PURO-2B: *POOR* LAB'S QWEN2-1.5B TRAINED ON RTX 5090 WITHIN $5090

Anonymous authors
Paper under double-blind review

ABSTRACT

Language model pretraining has become almost synonymous with prohibitive cost, placing it out of reach for much of the academic and open-source communities. Although strong open-source efforts already exist, including open-weight models and open-source training recipes, a *cost-efficient, hardware-accessible, and open-source* pretraining recipe has long been missing. Even at a small scale, training Llama-3.2-3B costs over $1.5M, and reproducing SmolLM3-3B needs over $700K. In this report, we present an open pretraining recipe designed to lower this barrier. Using this recipe, we train a collection of PURO-2B models from scratch on up to 1.4 trillion tokens with FP8 precision on consumer-grade RTX 5090 GPUs. The models differ in token budgets and selected recipe variants. Our best model is trained for less than $6.9K in compute and approaches Qwen2.5-1.5B performance under our evaluation protocol. This cost efficiency comes from combining hardware selection, low-precision training, hyperball optimization, curriculum model averaging, and the data recipe. Beyond the recipe, we provide two additional results. First, across the PURO-2B collection, we derive a *PURO Cost Scaling Law* that relates training cost to average model performance. The fitted law shows that about $4.4K, *less than $5,090*, is sufficient to reach the performance of Qwen2-1.5B. Second, our end-to-end case study shows that the curriculum-based pretraining advantage persists under matched supervised fine-tuning, with higher performance on mathematical and general-purpose tasks. Access to the full pretraining pipeline rather than model weights alone enables such controlled studies. We release the full PURO-2B training recipe, including data, code, and model weights.

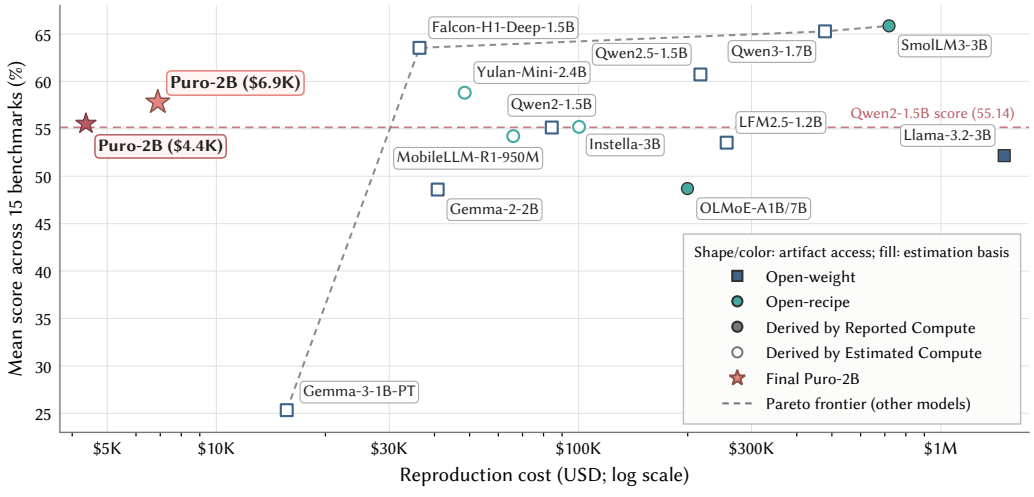


Figure 1: Model performance versus reproduction cost under the accounting protocol in Section E. Performance is measured by the average scores over the 15 mathematics, code, reasoning, and knowledge benchmarks in Table 2. Costs of two PURO-2B runs are summarized in Table 19.

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

1 INTRODUCTION

Scaling parameters and training data has substantially improved large language models (LLMs) (Hoffmann et al., 2022; Kaplan et al., 2020). However, prohibitive costs and limited access to training artifacts make it difficult for academic and open-source researchers to study training behavior, reproduce complete pipelines, and test alternatives at meaningful scales. Open-recipe projects such as OLMo, SmolLM, and Marin go beyond open weights by releasing data mixtures, training code, configurations, and checkpoints (Walsh et al., 2025; Allal et al., 2025; Bakouch et al., 2025; Hall et al., 2025). Although these releases make pretraining more transparent, they are not necessarily affordable to reproduce.

Under shared reference rental rates (Figure 1 and § E), we estimate that reproducing OLMoE-1B-7B would cost $200K and SmolLM3-3B would cost $719K. These estimates are based on reported or estimated training compute. Even transparent open-recipe releases can therefore remain *open to the world but out of reach for many small and resource-constrained research labs—the poor labs*, leaving a practical gap between reproducibility and accessibility.

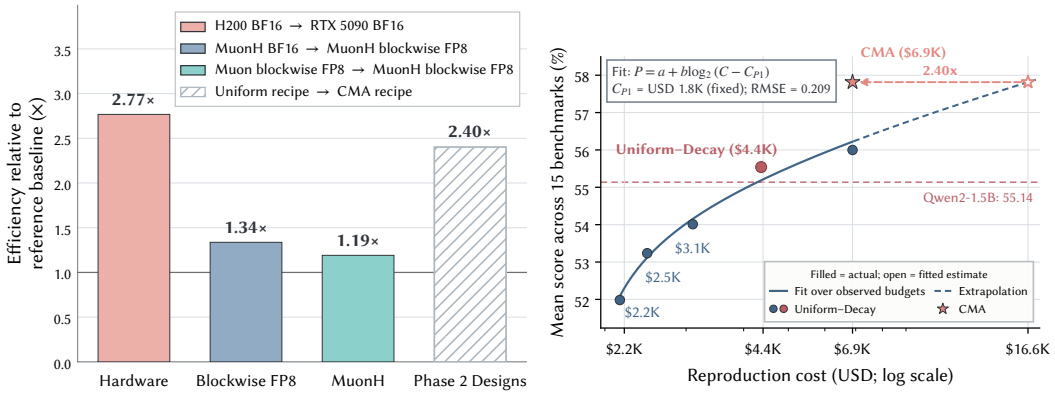
To bridge this gap, we present a reproducible training recipe covering systems, algorithms, and data to train a collection of **PURO-2B** checkpoints[1]. The recipe targets low-cost, accessible infrastructure for billion-parameter, trillion-token dense pretraining and produces a 2B model with performance matched against Qwen2-1.5B and Qwen2.5-1.5B under our protocol. These checkpoints share the same architecture but differ in training budget and recipe variants. Training uses predominantly open-source datasets and runs on a cluster of consumer-grade RTX 5090 GPUs. Figure 1 compares counterpart models under a common cost-accounting protocol, where it can be clearly seen that PURO-2B attains competitive performance at the lowest reproduction cost among the plotted models. Moreover, a checkpoint trained for about $4.4K in compute surpasses Qwen2-1.5B overall, while our best checkpoint approaches Qwen2.5-1.5B at about $6.9K. The best run uses a *two-phase* schedule, includes 1.4T tokens and requires 22,514 training GPU-hours, spanning 17.6 days of training (Table 19). Cost details are in Sections 3.2 and E.

Where does the cost efficiency come from? Our PURO-2B recipe combines improvements across three aspects of pretraining: (1) *Infrastructure.* We use **consumer-grade RTX 5090 GPUs** as a widely accessible platform with lower cost per FLOP in our setting (Sections 2.1 and D.1). We further use blockwise **FP8 training**, supported by the Blackwell architecture, to increase computational throughput while maintaining model quality comparable to bfloat16 (BF16) training (Sections 2.1 and D.2). (2) *Optimization.* We apply the **MuonH optimizer**, which extends the Muon optimizer with hyperball constraints on parameter weights and updates. We show that the observed convergence benefits of hyperball constraints are largely explained by the *effective learning rate (LR) schedule* and use this finding to guide LR schedule design (Sections 2.2 and D.3). (3) *Data strategy.* We conduct **dataset benchmarking** via proxy experiments to guide dataset selection and mixture design from a large pool of candidate datasets (Sections 2.3.2 and D.6). We further apply **Curriculum Model Averaging (CMA)** to make better use of high-quality data (Luo et al., 2026) (Sections 2.3.1 and D.4). Together, these choices address bottlenecks across the training pipeline, and we also conduct targeted ablations of these key design choices and report cost-efficiency estimates (Section 3.2).

Importantly, these components are not simply independent techniques. Instead, they form a co-designed training system with interactions across pipeline layers. For example, the Blackwell architecture of RTX 5090 provides the hardware foundation for efficient FP8 training; both MuonH and CMA have LR schedule preferences, requiring careful coordination; CMA needs data arrangement, which relies on the data preprocessing system. Together, these designs enable our cost efficiency.

