When Peak FLOPs Mislead: Utilization and Cost Frontiers for Small-Language-Model Pretraining

Tavish Mankash tavish@mankash.com
Vardhaman Kalloli vardhamankalloli722@gmail.com
Keshava Prasad keshava.prasad07@gmail.com
Deepan Muthirayan deepan.muthirayan@plaksha.edu.in

FAIRC · Plaksha University

Abstract

Peak tensor throughput and hourly rental price are weak proxies for the economics of small-language-model (SLM) pretraining. We benchmark complete eager-PyTorch training steps for nine dense models from 150M to 8B parameters on ten NVIDIA GPUs, sweeping power-of-two batch sizes and seven context lengths. Each batch uses adaptive warmup, 20 measured steps, and two input-seed timing replicates, recording tokens/second (TPS), modeled TFLOP/s, model FLOP utilization (MFU), and peak allocated VRAM. The resulting SLMTrainBench dataset contains 2,963 configurations and 430 replicated throughput optima. At context 2,048, a dated RunPod Secure snapshot places the RTX 5090 or 4090 on the minimum-cost frontier through 1B parameters, the A6000 at 2B, and the B200 at 4B and 8B. At 2B, the B200 is 9.30× faster than the minimum-cost A6000 while costing only 19.5% more per steady-state billion tokens. Longer contexts raise measured MFU while reducing TPS, so MFU should not be optimized alone. The two replicates select the same best batch in 98.8% of workload anchors, with a 0.386% 95th-percentile TPS difference. In whole-GPU holdout, one target-GPU reference sweep reduces median TPS error from 13.13% to 5.30% and recovers the measured time–cost Pareto frontier at 87.9% F1. Finally, we separate measured throughput from a dated price layer spanning RunPod, Vast.ai, Lambda, AWS, and Google Cloud; non-RunPod values are price-transfer projections, not provider measurements.

Artifact: `github.com/openlanguagemodel/openlanguagemodel` (SLMTrainBench)

1 Introduction

Which GPU is cheapest for pretraining a small language model? Peak BF16 throughput suggests a simple answer: buy or rent the most recent accelerator, then divide its advertised FLOP rate by its hourly price. That calculation assumes the workload reaches a comparable fraction of peak on every device. Small transformers violate the assumption. Their matrices, launch sequence, and feasible batches can leave a high-throughput accelerator underfilled, while a lower-peak device executes the same eager training program at a much larger fraction of its nominal capacity.

Model FLOP utilization (MFU) makes this gap visible, but MFU is not itself the training objective. A practitioner usually cares about tokens processed per second, whether the workload fits, elapsed training time, and dollars per token. The cheapest device can be slower; the fastest device can be poor value; and a longer context can increase MFU even as it decreases token throughput. A useful hardware comparison must expose all of these quantities at the batch size the device can actually sustain.

We study this regime with OpenLanguageModel (OLM), a readable PyTorch pretraining library [9]. The benchmark is an ordinary, uncompiled eager run: BF16 autocast, PyTorch scaled dot-product attention, a shifted-token loss, backward propagation, and fused AdamW. It is intentionally representative of a straightforward SLM experiment rather than a kernel-specialized record attempt.

This paper contributes:

1. SLMTrainBench, a two-input-replicate, batch-saturated measurement map over 10 GPUs, 9 model scales, and 7 context lengths, retaining TPS, modeled TFLOP/s, nominal and configured-clock MFU, VRAM, OOM boundaries, per-step timing, and GPU telemetry;

2. scale-dependent speed and cost frontiers that show where inexpensive workstation and consumer GPUs dominate price, and where capacity-rich datacenter accelerators regain the frontier; and

3. a shared capacity and throughput predictor that uses hardware specifications plus one target-GPU saturation sweep, with whole-GPU, model-scale, batch-choice, OOM, and Pareto-recovery validation; and

4. a reproducible separation between measured performance and dated cloud prices, including a normalized snapshot across six provider price surfaces.

The result is not a timeless ranking of GPU brands. It is a workload-specific map, with the benchmark stack, price date, and feasibility boundary made explicit.

2 Related Work

Large-scale language-model systems report achieved FLOPs and MFU to distinguish usable throughput from device peak. PaLM popularized MFU as a model-level training-efficiency measure [2]; Megatron-LM and its distributed extensions study utilization and scaling at much larger model and cluster sizes [11, 19]. MLPerf Training provides controlled time-to-quality comparisons for standardized large training workloads [10]. Our question is different: given one common SLM implementation and one GPU, how do batch, context, capacity, and rental price alter the frontier?

Kernel and hardware studies explain why peak specifications need not transfer to small workloads. FlashAttention shows that memory traffic and tiling matter alongside operation count [3]. NVIDIA's matrix multiplication guide describes arithmetic-intensity, tile, and wave-quantization effects that reduce realized throughput for small or poorly aligned GEMMs [12]. Recent work on Overall FLOP Utilization (OFU) further distinguishes hardware-counter utilization from application FLOP accounting [14]. We report MFU because it is reproducible from model shape and elapsed time; we do not treat it as a complete hardware-counter measurement.

Performance prediction for DNN training spans analytical, trace-driven, and reference-device approaches. Paleo maps declarative network requirements to hardware and parallelization choices [16]; Daydream transforms a fine-grained trace graph to estimate the effect of training optimizations [23]. Habitat scales one measured iteration across GPU architectures and reports 11.8% average error over six architectures [22]. Centimani is closest to our joint capacity, batch, and accelerator-selection objective: it combines memory and decoupled performance models and reports 93.1% average single-device prediction accuracy across six DNNs and four accelerator types [21]. Lumos instead targets trace-driven replay and configuration exploration for distributed GPT-3 training, reporting 3.3% average replay error at up to 512 H100s [8].

