ahead of the light path at $E = 10,000$ and the pods’ at parity; we did not measure this. The H100 and host A had no CPU arm.

C The held-out router failure that motivated this study

The system’s earlier research campaign (preserved unchanged in `paper/paper.tex`) selected a scalar routing rule, GPU if $E \cdot D \cdot B \geq 10^6$, with zero errors on 31 development CPU/GPU pairs. On a held-out suite of 16 workloads on an RTX 4050 laptop it chose the slower route in 8, with median regret 0.4% and maximum regret 235.7%. Each route was measured in a single process with five in-process runs. An audit of that archive found that the automatic arm differed from the forced arm of the same backend by more than 25% in 9 of 16 cells, and that two of the eight “wrong” labels were within 3.2% regret. In light of Section 4, labels near parity in such a suite partly measure the GPU’s power state during each process, and the per-workload count is not a stable

Table 8: End-to-end P50 (ms), median over blocks per column, for the single-threaded CPU scan, the light GPU path with the heater off and on (winner against CPU in parentheses), and the heavy GPU path. The heater never changed a winner.

GPU	E	CPU	Light, heater off	Light, heater on	Heavy
RTX 4050 Laptop	100	0.010	0.172 (CPU)	0.253 (CPU)	1.999
RTX 4050 Laptop	1,000	0.083	0.230 (CPU)	0.374 (CPU)	2.050
RTX 4050 Laptop	3,000	0.308	0.548 (CPU)	0.559 (CPU)	2.109
RTX 4050 Laptop	10,000	3.332	1.682 (GPU)	1.685 (GPU)	2.204
RTX 4050 Laptop	30,000	9.187	2.265 (GPU)	2.138 (GPU)	2.791
RTX 4090 pods B	100	0.008	0.103 (CPU)	0.100 (CPU)	0.828
RTX 4090 pods B	1,000	0.048	0.123 (CPU)	0.123 (CPU)	0.835
RTX 4090 pods B	3,000	0.133	0.178 (CPU)	0.175 (CPU)	0.849
RTX 4090 pods B	10,000	1.495	0.430 (GPU)	0.405 (GPU)	0.860
RTX 4090 pods B	30,000	4.907	0.779 (GPU)	0.833 (GPU)	0.894

statistic. Whether the 235.7% maximum regret ($3.36\times$) reflects the rule or the GPU’s state cannot be settled from that archive: the suite ran on the same laptop whose light kernel later varied up to $4.2\times$ between processes, and kernel-time spread is not the same quantity as end-to-end route regret. The earlier manuscript also documents representation, preparation and harness findings that are not repeated here.

D Cost and resources

New cloud runs used five RTX 4090 pods (one block each), one H100 SXM pod and a 4-minute H100 probe, at \$0.74 and \$3.49 per hour. The operator’s ledger totals about US\$6 (research/experiments/dvfs-dlr/campaign-log.md). All pods were deleted after their data were downloaded and hash-checked. A container-level Vulkan fix (pointing the loader at `libEGL_nvidia.so.0` rather than the toolkit’s GLX ICD) was required on every pod and is documented in the log; it may explain an earlier “no Vulkan” failure on an A6000.