Furthermore, we formulate a **PURO Cost Scaling Law**, which fits a function between model capability and training budget. This law tells us how good a model we can train for a given scaled-down cost. We use a basic recipe (*Uniform–Decay (UD)*) with uniform data ordering and scheduled LR decay. Since our training has two phases, we resume training from the shared Phase 1 checkpoints and adopt different Phase 2 token budgets. We then evaluate the resulting checkpoints and fit model performance as a function of total training cost. This curve provides a reference for achievable model capability under a limited budget with our recipe. According to the scaling trend, **our recipe can surpass the Qwen2-1.5B average performance at a cost of USD 4.4K**. This trend also serves as

[1] Unless otherwise noted, we will use PURO-2B to denote the best one in this collection.



(a) Relative Cost-Efficiency Improvement.

(b) PURO Cost Scaling Law (scale-down)

Figure 2: **(a)** Cost-efficiency improvements from hardware, FP8, MuonH, and Phase 2 recipes, expressed as equivalent USD savings. Each bar uses its own baseline. **(b)** PURO Cost Scaling Law is fitted to Phase 2 runs using **Uniform–Decay (UD)** at different token budgets. The $4.4K UD checkpoint already exceeds Qwen2-1.5B in average performance over the 15 benchmarks in Table 2. At the same full Phase 2 budget, **Curriculum Model Averaging (CMA)** replaces uniform ordering with a more complex data and training strategy, achieving 2.40× cost-efficiency gains. Filled markers denote actual reproduction costs, while hollow markers denote fitted uniform-equivalent costs. See Section D.4 and Tables 19 and 20 for details.

a reference for advanced recipes using *Curriculum–Decay (CD)* and *Curriculum Model Averaging (CMA)*, with details in Sections 3.2 and D.4.

Since our PURO-2B recipe lowers the pretraining cost, it enables end-to-end evaluation of pretraining designs by comparing different pretraining models after post-training. This matters because the best design choices for pretraining are not necessarily the best after post-training. As a case study, we examine whether the performance difference between the UD and CMA Phase 2 recipes persists after matched supervised adaptation. In GSM8K-based SFT and Math&Code SFT with replay, models initialized from the CMA checkpoint achieve higher GSM8K accuracy across repeated runs. In the Tulu-3 mixed-domain SFT setting, models initialized from CMA also achieve higher overall benchmark performance and better instruction-following performance.

In summary, our contributions are:

1. We provide a *cost-efficient, hardware-accessible* pretraining recipe and train a PURO-2B model collection. A checkpoint in this collection exceeds Qwen2-1.5B at about $4.4K, and the best checkpoint approaches Qwen2.5-1.5B at about $6.9K. More broadly, *our main contribution is to show that affordable, open pretraining is practical today*.
2. We evaluate the main recipe choices: RTX 5090 infrastructure, blockwise FP8, MuonH, and curriculum model averaging with supporting ablation experiments. Moreover, we formulate the PURO Cost Scaling Law relating cost to performance at reduced budgets.
3. We release the datasets, model weights, and other assets for reproducing and inspecting this pretraining recipe. We also provide a post-training case study showing how this recipe enables end-to-end comparison of one pretraining design.

2 TRAINING RECIPE

Pipeline overview. Our PURO-2B model primarily follows the Qwen3-1.7B architecture. But it unties the input and output embeddings, so it has 1.4B non-embedding parameters and 2B total parameters. We pretrain the model from scratch. The pipeline starts with constructing the training corpus from publicly accessible sources. Then we choose cost-efficient, widely accessible RTX 5090 hardware and improve its system efficiency. It then proceeds through two phases of pretraining. We improve the convergence rate via hyperball optimization and LR schedule design.We also use a data curriculum and model averaging strategy in Phase 2 and model averaging to better utilize high-quality data. The overall pipeline is illustrated in Figure 3. Detailed recipe records are provided in Section D.

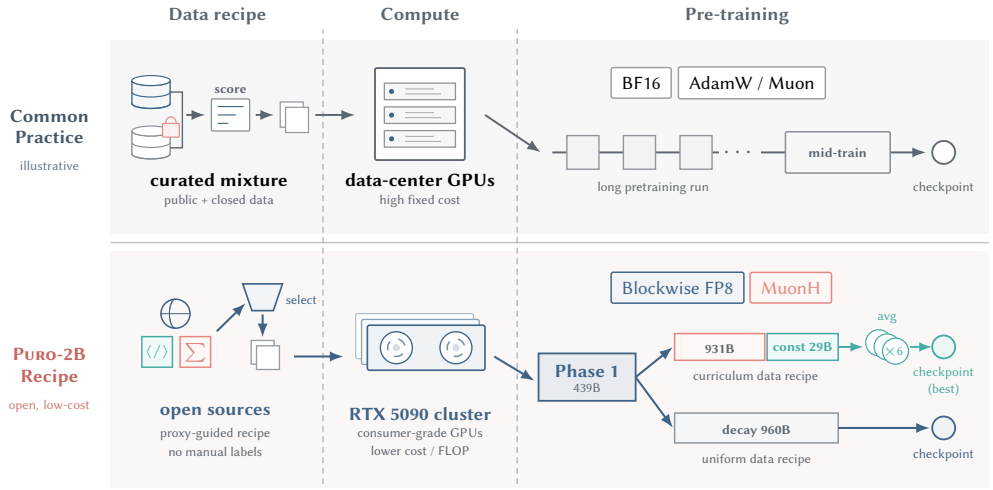


Figure 3: **Puro-2B pipeline for *poor*-lab design.** The upper row shows common practice; the lower row shows *our open, low-cost Puro-2B recipe*. We build the training dataset from selected publicly accessible sources (Section D.6), without elaborative data scoring and curation. Our consumer-grade RTX 5090 cluster offers higher cost efficiency (Section D.1), in contrast to costly data-center GPUs. Pretraining has two phases, with two headline Phase 2 variants: **Uniform–Decay (UD)** and **Curriculum Model Averaging (CMA)** with late constant-LR continuation and six-checkpoint averaging. Both variants use 960B tokens in Phase 2 (Section D.4). By comparison, common practice uses long pretraining followed by shorter mid-training with LR annealing to obtain the final checkpoint. To further improve cost efficiency, we adopt blockwise FP8 training (Section D.2) and Muon with Hyperball optimizer (Section D.3) throughout Phase 1 and Phase 2.

2.1 RTX 5090 GPUs for Pretraining: Cost-Efficient and Accessible

For affordable pretraining in resource-constrained labs, we select consumer-grade RTX 5090 GPUs for favorable peak FLOP per dollar and accessibility. We address memory and bandwidth limits by optimizing intra- and inter-node communication and balancing computation and memory across GPUs. We further improve efficiency with their support for blockwise FP8 mixed-precision training.

RTX 5090 offers higher peak FLOPs per dollar. To make our recipe cost-efficient and broadly accessible, we choose a non-standard infrastructure: we run all pretraining and post-training on a consumer-grade RTX 5090 cluster instead of renting data-center accelerators. Since Puro-2B uses the Qwen3-1.7B architecture, our cost analysis shows that RTX 5090 is a better fit for this workload, although data-center GPUs are generally expected to be more stable and compute-efficient. As shown in Table 1, while RTX 5090 lags behind data-center GPUs in absolute peak throughput[2], its much lower effective price gives substantially higher FLOP per dollar: *BF16 and FP8 cost efficiency of RTX 5090 is about 2.7× that of H200.* Accounting details are in Section D.1.1.

Table 1: Peak hardware capability and cost efficiency of candidate NVIDIA GPUs; accounting details in Table 3.

GPU Model	Peak tensor performance (TFLOPS)[a]			Memory capacity	Memory bandwidth	TDP	Intra-node P2P BW (Bi-Di)	Single-GPU price[b]	Peak compute/$ (EFLOP/USD)		
	BF16	FP8	NVFP4						BF16	FP8	FP4
A100 SXM4	312	N/A	N/A	80 GB	2 TB/s	400 W	600 GB/s	$1.79/h	0.63	N/A	N/A
H200 SXM5	989.5	1979 [c]	N/A	141 GB	4.8 TB/s	700 W	900 GB/s	$4.00/h	0.89	1.78	N/A
RTX PRO 6000	503.8	1007.6	2015.2	96 GB	1.8 TB/s	600 W	128 GB/s [d]	$1.89/h	0.96	1.92	3.85
RTX 5090	209.5	419	1676	32 GB	1.8 TB/s	575 W	128 GB/s [d]	**$0.31/h**	**2.43**	**4.87**	**19.46**

Optimizing intra-node and inter-node communication of RTX 5090. Despite the cost efficiency of a single GPU, RTX 5090 still has significant hardware limitations. RTX 5090 lacks NVLink for high-bandwidth GPU-to-GPU scaling. Moreover, PCIe P2P for intra-node communication and GPUDirect RDMA (shortened as GDR) for inter-node communication are not supported. Without these features, GPU communication must pass through host memory, also known as "ping-pong" transfer. We perform optimizations based on the open-source driver to enable PCIe P2P for intra-node communication, and fix the CUDA user-space driver to enable GDR. After enabling PCIe P2P,

[2]Notably, RTX 5090 Tensor Cores are artificially limited to half peak performance with FP32 accumulation.

we increase the bidirectional bandwidth from 32GB/s to 111 GB/s, close to the theoretical limits of 128GB/s. The latency decreases drastically from 14.3 μs to 0.4 μs. These improvements increase eight-GPU PCIe `AllReduce` bandwidth, `nccl-tests` `busbw`, from 14.75 GB/s to 27.34 GB/s. In end-to-end tests, enabling GDR improved the 24-GPU (3 servers) `AllReduce` bandwidth (`busbw`) from approximately 8.87 GB/s to 19.93 GB/s. Implementation details are in Section D.1.2.