Our predictor is deliberately lighter than these operator- or trace-level systems. It targets one contemporary eager-PyTorch SLM family across ten single-GPU designs, uses published specifications plus at most one batch-saturation sweep, and couples throughput to an explicit OOM and mutable

price frontier. Because the workloads, target metrics, and validation splits differ, published error percentages above are context rather than direct head-to-head baselines. We instead compare peak BF16, specifications-only, and one-sweep variants under the same whole-GPU holdout, reporting TPS error, fastest- and cheapest-GPU decisions, and time–cost Pareto recovery.

Resource-constrained language modeling has motivated single-GPU benchmarks and auditable small-model training [5, 7]. Rao [17] studies GPT-2 medium and large under several distributed strategies on homogeneous and heterogeneous FABRIC clusters. The present study complements that work with a dense single-GPU batch/context sweep across ten GPU SKUs, explicit OOM frontiers, MFU, and a detachable cloud price layer.

3 Experimental Design

3.1 Models and training step

We instantiate nine tied-embedding, decoder-only OLM models with a 32,000-token vocabulary and 150,436,608 to 7,983,108,096 unique parameters. The architecture uses RMSNorm, rotary position embeddings, SwiGLU, grouped-query attention, and PyTorch scaled dot-product attention. Widths, depths, and head counts are listed in Appendix A. All publication rows use PyTorch 2.8.0 with CUDA 12.8 and Python 3.12.3 [13].

One timed step includes optimizer zeroing, BF16-autocast forward propagation, shifted-token cross-entropy, backward propagation, and a fused AdamW update. Parameters, gradients, and Adam states remain FP32. Gradient accumulation is one, and activation checkpointing and `torch.compile` are disabled. Synthetic integer tokens remain resident on the GPU so the experiment isolates the model training step rather than storage or input-pipeline performance.

3.2 Batch and context sweeps

At context 2,048, each model/device pair begins at batch one and sweeps power-of-two batches through 64. The sweep stops at OOM or after two consecutive doublings fail to improve TPS by at least 1%. The context extension evaluates 512, 1,024, 4,096, and 8,192 tokens around the token-equivalent batch implied by the 2,048-token optimum, including adjacent powers of two. Surviving workloads are extended to 16,384 and 32,768 tokens. Thus a displayed point is the best observed shared batch at that model, GPU, and context; it is not a fixed global batch.

Input seeds 11 and 22 produce separate model initializations and synthetic-token streams. We use them as timing replicates, not as samples of training quality or physical hardware. For each GPU/model/context anchor, a batch is eligible only when both replicates completed and stabilized. We take median TPS across the two replicates at each eligible batch, select the batch with maximum median TPS, and report medians of TPS, MFU, and VRAM at that shared batch. A larger batch breaks an exact TPS tie.

3.3 Adaptive warmup and timing

CUDA events time every training step. Warmup compares the most recent two five-step windows. A check passes when their medians differ by at most 2% and the current window's median-absolute-deviation jitter is at most 3%. Timing starts after two consecutive checks pass, or after a 50-step cap. We then record 20 steps and use their median time. Of the 2,105 complete stable rows, 95.8% stabilized at the first eligible check (step 11); the maximum was 45 steps.

3.4 TPS, MFU, and memory

For microbatch b, sequence length s, and median step time t,

$$\mathrm{TPS} = \frac{bs}{t}.$$

For P unique parameters, n_ℓ layers, and hidden width h, we use the standard dense-transformer training approximation

$$f_{\text{token}} = 6P + 12n_\ell hs, \qquad A = \frac{\text{TPS}\, f_{\text{token}}}{10^{12}}, \qquad \text{MFU}_{\text{nom}} = \frac{A}{F_{\text{BF16,nom}}}.$$

Here A is achieved modeled TFLOP/s and the second term in f_{token} accounts for sequence-dependent attention FLOPs; elementwise operations are omitted. Main-text MFU uses the conventional vendor nominal dense BF16 peak. We also retain a configured-clock sensitivity value, $\text{MFU}_{\text{cfg}} = A/F_{\text{BF16,cfg}}$, where the nominal peak is linearly rescaled only when the captured maximum application clock differs from the vendor reference clock. Both denominators and their source derivations are released. Peak allocated and reserved VRAM come from PyTorch's CUDA allocator; allocated VRAM is reported in the main results.

3.5 Price layer

Prices were captured on 27 July 2026 from RunPod Secure and Community, the Vast.ai marketplace, Lambda on-demand instances, AWS Capacity Blocks, and Google Cloud accelerator-optimized VMs [1, 4, 6, 18, 20]. For Vast.ai, the quote is the median of the three cheapest then-rentable, verified, single-GPU offers with reliability at least 0.98, CUDA at least 12.8, and matching VRAM; one SKU had only one eligible listing. Other providers use their advertised rate for the closest measured SKU. Exact and near-SKU matches are retained separately.

For provider p, GPU g, and measured workload w, the normalized steady-state cost is

$$C_{p,g,w} = \frac{r_{p,g}\, 10^9}{3600\, \text{TPS}_{g,w}},$$

where $r_{p,g}$ is USD per GPU-hour. Only RunPod throughput was measured. Every other provider therefore changes the price numerator while retaining the measured RunPod denominator. Multi-GPU-only products are divided by their GPU count for an amortized per-GPU rate; this assumes every device can be used at the single-GPU rate and is not a claim of measured distributed efficiency. Setup, idle time, storage, data loading, checkpointing, evaluation, failures, egress, tax, commitments, and availability are excluded.

4 Results

4.1 Peak FLOPs do not determine the SLM frontier

Figure 1 compares nominal-reference MFU and RunPod Secure compute cost at context 2,048. The utilization ordering is not the specification ordering. For the 150M model, the H100 reaches 139.7k TPS, 178.8 modeled TFLOP/s, and 18.1% nominal MFU; the RTX 5090 reaches 87.0k TPS, 111.4 TFLOP/s, and 53.2% nominal MFU. At 500M, the same devices reach 63.1k TPS and 24.8% MFU versus 33.2k TPS and 61.4% MFU. The H100 is faster in both cases, but not in proportion to its dense BF16 peak or hourly price.