Hardware-aware parallelism and memory balancing improve RTX 5090 utilization. Given the limited bandwidth even after optimization, we use data parallelism (DP) and pipeline parallelism (PP), while avoiding tensor parallelism (TP) due to its frequent per-layer collectives. Compared with data-center-class systems, our RTX 5090 platform faces a more pronounced communication bottleneck within each node. We therefore use topology-aware rank mapping to assign each PP group to topologically close GPUs. Since the relatively large vocabulary makes the LM head a major source of FLOPs, we use our extended analytical FLOP estimator to rebalance Transformer-layer placement across pipeline stages, together with a memory-aware optimizer placement that balances per-device memory usage (DeepSeek-AI et al., 2026). Implementation details are in Section D.1.3.

RTX 5090 enables blockwise FP8 with a net speedup over BF16. Overall, we train directly with FP8 mixed precision from random initialization, without a higher-precision warm-up stage. To mitigate the sensitivity of per-tensor scaling to outliers, we adopt blockwise FP8 (NVIDIA Corporation, 2026b) supported on RTX 5090, following the recipe introduced by DeepSeek-V3 (DeepSeek-AI et al., 2024): activations and activation gradients use 1×128 blocks, while weights use 128×128 blocks. Linear-layer GEMM operands and saved activations use E4M3 (Micikevicius et al., 2022), whereas precision-sensitive operations, communication tensors, and checkpoint weights remain in higher precision, as illustrated in Figure 7. Across five ladder scales, FP8 exhibits a modest but consistent validation loss gap relative to BF16. Nevertheless, its throughput gain outweighs this quality penalty, yielding a net speedup at matched validation loss.

2.2 MuonH and Effective Learning Rate Schedule Design

After discussing hardware and training infrastructure optimization, we ask how to lower loss within a fixed token budget. We focus on the interaction between the optimizer and the learning rate schedule. Puro adopts MuonH (Wen et al., 2026), which explicitly controls the update magnitude relative to the weight scale. Our key observation is that, using Muon to match MuonH's effective learning rate (ELR) schedule without normalization largely reproduces its convergence advantage. This motivates production LR schedule design from the ELR perspective.

Preliminary: MuonH explicitly controls the update size relative to weight norms. We use MuonH (Wen et al., 2026) for approximately scale-invariant (explained in Section D.3.1) attention and MLP matrices, and AdamW for the remaining parameters. MuonH is built upon Muon (Jordan et al., 2024): it follows the Muon update direction, but also normalizes the update magnitude and projects the updated weight matrix onto a fixed-radius hyperball. Intuitively, these operations prevent the scales of both the update and the weight from drifting freely. For a weight matrix W_t, Muon update u_t, and scalar learning rate η_t, the *effective learning rate* (ELR) is defined as the update size relative to the weight scale: $\rho_t(W) = \eta_t \|u_t\|_F / \|W_t\|_F$. For ordinary Muon, both $\|u_t\|_F$ and $\|W_t\|_F$ evolve during training, so the scalar LR η_t alone does not determine the ELR. In MuonH, the weight is maintained at a fixed norm $R = \|W_0\|_F$, while the normalized Muon update is also scaled to norm R. Consequently, the hyperball LR η_t directly specifies the ELR. This explicit control will be central to our analysis below. The full MuonH update rule and detailed derivation are in Section D.3.

Matching the effective LR nearly reproduces MuonH's convergence speed. We find that MuonH's advantage largely comes from the ELR schedule rather than normalization. In our production experiments, as exemplified in Figure 9, MuonH achieves a lower final loss than Muon. To diagnose the effective factors, we design *Muon-ELR*, a Muon variant bridging ordinary Muon and MuonH. We set the MuonH run's LR schedule as Muon-ELR's target ELR schedule. Then, at each step, Muon-ELR only rescales the weight matrix's learning rate using the current update and weight norms to match the target ELR. Muon-ELR does not normalize or project the update or weight matrix. Then we find that Muon-ELR nearly reproduces MuonH's loss curve: matching the effective LR nearly matches its convergence rate without the hyperball constraint. As shown in Tables 17 and 18, this holds across different ELR schedules and weight decay values in our ablation (Section I.1).

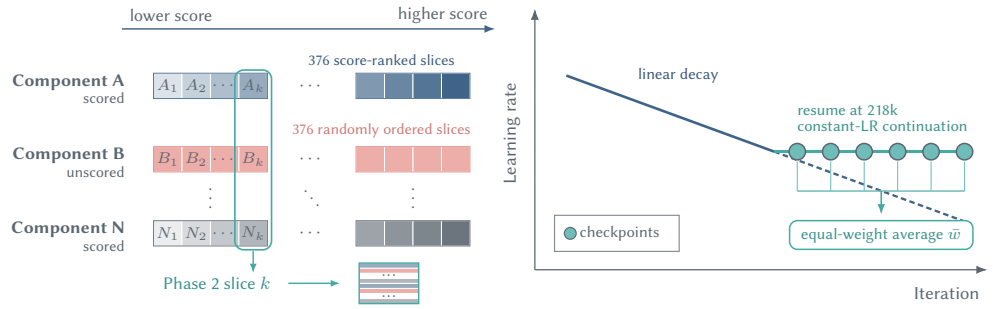


Figure 4: **CMA-inspired data and optimization flow in Phase 2. Left:** Sort within each component and merge blocks by training progress; **Right:** use a late constant-LR continuation and average last 6 checkpoints. Details of bucket construction and model averaging are given in Sections K.1 to K.3.

Furthermore, the loss convergence rate is highly related to the ELR schedule. Muon's ELR decays fast early and slow later, while MuonH's ELR is set to decay linearly. Correspondingly, Muon's loss decays fast early and slow later, and MuonH surpasses Muon near the end. We further adapt the form of an LR schedule scaling law to the ELR schedule and find that it still holds (Section I). This law further explains how ELR schedule differences lead Muon to lag behind MuonH and Muon-ELR. These results suggest that optimizer choice alone does not determine performance, while the ELR schedule also plays a key role. Hence, beyond choosing an optimizer, we should design a better effective LR schedule. More details are discussed in Sections D.3.1 and I.

LR schedule design: open-ended power decay followed by long linear decay. To design a good ELR schedule, we first compare existing LR schedule shapes, including cosine (Loshchilov & Hutter, 2017) and warmup-stable-decay (WSD) (Hu et al., 2024). As shown in Table 17, cosine and linear perform similarly and better than WSD with a 20% decay range. Hence, the decay ratio can affect convergence beyond the schedule shape. We then experiment with MuonH and WSD LR schedules. We ablate two factors: peak LR and data size in tokens per parameter. As shown in Figure 10, we find that (1) a larger peak LR prefers a longer decay range, and (2) a longer training horizon calls for a longer decay range. Additionally, the final token budget may not be known in advance (Hu et al., 2024; Shen et al., 2024b; Allal et al., 2025). Considering this scenario, we adopt two-phase training. Phase 1 therefore uses an open-ended power schedule that allows training to continue beyond the initial horizon (Equation (10)). Following the sweep results, Phase 2 uses a long linear decay rather than a short terminal drop (Figure 14). Due to compute limits, the ratio was not selected by a sweep.

2.3 Curriculum Training and Data Selection

2.3.1 Multi-Source Curriculum Model Averaging

Except for optimization, data efficiency also depends on data arrangement. High-quality data is sparse but important in the pretraining corpus; placing better data later can improve their utilization. However, prior work (Luo et al., 2026) shows that such a data curriculum can conflict with LR decay: when the LR becomes too small in later steps, the model cannot fully learn from the better data. It therefore proposes Curriculum Model Averaging (CMA), which combines an ascending-quality data curriculum with constant-LR training and model averaging. We extend this paradigm to practical multi-source pretraining in two ways: we construct a multi-source curriculum with an approximately stable mixture and adapt CMA to our training schedule with a constant-LR final stage after LR decay. We also provide component ablations in Section D.4.

Putting better data later while keeping the source mixture stable. Many open-source datasets provide sample quality scores. Using these scores as quality proxies, we order samples by increasing estimated quality within each source. For sources without usable scores, we simply shuffle samples. Since score meanings differ across sources, we do not directly compare scores between sources. We then divide each source into equal-fraction blocks and mix corresponding blocks across sources. The mixed samples within each block are further shuffled. In this way, training progresses toward higher-quality regions within each scored source, while the local source mixture remains approximately stable throughout training (Figure 4). Phase 2 curriculum composition is in Tables 21 and 22.

LR decay followed by constant-LR training and checkpoint averaging. Putting better data later is not enough. In practice, data curriculum can conflict with LR decay: small late-stage LRs weaken

Table 2: Evaluation on 15 benchmarks. Avg. is the unweighted mean over all displayed benchmarks.

Model	Params	Math & Code				Reasoning & Knowledge											Avg.
		GSM8K	MATH	MBPP	HEval	MMLU	M-Pro	ARC-C	ARC-E	BoolQ	CSQA	HSwag	PIQA	SIQA	Wino	BBH	
Open-Weight Models																	
Qwen2-1.5B	1.5B	59.8	23.7	50.2	27.4	56.4	23.6	69.8	83.8	71.9	70.5	59.9	75.2	63.4	59.7	31.7	55.1
Qwen2.5-1.5B	1.5B	67.7	32.3	58.4	31.7	61.6	30.4	79.3	90.5	76.4	75.1	64.2	76.2	64.9	59.7	42.6	60.7
Qwen3-1.7B-Base	1.7B	76.0	42.4	64.2	51.2	65.5	37.6	80.3	91.9	79.6	74.7	60.5	76.1	68.7	59.2	51.2	65.3
Qwen3.5-2B-Base	2B	69.8	35.8	43.6	32.3	64.0	37.3	83.7	93.1	85.6	73.1	63.5	76.5	69.1	58.8	60.4	63.1
Gemma-2-2B	2B	33.8	14.8	38.5	20.1	55.2	24.4	66.4	82.5	71.8	66.2	35.2	66.1	65.9	52.0	35.9	48.6
Gemma-3-1B-PT	1B	2.6	0.9	9.3	8.5	26.2	11.8	26.8	26.5	62.9	20.6	25.7	49.4	32.6	49.6	26.9	25.3
Gemma-4-E2B-Base	E2B	30.7	9.7	47.9	24.4	59.7	25.6	73.6	86.2	77.3	65.8	44.4	67.8	65.7	52.6	39.7	51.4
Llama-3.2-3B	3B	28.2	8.1	50.2	31.1	57.6	27.7	72.9	84.8	76.0	66.9	48.1	76.1	64.3	52.6	38.0	52.2
LFM2.5-1.2B-Base	1.2B	53.2	28.0	38.1	25.6	57.5	17.7	82.0	87.8	74.4	59.1	54.2	75.0	59.3	57.5	33.5	53.5
Falcon-H1-1.5B-Base	1.5B	74.7	42.6	61.1	28.1	65.3	36.4	81.0	90.3	78.8	71.3	62.2	75.7	67.3	57.8	45.4	62.5
Falcon-H1-Deep-1.5B-Base	1.5B	53.5	41.3	63.0	32.9	68.7	43.0	85.1	93.1	85.2	72.4	65.2	76.6	70.8	60.1	41.9	63.5
Open-Recipe Models																	
Instella-3B	3B	64.4	13.3	41.3	14.6	58.6	25.3	74.9	87.3	76.8	75.4	59.9	73.2	69.9	54.0	39.1	55.2
OLMoE-1B-7B-0125	A1B/7B	54.6	15.5	33.5	12.8	54.1	19.7	66.4	84.3	73.0	62.8	36.5	71.2	65.2	50.3	30.7	48.7
Yulan-Mini-2.4B	2.4B	69.0	27.2	62.3	50.0	51.7	23.7	65.1	82.5	78.5	66.2	59.9	76.2	63.3	62.4	44.1	58.8
SmolLM3-3B-Base	3B	81.7	56.1	60.3	37.8	62.2	39.0	79.3	90.0	84.8	72.7	70.2	81.1	70.6	64.3	37.5	65.9
MiniCPM5-1B-Base	1B	39.9	23.3	46.3	29.3	45.0	22.0	46.4	70.9	73.1	56.9	51.5	71.9	50.2	54.1	36.5	47.8
MobileLLM-R1-950M-base	950M	68.3	27.2	50.6	43.3	48.2	22.0	58.3	78.1	78.9	61.6	54.2	73.5	59.8	55.3	34.2	54.2
Ours																	
Puro-2B ($4.4K)	2B	56.0	27.9	49.8	20.7	54.8	24.0	66.8	84.7	77.3	67.7	63.6	75.8	62.9	59.9	41.1	55.5
Puro-2B	2B	59.7	30.3	52.9	31.1	57.4	25.8	74.2	86.1	76.9	67.8	63.2	75.5	63.6	60.6	42.1	57.8

learning from better data. Prior work proposes CMA to combine an ascending-quality curriculum with constant-LR training and model averaging (Luo et al., 2026). We extend this co-design approach in Phase 2. Rather than keeping LR constant throughout Phase 2 or changing the linear-decay slope, we first follow the scheduled LR decay in Section 2.2, then resume from a selected late checkpoint with a constant LR on the remaining curriculum data. This allows us to test different late-stage LR values without restarting Phase 2. We then equally average six late checkpoints from each constant-LR run to reduce training noise, and use validation performance to select the final model. Continuation and averaging settings are in Section K.3.

We further ablate the curriculum ordering, late-stage LR, and model averaging in Section D.4. The results show that curriculum ordering already improves over uniform ordering under the same LR decay. Model averaging alone gives little benefit along the decaying LR. It becomes more effective together with constant-LR training. These recipe variants are tabulated in Tables 19 and 20.

2.3.2 Dataset Benchmarking for Data Selection and Mixture Design

Besides data arrangement, we must decide which data to use and how to mix them. Since full-scale comparisons are too expensive, we use proxy experiments to evaluate sources and score ranges and guide the final mixture. We outline the approach in Section A and describe it in detail in Section L.

3 Evaluation

3.1 End-to-End Evaluation

Evaluation setup. We compare Puro-2B with pretrained or base *open-weight* and *open-recipe* language models, primarily in the 1B–3B parameter range. We evaluate all models on 15 benchmarks covering mathematics, code, and general reasoning and knowledge, and report their average as overall performance. Baseline models, benchmarks, and protocol are detailed in Section F.1.

Overall performance. As shown in Table 2, Puro-2B reaches an average score of 57.8 across the 15 benchmarks. It exceeds Qwen2-1.5B by 2.7 points, and the scaled $4.4K checkpoint already reaches 55.5, slightly above Qwen2-1.5B. Among evaluated open-recipe models, Puro-2B surpasses those with at most 2B parameters, remaining within 1.0 point of Yulan-Mini-2.4B. It also outperforms larger models, including the open-weight Llama-3.2-3B and the open-recipe Instella-3B. As shown in Figure 1, Puro-2B extends the cost–performance Pareto frontier of the compared models toward lower replication costs. The cost-accounting protocol and frontier construction are detailed in Section E.

3.2 Cost-Saving Ablation of Each Component

We estimate four cost factors separately, each using its own reference comparison. Their products should therefore be treated as illustrative transfers rather than measurements of a single sequential

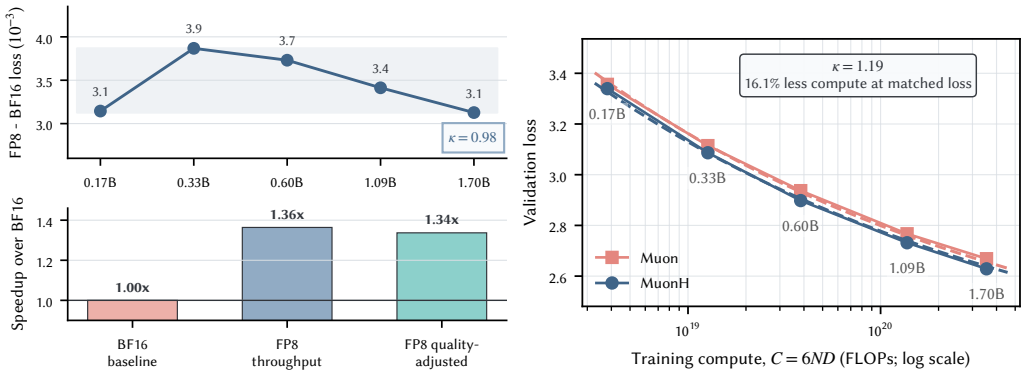


Figure 5: Cost-saving evidence for MuonH and blockwise FP8. **Left:** FP8 validation loss gap and the resulting quality-adjusted speedup relative to BF16. **Right:** Shared-shape scaling fits for MuonH and tuned Muon at fixed blockwise FP8. Full setups and fit diagnostics are given in Section G.

end-to-end speedup[3]. The data recipe is also ablated in Section L, but we report it separately because its effect cannot be cleanly summarized by a single speedup factor.