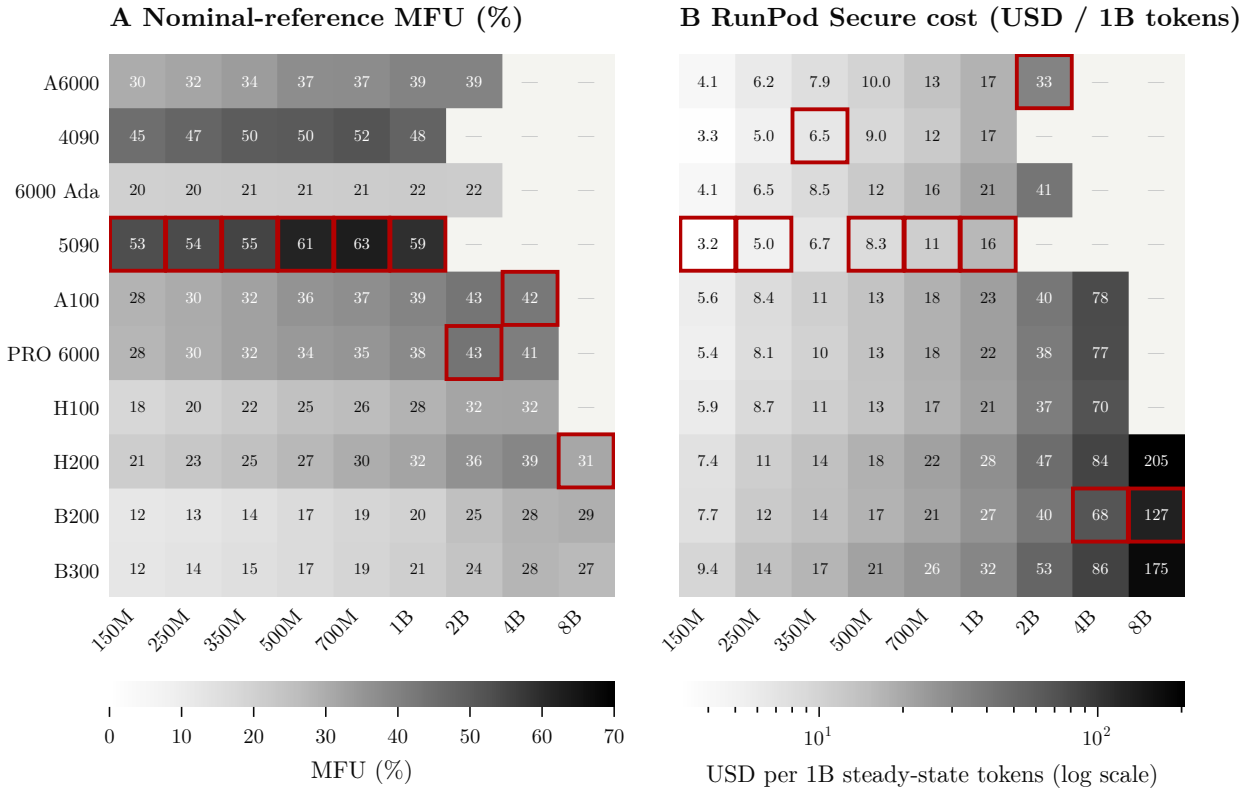

Figure 1. Utilization and cost at context 2,048. Each cell is the median of two input-seed timing replicates at their best shared batch. **A:** nominal-reference MFU. **B:** projected RunPod Secure USD per billion steady-state tokens, on a log color scale. Red outlines mark the maximum MFU or minimum cost in each model column; numbers, not color, carry the values. A dash means no stable dual-replicate training point fit. Cost excludes all non-compute overheads listed in Section 3.5.

The minimum-cost device consequently changes with model scale (Table 1). RTX 5090 or 4090 leads from 150M through 1B, A6000 leads at 2B, and B200 leads at 4B and 8B. Cost alone is insufficient: the 2B A6000 point costs $33.34 per billion steady-state tokens but requires 62.9 hours; the B200 costs $39.84 and requires 6.76 hours. The latter is 9.30× faster for 19.5% more projected compute cost. These two points are both Pareto-relevant under different values of wall time.

Scale	Minimum-cost GPU	Batch	kTPS	TF/s	Nominal MFU (%)	$/1B	Fastest GPU	Speedup
150M	5090	8	87.0	111.4	53.2	3.16	B300	2.50×
250M	5090	4	54.8	112.5	53.7	5.01	B300	2.71×
350M	4090	4	29.3	82.0	49.6	6.54	B300	4.14×
500M	5090	4	33.2	128.7	61.4	8.29	B300	2.90×
700M	5090	4	25.1	132.3	63.1	10.94	B300	3.17×
1B	5090	2	17.2	123.9	59.1	15.99	B300	3.73×
2B	A6000	2	4.4	61.0	39.4	33.34	B200	9.30×
4B	B200	8	24.0	637.3	28.3	68.16	B200	1.00×
8B	B200	4	12.9	662.3	29.4	127.29	B200	1.00×

Table 1. RunPod Secure price snapshot at context 2,048. Each row uses the dual-input-replicate, shared-batch TPS optimum. TF/s is achieved modeled training throughput. Cost is projected steady-state compute cost and excludes setup, storage, data, checkpointing, evaluation, and tax.

4.2 Context length can raise MFU while lowering TPS

The 500M sweep isolates the context effect (Figure 2). On the H100, increasing context from 512 to 32,768 raises MFU from 22.5% to 41.0%, yet lowers TPS from 68.5k to 24.4k. The B200 moves from 15.6% to 21.7% MFU while falling from 108.2k to 29.3k TPS. The RTX 5090 shows the same direction before its 8,192-token capacity boundary.