For the ladder evidence below, $C = 6ND$ denotes theoretical training compute, where N is the scaling parameter count and D is the number of training tokens. We define the compute-efficiency multiplier as $\kappa = C_{\text{baseline}}/C_{\text{variant}}$, where C_{baseline} and C_{variant} denote the theoretical training compute required by the baseline and the variant, respectively, to reach the same fitted validation loss.

RTX 5090 exhibits a high cost-performance ratio and utilization. Under the specifications and prices in Table 3, the RTX 5090 proxy provides $2.77\times$ the peak BF16 compute and $2.74\times$ the peak FP8 compute per unit price of H200. With training configuration tuning in Section D.1.3, our mixed-precision recipe can achieve 73% MFU, effectively utilizing RTX 5090 compute.

Blockwise FP8 still achieves a net speedup after accounting for precision-induced quality degradation. Since numerical precision affects both execution throughput and model quality, we define the net speedup as the product of the throughput speedup and the compute-efficiency multiplier κ. As shown in Figure 5, the shared-shape fit gives $\kappa = 0.98$ for blockwise FP8 relative to BF16. At 1.7B, the closest throughput proxy to the 2B production model, blockwise FP8 improves throughput by $1.36\times$. Combining this throughput gain with the fitted quality penalty yields a $1.34\times$ net speedup.

MuonH reaches matched validation loss with less compute than Muon. Using the scaling ladder designed in Figure 5 and § G, we compare the compute efficiency of the complete MuonH recipe and a separately tuned Muon baseline, both using blockwise FP8. The shared-shape fit estimates a compute-efficiency multiplier of $\kappa = 1.19\times$ for MuonH relative to Muon. Under this estimate, MuonH requires only $1/\kappa = 84\%$ of the theoretical compute needed by Muon to reach the same fitted validation loss.

The CMA recipe reaches matched performance at substantially lower cost than uniform training. To ablate the compute efficiency of CMA, we continue Phase 2 training with UD from the same Phase 1 checkpoint, as a baseline recipe. Using several budgets and their training results, we characterize the empirical cost-performance relationship, called the PURO Cost Scaling Law[4]. As shown in Figure 2b, this law provides both scaling trends for the basic recipe, and a reference performance level when the budget is limited and the compute is scaled down. Then, we compare the PURO-2B checkpoint with CMA against the scaling curve, and find that it achieves a $2.40\times$ cost-efficiency gain over uniform training.

[3]These comparisons are not a joint factorial ablation; the values are accounting estimates rather than measured end-to-end speedups. For example, MuonH and FP8 ablations use 20 tokens per parameter (TPP=20), while production training is overtrained (TPP=700). We use the compute-optimal setup due to limited compute.

[4]The fit uses $P = a + b\log_2(C - C_{\text{P1}})$, where P is the average performance over the 15 benchmarks reported in Table 2, C is the total reproduction cost, and $C_{\text{P1}} = \text{USD } 1.84\text{K}$ is the fixed Phase 1 cost.

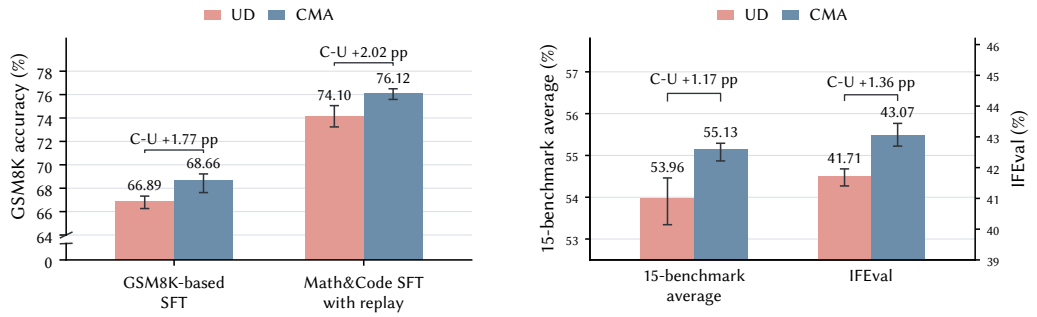


Figure 6: **The advantage of CMA pretraining persists after post-training. Left:** GSM8K accuracy after GSM8K-oriented SFT and Math&Code SFT with replay. **Right:** 15-task average and IFEval after Tulu-3 mixed-domain SFT.

3.3 Case Study: Post-Training Comparison between UD and CMA Pretraining

Optimal pretraining hyperparameters or design choices are not necessarily optimal, or even beneficial, after post-training (Qi et al., 2025). This motivates our question: *does a pretraining advantage persist after post-training?* Answering this requires controlled pretraining comparisons, which can be difficult with model weights alone and expensive even with an open recipe. Small-scale studies cost less, but limited base-model capabilities may restrict insights from post-training. Our cost-efficient recipe enables this study at the scale of Puro-2B. We perform a case study comparing the Uniform-Decay (UD) and Curriculum Model Averaging (CMA) recipes (Section D.4). Because CMA's base-model advantage may reflect its final-stage data distribution and diminish after post-training, we test whether it persists after both checkpoints undergo the same supervised fine-tuning (SFT) procedure.

Experimental setup. We compare the UD and CMA checkpoints under three SFT settings: GSM8K-based SFT, Math&Code SFT with replay, and Tulu-3 mixed-domain SFT. These settings span SFT data budgets and probe mathematical, general, and instruction-following capabilities. Within each setting, both checkpoints receive the same SFT procedure with matching training data, optimizer, learning-rate schedule, and budget. We repeat three times for each setup, and report the mean and observed minimum-maximum range. GSM8K-based and Math&Code models are evaluated on GSM8K with identical generation and final-answer extraction procedures. The Tulu-3 setting is evaluated on the 15-task benchmark suite and IFEval. Training and evaluation details are in Sections D.5 and H.

Results across SFT settings. (1) *GSM8K-based SFT.* We first test whether the CMA advantage persists after SFT on grade-school mathematics. UD and CMA achieve mean GSM8K accuracies of 66.89% and 68.66%, respectively. At the final checkpoints, CMA outperforms UD in all three paired runs by 1.36 to 2.12 percentage points (Figure 6 and table 12). (2) *Math&Code SFT with replay.* We enlarge the SFT mixture with more mathematics and code data, together with a small replay component from the final stage of the Phase 2 curriculum. This replay component aims to mitigate forgetting during SFT. The mean GSM8K accuracy is 74.10% for UD and 76.12% for CMA. CMA leads in all three paired endpoint runs, with gains ranging from 1.21 to 3.26 percentage points (Figure 6 and table 14). (3) *Tulu-3 mixed-domain SFT.* Across the three paired runs, CMA leads on both aggregate means. The 15-task macro-average is 53.96% for UD and 55.13% for CMA. On IFEval for instruction following, accuracy is 41.71% for UD and 43.07% for CMA. Both comparisons favor CMA in all three runs. CMA also performs better on 10 of the 15 component benchmarks (Figure 6 and table 16), showing that the advantage extends beyond mathematics to broader capabilities.

Taken together, CMA scores higher on the primary metrics in all three SFT settings. Consistency across the three data orders shows that the pretraining advantage persists after supervised adaptation.

4 Conclusion

In this work, we present a cost-efficient and reproducible pretraining recipe on consumer-grade RTX 5090 GPUs, with a \$4.4K run surpassing Qwen2-1.5B and our \$6.9K best checkpoint approaching Qwen2.5-1.5B. We further formulate the *Puro Cost Scaling Law* and show that billion-parameter, trillion-token pretraining can be practical on accessible hardware and modest budgets.