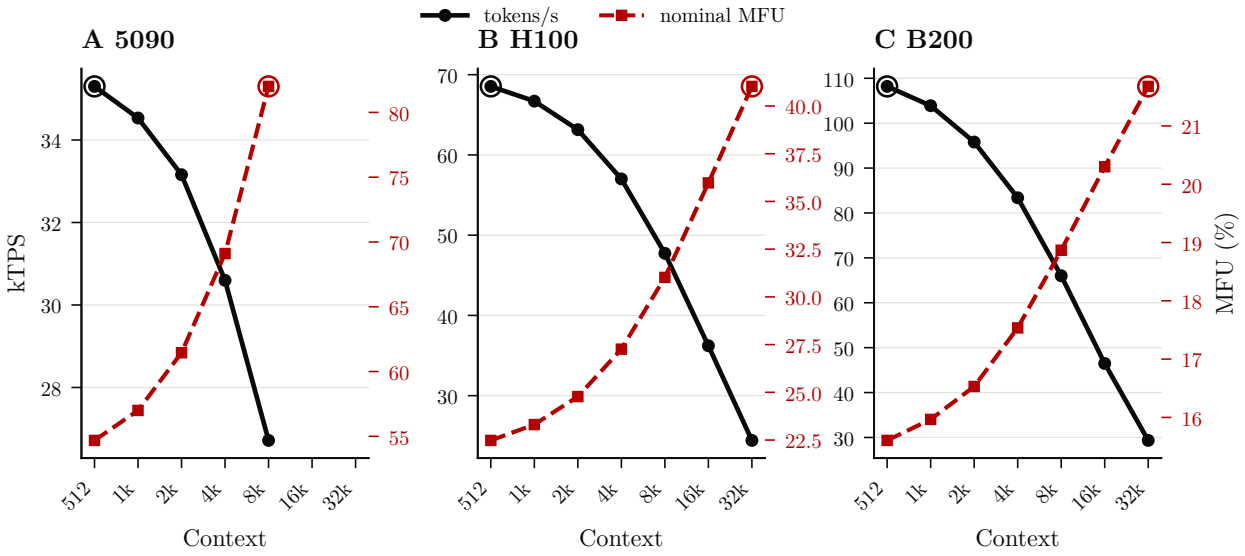

Figure 2. Context trade-off for the 500M model. Black circles show TPS; red squares show nominal-reference MFU. Open rings mark each metric's maximum within a panel. Batch is re-saturated at every context. Missing 5090 points at 16k and 32k did not fit. Longer contexts perform more attention FLOPs per token, so MFU may rise even when the number of training tokens processed per second falls.

This is not contradictory: the MFU numerator counts more modeled work per token as s grows. If the training plan fixes a token budget, the TPS maximum is the relevant single-GPU throughput choice. Context should instead be selected for the modeling objective, then batch should be saturated within that constraint.

4.3 Transferred prices change the projected frontier

Figure 3 is a price-transfer sensitivity analysis: it applies the same RunPod-measured TPS to the dated multi-provider snapshot. The cheapest projected SKU changes because catalogs, purchasing modes, and prices differ. Under the normalized snapshot, Vast.ai's RTX 5090 is the global minimum through 1B, RunPod Community's A6000 at 2B, Vast.ai's RTX PRO 6000 at 4B, and RunPod Secure's B200 at 8B. These are counterfactual price substitutions, not provider-specific benchmark results or permanent recommendations.

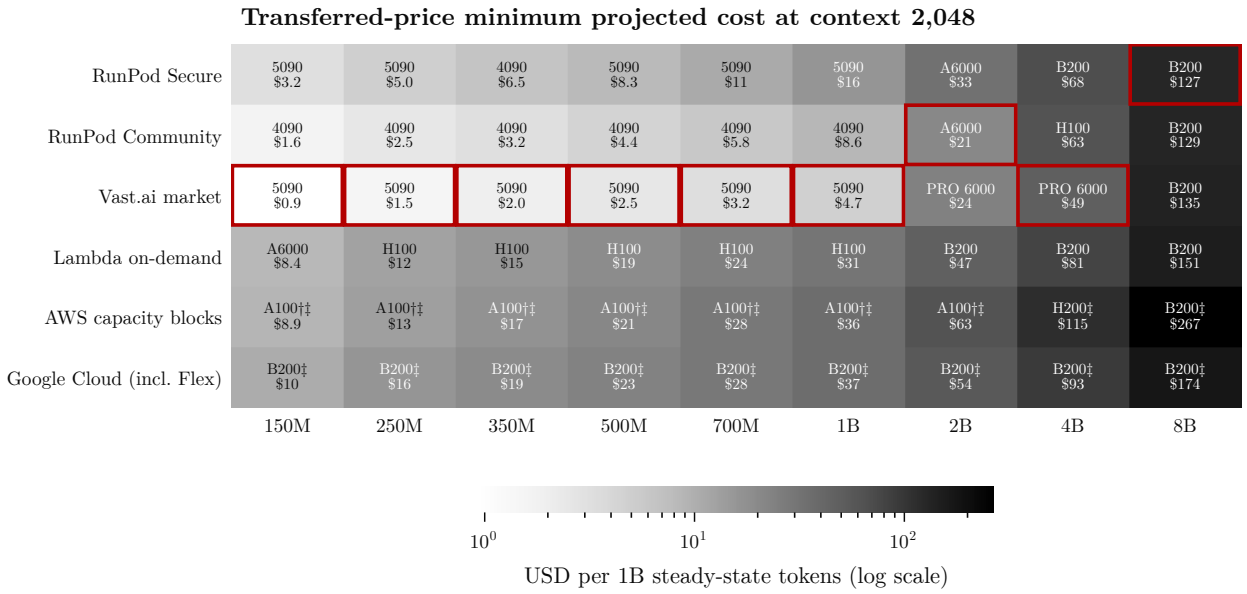

Figure 3. Transferred-price sensitivity at context 2,048, captured 27 July 2026. Each cell names the provider-specific minimum-cost measured GPU and projected USD per billion steady-state tokens; red outlines mark the cross-provider minimum in each model column. Throughput was measured on RunPod and transferred unchanged. † denotes a near-SKU rather than exact hardware match. ‡ denotes a product whose minimum instance contains multiple GPUs; its price is amortized per GPU and assumes all devices can be kept useful. Google Cloud includes Flex-start where no ordinary on-demand B200 price was listed.

4.4 Warmup and input-replicate repeatability

Across 430 matched anchors, the independently best batch is identical across input-seed replicates in 98.8% of cases. At the conservatively shared batch, the absolute TPS difference has median 0.069%, 95th percentile 0.386%, and maximum 0.815%. Figure 4 also shows why the first step is unsuitable: the illustrated B200 point begins at 703 ms and settles near 125 ms before measurement.

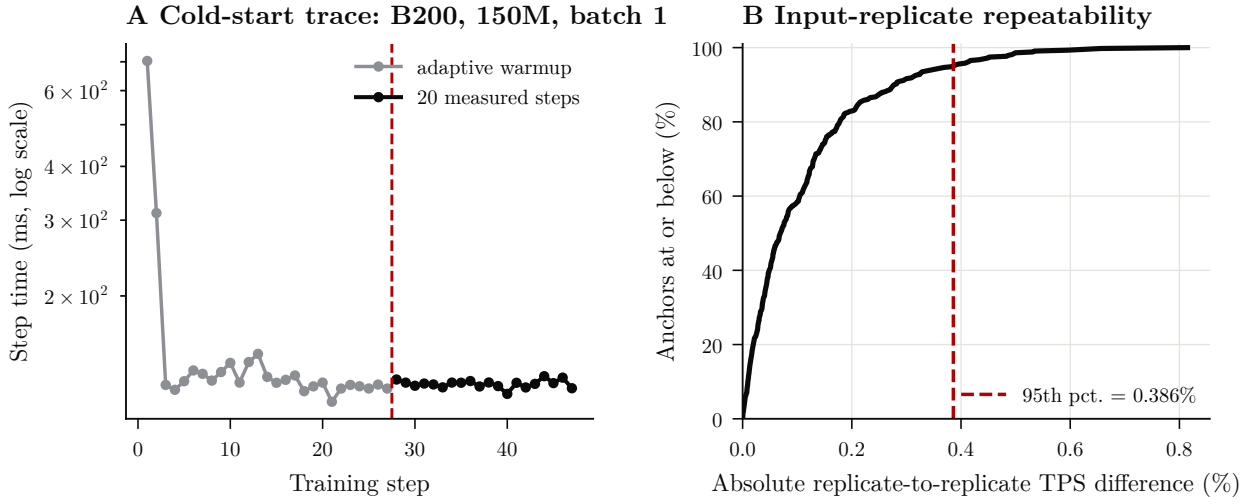

Figure 4. Timing quality control. A: all warmup and measured step times for the B200, 150M, context-2,048, batch-1, input-seed-11 replicate. The vertical rule separates adaptive warmup from the 20 retained steps. **B:** empirical CDF of absolute replicate-to-replicate TPS difference at the shared selected batch over 430 GPU/model/context anchors. The dashed rule marks the 0.386% 95th percentile.

These values establish short-run timing repeatability on the sampled machines. They are not confidence intervals over hosts, providers, driver versions, or long training jobs.

4.5 One reference sweep predicts unmeasured frontiers

A shared hardware/workload surrogate provides a bounded planning aid for model sizes between the measured anchors. Leaving out one entire GPU at a time, a specifications-only TPS model has 13.13% median absolute error (95% cluster-bootstrap CI 8.19–21.18%) and 26.48% 90th-percentile error. Replacing only its reference speed with one measured 250M/context-2,048 saturation sweep reduces these to 5.30% (4.50–6.43%) and 14.47% (12.71–15.56%). In the stricter test that also holds out the target model scale, calibrated median error is 5.26% (4.11–6.67%) over bracketed 350M–4B models.

Predictor	Median TPS error (%)	Fastest exact (%)	Cheapest exact (%)	Pareto F1 (%)
Peak dense BF16	–	26.7	18.3	36.9
Specifications only	13.1	20.0	**75.0**	69.7
One reference sweep	**5.3**	**72.9**	67.8	**87.9**

Table 2. Whole-GPU-holdout hardware selection from existing measurements. Decisions are evaluated per model/context workload among GPUs with a stable measured frontier and exact frozen RunPod Secure price. Peak BF16 is a ranking baseline, so an absolute TPS error is not defined. Pareto F1 treats membership in the measured time–cost frontier as a binary decision.

Across 59 model/context decisions, the calibrated model selects the measured fastest GPU in 72.9% of cases (95% workload-bootstrap CI 61.0–84.7%), the minimum-cost GPU in 67.8% (55.9–79.7%), and recovers time–cost Pareto membership at 87.9% micro-F1 (83.6–91.6%). When it misses the exact winner, nearby devices are often close: 90% of selected fastest-GPU decisions incur at most 5.3% throughput regret. The formula, capacity model, validation design, and cost-selection boundary appear in Appendix C. An actual 7B run remains necessary before treating a 7B estimate as measured evidence.

5 Discussion

5.1 Why the inversion is plausible

Dense BF16 peak describes a favorable, large Tensor Core workload. It does not encode whether a particular model supplies enough tiles, parallel waves, or arithmetic intensity, nor the relative cost of launches and non-matrix operations. Small GEMMs are especially vulnerable to tile and wave under-utilization [12]. OLM's readable eager graph also executes ordinary PyTorch modules around attention and matrix multiplication. The observed inversion is consistent with these mechanisms: the smaller GPUs often use a larger fraction of a lower peak, while the large accelerators remain fastest in absolute TPS. Establishing each kernel's causal contribution would require profiler and hardware-counter experiments beyond this study.