AI USE STATEMENT

In this work, we used generative AI tools to assist with translation. We did not use generative AI tools for other tasks requiring disclosure, and the remaining required-disclosure tasks are not applicable to this work. Additionally, we used generative AI tools to create or modify scientific figures or images and to edit the manuscript for readability. All AI-assisted text was reviewed by the authors for accuracy and consistency with the underlying research, and AI-assisted figures were checked against the underlying experimental results. We take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

We have released the following artifacts under the Apache 2.0 license unless otherwise noted separately. Upstream data terms remain component-specific. **We do not provide the HuggingFace links here due to double-blind review.** We describe in detail how to reproduce our work. As a distinctive feature, we select the widely accessible RTX 5090 as our training infrastructure. Moreover, Table 24 provides the evaluation results of the proxy experiments. Tables 5 and 6 provide the reference prices and data sources used in our cost accounting. Tables 4 and 23 list the data composition details of our pretraining recipe. Tables 11, 13 and 15 provide details of the data recipes used in our post-training setup. Section F.1 provides the training and evaluation setup in detail. Tables 19 and 20 provide the setup and evaluation results of the intermediate checkpoints for our recipe variants. For reference, we provide an anonymous Github link here: `https://anonymous.4open.science/r/Puro-Megatron-8D8E/`.

REFERENCES

aikitoria. Nvidia linux open gpu with p2p support. https://github.com/aikitoria/open-gpu-kernel-modules, 2026.

Loubna Ben Allal, Anton Lozhkov, Elie Bakouch, Gabriel Martín Blázquez, Guilherme Penedo, Lewis Tunstall, Andrés Marafioti, Hynek Kydlícek, Agustín Piqueres Lajarín, Vaibhav Srivastav, Joshua Lochner, Caleb Fahlgren, Xuan-Son Nguyen, Clémentine Fourrier, Ben Burtenshaw, Hugo Larcher, Haojun Zhao, Cyril Zakka, Mathieu Morlon, Colin Raffel, Leandro von Werra, and Thomas Wolf. Smollm2: When smol goes big - data-centric training of a small language model. *CoRR*, abs/2502.02737, 2025. doi: 10.48550/ARXIV.2502.02737. URL https://doi.org/10.48550/arXiv.2502.02737.

AMD. Instella-3B model card, 2025. URL https://huggingface.co/amd/Instella-3B. Accessed: 2026-08-20.

Michael Andersch, Greg Palmer, Ronny Krashinsky, Nick Stam, Vishal Mehta, Gonzalo Brito, and Sridhar Ramaswamy. Nvidia hopper architecture in-depth. https://developer.nvidia.com/blog/nvidia-hopper-architecture-in-depth/, 2022.

Alpay Ariyak, Junda Zhang, Junxiong Wang, Shang Zhu, Federico Bianchi, Sanjana Srivastava, Ashwinee Panda, Siddhant Bharti, Chenfeng Xu, John Heo, Xiaoxia Shirley Wu, James Zhou, Percy Liang, Leon Song, Ce Zhang, Ben Athiwaratkun, Zhongzhu Zhou, and Qingyang Wu. Coderforge-preview: Sota open dataset for training efficient agents, February 2026. URL https://www.together.ai/blog/coderforge-preview. Project core leads: Alpay Ariyak; Zhongzhu Zhou; Qingyang Wu.

Jacob Austin, Augustus Odena, Maxwell I. Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie J. Cai, Michael Terry, Quoc V. Le, and Charles Sutton. Program synthesis with large language models. *CoRR*, abs/2108.07732, 2021. URL https://arxiv.org/abs/2108.07732.

Elie Bakouch, Loubna Ben Allal, Anton Lozhkov, Nouamane Tazi, Lewis Tunstall, Carlos Miguel Patiño, Edward Beeching, Aymeric Roucher, Aksel Joonas Reedi, Quentin Gallouédec, Kashif Rasul, Nathan Habib, Clémentine Fourrier, Hynek Kydlicek, Guilherme Penedo, Hugo Larcher, Mathieu Morlon, Vaibhav Srivastav, Joshua Lochner, Xuan-Son Nguyen, Colin Raffel, Leandro von Werra, and Thomas Wolf. SmolLM3: smol, multilingual, long-context reasoner. https://huggingface.co/blog/smollm3, 2025.

Aarti Basant, Abhijit Khairnar, Abhijit Paithankar, Abhinav Khattar, Adithya Renduchintala, Aditya Malte, Akhiad Bercovich, Akshay Hazare, Alejandra Rico, Aleksander Ficek, Alex Kondratenko, Alex Shaposhnikov, Alexander Bukharin, Ali Taghibakhshi, Amelia Barton, Ameya Sunil Mahabaleshwarkar, Amy Shen, Andrew Tao, Ann Guan, Anna Shors, Anubhav Mandarwal, Arham Mehta, Arun Venkatesan, Ashton Sharabiani, Ashwath Aithal, Ashwin Poojary, Ayush Dattagupta, Balaram Buddharaju, Banghua Zhu, Barnaby Simkin, Bilal Kartal, Bita Darvish Rouhani, Bobby Chen, Boris Ginsburg, Brandon Norick, Brian Yu, Bryan Catanzaro, Charles Wang, Charlie Truong, Chetan Mungekar, Chintan Patel, Chris Alexiuk, Christian Munley, Christopher Parisien, Dan Su, Daniel Afrimi, Daniel Korzekwa, Daniel Rohrer, Daria Gitman, David Mosallanezhad, Deepak Narayanan, Dima Rekesh, Dina Yared, Dmytro Pykhtar, Dong Ahn, Duncan Riach, Eileen Long, Elliott Ning, Eric Chung, Erick Galinkin, Evelina Bakhturina, Gargi Prasad, Gerald Shen, Haifeng Qian, Haim Elisha, Harsh Sharma, Hayley Ross, Helen Ngo, Herman Sahota, Hexin Wang, Hoo Chang Shin, Hua Huang, Iain Cunningham, Igor Gitman, Ivan Moshkov, Jaehun Jung, Jan Kautz, Jane Polak Scowcroft, Jared Casper, Jian Zhang, Jiaqi Zeng, Jimmy Zhang, Jinze Xue, Jocelyn Huang, Joey Conway, John Kamalu, Jonathan M. Cohen, Joseph Jennings, Julien Veron Vialard, Junkeun Yi, Jupinder Parmar, Kari Briski, Katherine Cheung, Katherine Luna, Keith W. Ross, Keshav Santhanam, Kezhi Kong, Krzysztof Pawelec, and Kumar Anik. NVIDIA nemotron nano 2: An accurate and efficient hybrid mamba-transformer reasoning model. *CoRR*, abs/2508.14444, 2025. doi: 10.48550/ARXIV.2508.14444. URL https://doi.org/10.48550/arXiv.2508.14444.

Loubna Ben Allal, Anton Lozhkov, Guilherme Penedo, Thomas Wolf, and Leandro von Werra. Smollm-corpus, July 2024. URL https://huggingface.co/datasets/HuggingFaceTB/smollm-corpus.

Stella Biderman, Hailey Schoelkopf, Quentin Gregory Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, USVSN Sai Prashanth, Edward Raff, et al. Pythia: A suite for analyzing large language models across training and scaling. In *International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 2397–2430. PMLR, 2023. URL https://proceedings.mlr.press/v202/biderman23a.html.

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. PIQA: reasoning about physical commonsense in natural language. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pp. 7432–7439. AAAI Press, 2020. doi: 10.1609/AAAI.V34I05.6239. URL https://doi.org/10.1609/aaai.v34i05.6239.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Pondé de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Joshua Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code. *CoRR*, abs/2107.03374, 2021. URL https://arxiv.org/abs/2107.03374.

Yukang Chen, Shaozuo Yu, Shengju Qian, Haotian Tang, Xin Lai, Zhijian Liu, Song Han, and Jiaya Jia. Long alpaca: Long-context instruction-following models. https://github.com/dvlab-research/LongLoRA, 2023.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. BoolQ: Exploring the surprising difficulty of natural yes/no questions. In Jill Burstein, Christy Doran, and Thamar Solorio (eds.), *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 2924–2936, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1300. URL https://aclanthology.org/N19-1300/.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try ARC, the AI2 reasoning challenge. *CoRR*, abs/1803.05457, 2018. URL http://arxiv.org/abs/1803.05457.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *CoRR*, abs/2110.14168, 2021. URL https://arxiv.org/abs/2110.14168.

Baptiste Colle, Hanna Yukhymenko, and Leandro von Werra. Jupyter agent dataset, 2025.

Together Computer. Redpajama: An open source recipe to reproduce llama training dataset, April 2023. URL https://github.com/togethercomputer/RedPajama-Data.

OpenCompass Contributors. Opencompass: A universal evaluation platform for foundation models. https://github.com/open-compass/opencompass, 2023.

NVIDIA Corporation. Nvidia h200 tensor core gpu. https://resources.nvidia.com/en-us-hopper-architecture/hpc-datasheet-sc23, 2024.