This eager baseline should not be read as defective software. It uses the framework's optimized scaled dot-product attention, usually selecting FlashAttention, and represents a reasonable PyTorch training workflow. `torch.compile` can reduce Python and launch overhead, including through lower-overhead modes intended for small batches [15]; its benefit is deliberately outside the present baseline. A compiled follow-up should rerun the complete frontier because compilation may change both the relative ordering and the batch at which saturation occurs.

5.2 A practical selection rule

Hardware choice should proceed in four steps. First, constrain model architecture and context from the learning objective. Second, discard devices whose reproducible shared batch does not fit. Third, saturate batch and compare TPS, not peak FLOPs. Fourth, compute a time–cost Pareto set using the purchasing mode actually available. MFU helps diagnose why a device underperforms its specification, but neither maximum MFU nor minimum hourly price alone selects the best training system.

The 2B example makes the trade-off concrete. The A6000 minimizes projected RunPod Secure compute cost, but the B200 yields nearly an order-of-magnitude shorter time for a modest cost premium. Conversely, through 1B, paying for the fastest accelerator does not proportionally reduce time enough to win the cost frontier. Teams can therefore choose a slower single device when deadlines are loose, or replicate a cheaper class only after measuring multi-GPU scaling and availability.

5.3 Reproducible economics

Hourly prices are mutable observations, not experimental constants. The paper therefore stores provider, product, region, price class, match quality, instance GPU count, source URL, and capture timestamp independently of the benchmark rows. Regenerating the price projection never changes measured TPS, MFU, or VRAM. This schema is also the basis for a future live explorer, where current price collectors can update the economic layer while preserving dated snapshots for reproducibility.

The SLMTrainBench artifact packages the derived CSVs, figure builder, provider collector, price snapshot, selection analysis, environment manifests, step timing, OOM records, telemetry, and raw JSON backups. The released OLM paper and library remain separate from this benchmark study [9].

6 Limitations

Scope of performance. The measurements cover one OLM Llama-style family, eager PyTorch 2.8, BF16 compute, and NVIDIA GPUs. They do not establish the ordering for compiled graphs, alternative attention or MLP implementations, FP8, activation checkpointing, quantized optimizers, other frameworks, or non-NVIDIA accelerators. Fixed synthetic tokens exclude dataloading, checkpoint I/O, evaluation, and network effects. The study measures throughput, not convergence or time-to-quality.

Hardware sampling. Most SKUs were measured on one rented board; the B300 extension crossed two physical devices under the same captured software and clock configuration. Two input-seed replicates test short-run timing and batch-selection repeatability, not machine-to-machine variance or training outcomes. Consumer cards also differ from datacenter products in ECC, support, reliability, tenancy, interconnect, and operational guarantees that are not priced here.

MFU accounting. The FLOP model omits elementwise operations and uses published dense peaks. Nominal-reference MFU is the primary result; configured-clock MFU linearly rescales peak on applicable boards and is only a sensitivity analysis. Neither is a direct count of all executed hardware operations. MFU can increase when context adds attention work even if TPS falls.

Price transfer. Only RunPod execution was measured. CPU, RAM, storage, driver, power limits, and virtualization differ across providers, so projected AWS, Google Cloud, Lambda, and Vast.ai costs may not reproduce RunPod TPS. Marketplace availability and prices can change within minutes. Capacity Blocks and some accelerator VMs have time or multi-GPU minimums; dividing their bundle price by GPU count is attainable only if every device is used efficiently. The projection excludes startup and idle time, storage, egress, failures, discounts, taxes, and energy or carbon accounting.

Single-GPU boundary. The benchmark cannot answer whether several cheap GPUs beat one datacenter GPU at equal wall time. Communication, host bandwidth, and parallelism strategy determine that comparison [11]. It requires a separate scaling-efficiency study rather than multiplying single-GPU TPS by device count.

Interpolation. The surrogate is validated only inside the measured 150M–8B, context-512–32,768 region and for the present architecture. GPU specifications are strongly correlated across the ten devices, leaving the FP32 coefficient weakly identified. Predictions are planning aids and should not replace validation at a publication's focal configuration.

7 Conclusion

For small-language-model pretraining, peak FLOPs and hourly price do not determine either speed or cost. Across ten GPUs, the measured eager-PyTorch frontier is scale-dependent: inexpensive RTX devices dominate the dated RunPod Secure price frontier through 1B parameters, an A6000 minimizes 2B compute cost at a large wall-time penalty, and the B200 leads at 4B and 8B. Longer context raises modeled utilization while lowering token throughput, making TPS, capacity, and dollars per token necessary companions to MFU.

The durable contribution is the measurement and accounting procedure: wait for stability, saturate batch, preserve OOM and VRAM evidence, replicate the selected point, and keep mutable provider prices separate from performance. The fitted capacity model and one-reference-sweep throughput calibration turn that procedure into a reusable decision surface: they predict intervening scales and recover most of the measured time–cost Pareto membership while retaining explicit uncertainty. SLMTrainBench therefore supports future price updates, compiled-stack experiments, and multi-GPU scaling measurements without rewriting the underlying evidence.

References

[1] Amazon Web Services. Amazon EC2 Capacity Blocks for ML Pricing. `https://aws.amazon.com/ec2/capacityblocks/pricing/`, 2026. Accessed 27 July 2026.

[2] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, et al. PaLM: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*, 2022. doi: 10.48550/arXiv.2204.02311.

[3] Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. FlashAttention: Fast and memory-efficient exact attention with IO-awareness. *arXiv preprint arXiv:2205.14135*, 2022. doi: 10.48550/arXiv.2205.14135.

[4] Google Cloud. Accelerator-optimized machine family pricing. `https://cloud.google.com/products/compute/pricing/accelerator-optimized`, 2026. Accessed 27 July 2026.

[5] Kamal Raj Kanakarajan, Bhuvana Kundumani, and Malaikannan Sankarasubbu. Small-Bench NLP: Benchmark for small single GPU trained models in natural language processing. *arXiv preprint arXiv:2109.10847*, 2021. doi: 10.48550/arXiv.2109.10847.

[6] Lambda. On-demand cloud GPU instances. `https://lambda.ai/instances`, 2026. Accessed 27 July 2026.

[7] Yin Li. L20-Edu-135M: An auditable single-GPU study of data-efficient small language modeling. *arXiv preprint arXiv:2606.22189*, 2026. doi: 10.48550/arXiv.2606.22189.

[8] Mingyu Liang, Hiwot Tadese Kassa, Wenyin Fu, Brian Coutinho, Louis Feng, and Christina Delimitrou. Lumos: Efficient performance modeling and estimation for large-scale LLM training. In *Proceedings of Machine Learning and Systems*, volume 7, 2025. URL `https://proceedings.mlsys.org/paper_files/paper/2025/hash/a66caa1703fe34705a4368c3014c1966-Abstract-Conference.html`.

[9] Tavish Mankash, Vardhaman Kalloli, Keshava Prasad, and Deepan Muthirayan. OpenLanguageModel: Readable and composable small-language-model pretraining for education and research. *arXiv preprint arXiv:2607.16669*, 2026. doi: 10.48550/arXiv.2607.16669.

[10] MLCommons. MLPerf Training Benchmark. https://mlcommons.org/benchmarks/training/, 2026. Accessed 27 July 2026.

[11] Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Anand Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, Julie Bernauer, Bryan Catanzaro, Amar Phanishayee, and Matei Zaharia. Efficient large-scale language model training on GPU clusters using Megatron-LM. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2021. doi: 10.1145/3458817.3476209.

[12] NVIDIA. Matrix multiplication background user's guide. https://docs.nvidia.com/deeplearning/performance/dl-performance-matrix-multiplication/index.html, 2026. Accessed 27 July 2026.

[13] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

[14] Connor Pedersen, Dong H. Ahn, Michel Migdal, Collin Neale, and Nik Konyuchenko. Instant GPU efficiency visibility at fleet scale. *arXiv preprint arXiv:2605.20799*, 2026. doi: 10.48550/arXiv.2605.20799.

[15] PyTorch Contributors. `torch.compile` documentation. https://docs.pytorch.org/docs/stable/generated/torch.compile.html, 2026. Accessed 27 July 2026.

[16] Hang Qi, Evan R. Sparks, and Ameet Talwalkar. Paleo: A performance model for deep neural networks. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=SyVVJ85lg.

[17] Praveen Rao. Performance of small language model pretraining on FABRIC: An empirical study. *arXiv preprint arXiv:2602.02632*, 2026. doi: 10.48550/arXiv.2602.02632.

[18] RunPod. GPU cloud pricing. https://www.runpod.io/pricing, 2026. Accessed 27 July 2026.

[19] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-LM: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019. doi: 10.48550/arXiv.1909.08053.

[20] Vast.ai. Instance pricing. https://docs.vast.ai/guides/instances/pricing, 2026. Accessed 27 July 2026.

[21] Zhen Xie, Murali Emani, Xiaodong Yu, Dingwen Tao, Xin He, Pengfei Su, Keren Zhou, and Venkatram Vishwanath. Centimani: Enabling fast AI accelerator selection for DNN training with a novel performance predictor. In *2024 USENIX Annual Technical Conference (USENIX ATC 24)*, pages 1203–1221. USENIX Association, 2024. URL https://www.usenix.org/conference/atc24/presentation/xie.

[22] Geoffrey X. Yu, Yubo Gao, Pavel Golikov, and Gennady Pekhimenko. Habitat: A runtime-based computational performance predictor for deep neural network training. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*. USENIX Association, 2021. URL https://www.usenix.org/conference/atc21/presentation/yu.

[23] Hongyu Zhu, Amar Phanishayee, and Gennady Pekhimenko. Daydream: Accurately estimating the efficacy of optimizations for DNN training. In *2020 USENIX Annual Technical Conference (USENIX ATC 20)*, pages 337–352. USENIX Association, 2020. URL https://www.usenix.org/conference/atc20/presentation/zhu-hongyu.

A Hardware and Workloads

GPU	Generation	VRAM (GB)	Bandwidth (GB/s)	Nominal peak (BF16 TF/s)	Configured peak (BF16 TF/s)	Anchors
A6000	Ampere	48	768	154.8	180.6	39
4090	Ada	24	1008	165.2	205.5	27
6000 Ada	Ada	48	960	364.2	451.4	7
5090	Blackwell	32	1792	209.5	268.9	31
A100	Ampere	80	1935	312.0	312.0	49
PRO 6000	Blackwell	96	1597	468.0	468.0	50
H100	Hopper	80	3350	989.5	989.5	49
H200	Hopper	141	4800	989.5	989.5	56
B200	Blackwell	180	8000	2250.0	2250.0	60
B300	Blackwell	288	8000	2250.0	2250.0	62

Table 3. Measured devices and dense BF16 MFU denominators. Nominal peak is the main denominator. Configured peak is a sensitivity value that adjusts nominal peak when the captured graphics clock differed from the reference clock. Bandwidth is the published device-memory peak.

Label	Parameters	Width	MLP width	Layers	Q heads	KV heads
150M	150,436,608	768	2,048	20	12	4
250M	246,397,312	896	2,432	26	14	2
350M	348,447,744	1,024	2,816	28	16	4
500M	505,516,800	1,280	3,456	27	20	4
700M	707,480,064	1,536	4,096	27	24	4
1B	995,135,232	1,792	4,864	28	28	4
2B	2,008,824,320	2,560	6,912	28	20	4
4B	3,999,611,392	3,584	9,728	29	28	4
8B	7,983,108,096	4,096	14,336	36	32	8

Table 4. Exact benchmark model shapes. Every model uses a 32,000-token vocabulary, tied input/output embeddings, and the same OLM Llama-style component family.