NVIDIA Corporation. Nvidia rtx blackwell gpu architecture. https://images.nvidia.com/aem-dam/Solutions/geforce/blackwell/nvidia-rtx-blackwell-gpu-architecture.pdf, 2025a.

NVIDIA Corporation. Nvidia rtx pro blackwell gpu architecture. https://www.nvidia.com/content/dam/en-zz/Solutions/design-visualization/quadro-product-literature/NVIDIA-RTX-Blackwell-PRO-GPU-Architecture-v1.0.pdf, 2025b.

Yiming Cui, Ziqing Yang, and Xin Yao. Efficient and effective text encoding for chinese llama and alpaca. *arXiv preprint arXiv:2304.08177*, 2023. URL https://arxiv.org/abs/2304.08177.

DeepSeek-AI, Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Haowei Zhang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Li, Hui Qu, J. L. Cai, Jian Liang, Jianzhong Guo, Jiaqi Ni, Jiashi Li, Jiawei Wang, Jin Chen, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, Junxiao Song, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Lei Xu, Leyi Xia, Liang Zhao, Litong Wang, Liyue Zhang, Meng Li, Miaojun Wang, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingming Li, Ning Tian, Panpan Huang, Peiyi Wang, Peng Zhang, Qiancheng Wang, Qihao Zhu, Qinyu Chen, Qiushi Du, R. J. Chen, R. L. Jin, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, Runxin Xu, Ruoyu Zhang, Ruyi Chen, S. S. Li, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shaoqing Wu, Shengfeng Ye, Shirong Ma, Shiyu Wang, Shuang Zhou, Shuiping Yu, Shunfeng Zhou, Shuting Pan, T. Wang, Tao Yun, Tian Pei, Tianyu Sun, W. L. Xiao, and Wangding Zeng. DeepSeek-V3 technical report. *CoRR*, abs/2412.19437, 2024. doi: 10.48550/ARXIV.2412.19437. URL https://doi.org/10.48550/arXiv.2412.19437.

DeepSeek-AI, Anyi Xu, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, Chenchen Ling, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chengyu Hou, Chenhao Xu, Chenze Shao, Chong Ruan, Conner Sun, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Donghao Li, Dongjie Ji, Erhang Li, Fang Wei, Fangyun Lin, Fangzhou Yuan, Feiyu Xia, Fucong Dai, Guangbo Hao, Guanting Chen, Guoai Cao, Guolai Meng, Guowei Li, Han Yu, Han Zhang, Hanwei Xu, Hao Li, Haofen Liang, Haoling Zhang, Haoming Luo, Haoran Wei, Haotian Yuan, Haowei Zhang, Haowen Luo, Haoyu Chen, Haozhe Ji, Hengqing Zhang, Honghui Ding, Hongxuan Tang, Huanqi Cao, Huazuo Gao, Hui Qu, Hui Zeng, J Yang, JQ Zhu, Jia Luo, Jia Song, Jia Yu, Jialiang Huang, Jialu Cai, Jian Liang, Jiangting Zhou, Jiasheng Ye, Jiashi Li, Jiaxin Xu, Jiewen Hu, Jieyu Yang, Jin Chen, Jin Yan, Jingchang Chen, Jingli Zhou, Jingting Xiang, Jingyang Yuan, Jingyuan Cheng, Jingzi Zhou, Jinhua Zhu, Jiping Yu, Joseph Sun, Jun Ran, Junguang Jiang, Junjie Qiu, Junlong Li, Junmin Zheng, Junxiao Song, Kai Dong, Kaige Gao, Kang Guan, Kexing Zhou, Kezhao Huang, Kuai Yu, Lean Wang, Lecong Zhang, Lei Wang, Leyi Xia, Li Zhang, Liang Zhao, Lihua Guo, Lingxiao Luo, Linwang Ma, Linyan Zhu, Litong Wang, Liyu Cai, Liyue Zhang, Longhao Chen, MS Di, MY Xu, Max Mei, Miaojun Wang, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingming Li, Mingxu Zhou, Minmin Han, Ning Wang, Panpan Huang, Panpan Wang, Peixin Cong, Peiyi Wang, Peng Zhang, Qiancheng Wang, Qihao Zhu, Qingyang Li, Qinyu Chen, Qiushi Du, Qiwei Jiang, Rui Tian, Ruifan Xu, Ruijie Lu, Ruiling Xu, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, Runqian Chen, Runqiu Yin, Runxin Xu, Ruomeng Shen, Ruoyu Zhang, Ruyi Chen, SH Liu, Shanghao Lu, Shangmian Sun, Shangyan Zhou, Shanhuang Chen, Shaofei Cai, Shaoheng Nie, Shaoqing Wu, Shaoyuan Chen, Shengding Hu, Shengyu Liu, Shiqiang Hu, Shirong Ma, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, Shuying Yu, Songyang Zhou, Tao Ni, Tao Yun, Tian Jin, Tian Pei, Tian Ye, Tianle Lin, Tianran Ji, Tianyi Cui, Tianyuan Yue, Tingting Yu, Tun Wang, W Zhang, WL Xiao, Wangding Zeng, Wei An, Weilin Zhao, Wen Liu, Wenfeng Liang, Wenjie Pang, Wenjing Luo, Wenjing Yao, Wenjun Gao, Wenkai Yang, Wenlve Huang, Wenqing Hou, Wentao Zhang, Wenting Ma, Xi Gao, Xiang He, Xiangwen Wang, Xianzu Wang, Xiao Bi, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaokang Zhang, Xiaotao Nie, Xiaowen Sun, Xiaoxiang Wang, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xingchen Liu, Xingkai Yu, Xingyou Li, Xinyu Yang, Xinyu Zhang, Xu Chen, Xuanyu Wang, Xuecheng Su, Xueyin Chen, Xuheng Lin, Xuwei Fu, YC Yan, YQ Wang, YW Ma, Yanfeng Luo, Yang Zhang, Yanhong Xu, Yanru Ma, Yanwen Huang, Yao Li, Yao Li, Yao Xu, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Qian, Yi Shao, Yi Yu, Yichao Zhang, Yifan Ding, Yifan Shi, Yijia Wu, Yiliang Xiong, Yiling Ma, Ying He, Ying Tang, Ying Zhou, Yingjia Luo, Yinmin Zhong, Yishi Piao, Yisong Wang, Yixiang Zhang, Yixiao Chen, Yixuan Tan, Yixuan Wei, Yiyang

Ma, Yiyuan Liu, Yonglun Yang, Yongqiang Guo, Yongtong Wu, Yu Wu, YuKun Li, Yuan Cheng, Yuan Ou, Yuanfan Xu, Yuanhao Li, Yuduan Wang, Yuehan Yang, Yuer Xu, Yuhan Wu, Yuhao Meng, Yuheng Zou, Yukun Zha, Yunfan Xiong, Yupeng Chen, Yuping Lin, Yuqian Cao, Yuqian Wang, Yushun Zhang, Yuting Yan, Yutong Lin, Yuxian Gu, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuxuan Zhou, Yuyang Zhou, Yuzhen Huang, ZF Wu, Zehao Wang, Zehua Zhao, Zehui Ren, Zekai Zhang, Zhangli Sha, Zhe Fu, Zhe Ju, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zheren Gao, Zhewen Hao, Zhibin Gou, Zhicheng Ma, Zhigang Yan, Zhihong Shao, Zhixian Huang, Zhixuan Chen, Zhiyu Wu, Zhizhou Ren, Zhongyu Wu, Zhuoshu Li, Zhuping Zhang, Zian Xu, Zihao Wang, Zihua Qu, Zihui Gu, Zijia Zhu, Zilin Li, Zipeng Zhang, Ziwei Xie, Ziyi Gao, Ziyi Wan, Zizheng Pan, and Zongqing Yao. Deepseek-v4: Towards highly efficient million-token context intelligence, 2026. URL https://arxiv.org/abs/2606.19348.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. QLoRA: Efficient finetuning of quantized LLMs. In *Advances in Neural Information Processing Systems*, volume 36, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/1feb87871436031bdc0f2beaa62a049b-Abstract-Conference.html.

Shizhe Diao, Yu Yang, Yonggan Fu, Xin Dong, Dan SU, Markus Kliegl, ZIJIA CHEN, Peter Belcak, Yoshi Suhara, Hongxu Yin, Mostofa Patwary, Yingyan (Celine) Lin, Jan Kautz, and Pavlo Molchanov. Nemotron-climb: Clustering-based iterative data mixture bootstrapping for language model pre-training. In D. Belgrave, C. Zhang, H. Lin, L. Montoya, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38. Curran Associates, Inc., 2025. doi: 10.52202/085713-0865. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/24a8968affe71ffe4067d022b9d16566-Paper-Datasets_and_Benchmarks_Track.pdf.