The hardware table's "anchors" count is the number of GPU/model/context combinations with a stable batch shared by both input-seed replicates. The complete dataset has 2,963 rows: 2,105 complete stable rows and 858 OOM or model-build OOM rows. The RTX 6000 Ada has a complete dual-replicate 2,048-token baseline but only one replicate for most context extensions; the shared-replicate rule therefore retains seven anchors for that device rather than silently mixing replicate counts. Four RTX PRO 6000 edge-of-capacity configurations fit input seed 11 but OOMed under input seed 22; the larger batch is excluded from the shared frontier.

B Selection and Timing Details

For each workload anchor, let $T_{r,b}$ be TPS under input-replicate label r and batch b. With $\mathcal{R} = \{11, 22\}$, the selected batch is

$$b^\star = \arg \max_{b:\, \forall r \in \mathcal{R},\, T_{r,b} \text{ stable}} \text{median}_{r \in \mathcal{R}}\, T_{r,b}.$$

Every main-text metric is then a median across $T_{r,b^\star}$ or the corresponding MFU/VRAM value. This conservative intersection prevents a batch that sits on an unstable fit/OOM boundary from becoming the recommendation.

The stability statistic uses the median of adjacent five-step windows and 1.4826 MAD/median for robust relative jitter. Raw files retain every warmup and measurement time, loss value, nanosecond wall-clock boundary, selected attention backend, and a 200-ms `nvidia-smi` trace.

C Shared Hardware Surrogate

C.1 Peak allocated VRAM

Let $P_B = P/10^9$, b be batch, s context, h hidden width, n_ℓ layer count, and V vocabulary size. A nonnegative linear fit gives decimal GB

$$\widehat{M} = 14.7684 P_B + 54.0059 \frac{bsn_\ell h}{10^9} + 12.1746 \frac{bsV}{10^9}.$$

The 95% bootstrap intervals for the three coefficients are $[14.554, 14.986]$, $[53.609, 54.467]$, and $[11.801, 12.507]$. Leave-one-model-size-out validation over 279 shapes has 0.89% median absolute error (95% CI 0.61–1.65%) and 3.54% 90th-percentile error (2.24–5.16%). A 99% runtime-reported capacity rule classifies 1,519 of 1,520 observed fit/OOM shapes correctly. The approximately 15 GB per billion parameters is consistent with FP32 weights, gradients, and two Adam moments.

C.2 TPS

Use the 250M, context-2,048 workload as reference. Define

$$\begin{aligned} p &= \ln(P/246{,}397{,}312), & \ell &= \ln(s/2048), \\ f &= \ln(F_{32}/67), & w &= \ln(W/1935), \end{aligned}$$

where F_{32} is ordinary FP32 TFLOP/s and W is memory bandwidth in GB/s. The specifications-only reference is

$$\ln S = 10.96594 + 0.21737 f + 0.62655 w,$$

and the shared workload response is

$$\begin{aligned} R = {} & -0.680856 p - 0.120413 \ell - 0.033490 p^2 + 0.028013 p\ell - 0.057804 \ell^2 \\ & - 0.006811 fp + 0.021651 f\ell + 0.071141 wp - 0.069424 w\ell. \end{aligned}$$

Then $\widehat{\mathrm{TPS}} = S \exp(R)$. The calibrated version replaces only S with measured best TPS from one target-GPU 250M/context-2,048 saturation sweep. Cluster-bootstrap intervals for the specification intercept, FP32 exponent, and bandwidth exponent are respectively $[10.8916, 11.1410]$, $[-0.4544, 0.4432]$, and $[0.5185, 0.7245]$. The wide FP32 interval reflects ten correlated hardware designs; bandwidth is more stably identified.

Whole-GPU holdout empirical central-95% prediction multipliers are $[0.759, 1.386]$ for specifications only and $[0.847, 1.205]$ after the reference sweep. These are individual-prediction bands, not intervals for a mean. The largest calibrated errors occur at the sparse H200/8B boundary.

For the hardware-choice evaluation, we group whole-GPU-holdout predictions by model and context. Candidates must have a stable measured frontier and an exact rate in the frozen RunPod Secure snapshot. The peak-BF16 baseline ranks these candidates by nominal dense peak; the two regressions rank by predicted TPS. Fastest selection maximizes TPS, cheapest selection minimizes rate/TPS, and Pareto recovery compares nondominated membership in hours and USD per billion tokens. The one-sweep calibration workload is excluded from its own evaluation, leaving 59 decisions.

Reported selection intervals resample whole model/context decisions 5,000 times; they do not represent new hosts or providers.

The memory rule selects the largest power-of-two batch whose predicted allocation is at most 99% of runtime capacity. It exactly recovers 84.88% of measured best batches, is within one power of two for 96.05%, and within two for 100%. Adjacent batches often lie on a flat throughput plateau.

D Price Provenance and Reproduction

The normalized JSON snapshot contains 43 quotes and the following source classes:

Provider	Price class	Snapshot interpretation
RunPod	Secure and Community	Exact listed SKU where available
Vast.ai	Marketplace on-demand	Median eligible offer rule in Section 3.5
Lambda	On-demand	Advertised instance rate, normalized per GPU
AWS	Capacity Block	Region-specific block rate, normalized per GPU
Google Cloud	On-demand or Flex-start	Full accelerator-VM rate, normalized per GPU

Table 5. Provider price surfaces represented in the 27 July 2026 snapshot. Product name, region, exact/near match, minimum GPU count, notes, source URL, and timestamp are preserved for every quote.

From the paper directory, `make figures` rebuilds all four figures and compact CSV tables from the canonical benchmark JSON and frozen price snapshot; `make pdf` compiles the manuscript. Price collection is a separate explicit command so rebuilding the paper cannot silently alter its economics. The source tree retains the exact script and source-data path for every generated panel.