Kazuki Fujii, Yukito Tajima, Sakae Mizuki, Hinari Shimada, Taihei Shiotani, Koshiro Saito, Masanari Ohi, Masaki Kawamura, Taishi Nakamura, Takumi Okamoto, Shigeki Ishida, Kakeru Hattori, Youmi Ma, Hiroya Takamura, Rio Yokota, and Naoaki Okazaki. Rewriting pre-training data boosts llm performance in math and code, 2025. URL https://arxiv.org/abs/2505.02881.

Gemma Team. Gemma 3 technical report. *CoRR*, abs/2503.19786, 2025. doi: 10.48550/ARXIV.2503.19786. URL https://doi.org/10.48550/arXiv.2503.19786.

Gemma Team. Gemma 4 technical report. *CoRR*, abs/2607.02770, 2026. doi: 10.48550/ARXIV.2607.02770. URL https://doi.org/10.48550/arXiv.2607.02770.

Google. Gemma 2 model card, 2024. URL https://ai.google.dev/gemma/docs/core/model_card_2. Accessed: 2026-08-20.

Google. Gemma 3 model card, 2025. URL https://ai.google.dev/gemma/docs/core/model_card_3. Accessed: 2026-08-20.

Dirk Groeneveld, Iz Beltagy, Evan Walsh, Akshita Bhagia, Rodney Kinney, Oyvind Tafjord, Ananya Jha, Hamish Ivison, Ian Magnusson, Yizhong Wang, Shane Arora, David Atkinson, Russell Authur, Khyathi Chandu, Arman Cohan, Jennifer Dumas, Yanai Elazar, Yuling Gu, Jack Hessel, Tushar Khot, William Merrill, Jacob Morrison, Niklas Muennighoff, Aakanksha Naik, Crystal Nam, Matthew Peters, Valentina Pyatkin, Abhilasha Ravichander, Dustin Schwenk, Saurabh Shah, William Smith, Emma Strubell, Nishant Subramani, Mitchell Wortsman, Pradeep Dasigi, Nathan Lambert, Kyle Richardson, Luke Zettlemoyer, Jesse Dodge, Kyle Lo, Luca Soldaini, Noah Smith, and Hannaneh Hajishirzi. OLMo: Accelerating the science of language models. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 15789–15809, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.841. URL https://aclanthology.org/2024.acl-long.841/.

Yuling Gu, Oyvind Tafjord, Bailey Kuehl, Dany Haddad, Jesse Dodge, and Hannaneh Hajishirzi. OLMES: A standard for language model evaluations. In Luis Chiruzzo, Alan Ritter, and Lu Wang (eds.), *Findings of the Association for Computational Linguistics: NAACL 2025, Albuquerque, New Mexico, USA, April 29 - May 4, 2025*, pp. 5020–5048. Association for Computational Linguistics, 2025. doi: 10.18653/V1/2025.FINDINGS-NAACL.282. URL https://doi.org/10.18653/v1/2025.findings-naacl.282.

David Hall, Ahmed Ahmed, Christopher Chou, Abhinav Garg, Rohith Kuditipudi, Will Held, Nikil Ravi, Herumb Shandilya, Jason Wang, Jason Bolton, Siddharth Karamcheti, Suhas Kotha, Tony Lee, Nelson Liu, Joel Niklaus, Ashwin Ramaswami, Kamyar Salahi, Kaiyue Wen, Chi Heem Wong, Sherry Yang, Ivan Zhou, and Percy Liang. Introducing Marin: An open lab for building foundation models. Marin Community Blog, May 2025. URL https://marin.community/blog/2025/05/19/announcement/. Blog post.

Conghui He, Zhenjiang Jin, Chao Xu, Jiantao Qiu, Bin Wang, Wei Li, Hang Yan, Jiaqi Wang, and Dahua Lin. Wanjuan: A comprehensive multimodal dataset for advancing english and chinese large models, 2023. URL https://arxiv.org/abs/2308.10755.

Conghui He, Wei Li, Zhenjiang Jin, Chao Xu, Bin Wang, and Dahua Lin. Opendatalab: Empowering general artificial intelligence with open datasets, 2024. URL https://arxiv.org/abs/2407.13773.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021a. URL https://openreview.net/forum?id=d7KBjmI3GmQ.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the MATH dataset. In Joaquin Vanschoren and Sai-Kit Yeung (eds.), *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks 2021, December 2021, virtual*, 2021b. URL https://datasets-benchmarks-proceedings.neurips.cc/paper/2021/hash/be83ab3ecd0db773eb2dc1b0a17836a1-Abstract-round2.html.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Thomas Hennigan, Eric Noland, Katherine Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karén Simonyan, Erich Elsen, Oriol Vinyals, Jack Rae, and Laurent Sifre. An empirical analysis of compute-optimal large language model training. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 30016–30030. Curran Associates, Inc., 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/c1e2faff6f588870935f114ebe04a3e5-Paper-Conference.pdf.

Shengding Hu, Yuge Tu, Xu Han, Ganqu Cui, Chaoqun He, Weilin Zhao, Xiang Long, Zhi Zheng, Yewei Fang, Yuxiang Huang, Xinrong Zhang, Zhen Leng Thai, Chongyi Wang, Yuan Yao, Chenyang Zhao, Jie Zhou, Jie Cai, Zhongwu Zhai, Ning Ding, Chao Jia, Guoyang Zeng, dahai li, Zhiyuan Liu, and Maosong Sun. MiniCPM: Unveiling the potential of small language models with scalable training strategies. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=3X2L2TFr0f.

Keller Jordan, Yuchen Jin, Vlado Boza, Jiacheng You, Franz Cesista, Laker Newhouse, and Jeremy Bernstein. Muon: An optimizer for hidden layers in neural networks, 2024. URL https://kellerjordan.github.io/posts/muon/.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *CoRR*, abs/2001.08361, 2020. URL https://arxiv.org/abs/2001.08361.

Vijay Anand Korthikanti, Jared Casper, Sangkug Lym, Lawrence McAfee, Michael Andersch, Mohammad Shoeybi, and Bryan Catanzaro. Reducing activation recomputation in large transformer models. In Dawn Song, Michael Carbin, and Tianqi Chen (eds.), *Proceedings of the Sixth Conference on Machine Learning and Systems, MLSys 2023, Miami, FL, USA, June 4-8, 2023*. mlsys.org, 2023. URL https://proceedings.mlsys.org/paper_files/paper/2023/hash/80083951326cf5b35e5100260d64ed81-Abstract-mlsys2023.html.

Nathan Lambert, Jacob Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison, Faeze Brahman, Lester James V. Miranda, Alisa Liu, Nouha Dziri, Shane Lyu, Yuling Gu, Saumya Malik,

Victoria Graf, Jena D. Hwang, Jiangjiang Yang, Ronan Le Bras, Oyvind Tafjord, Chris Wilhelm, Luca Soldaini, Noah A. Smith, Yizhong Wang, Pradeep Dasigi, and Hannaneh Hajishirzi. Tülu 3: Pushing frontiers in open language model post-training. *CoRR*, abs/2411.15124, 2024. doi: 10.48550/arXiv.2411.15124. URL https://arxiv.org/abs/2411.15124.

Haoyang Li, Zhanchao Xu, Yiming Li, Xuejia Chen, Darian Li, Anxin Tian, Qingfa Xiao, Cheng Deng, Jun Wang, Qing Li, Lei Chen, and Mingxuan Yuan. Loopserve: An adaptive dual-phase llm inference acceleration system for multi-turn dialogues, 2025. URL https://arxiv.org/abs/2507.13681.

Jeffrey Li, Alex Fang, Georgios Smyrnis, Maor Ivgi, Matt Jordan, Samir Yitzhak Gadre, Hritik Bansal, Etash Kumar Guha, Sedrick Scott Keh, Kushal Arora, Saurabh Garg, Rui Xin, Niklas Muennighoff, Reinhard Heckel, Jean Mercat, Mayee F. Chen, Suchin Gururangan, Mitchell Wortsman, Alon Albalak, Yonatan Bitton, Marianna Nezhurina, Amro Abbas, Cheng-Yu Hsieh, Dhruba Ghosh, Josh Gardner, Maciej Kilian, Hanlin Zhang, Rulin Shao, Sarah M. Pratt, Sunny Sanyal, Gabriel Ilharco, Giannis Daras, Kalyani Marathe, Aaron Gokaslan, Jieyu Zhang, Khyathi Raghavi Chandu, Thao Nguyen, Igor Vasiljevic, Sham M. Kakade, Shuran Song, Sujay Sanghavi, Fartash Faghri, Sewoong Oh, Luke Zettlemoyer, Kyle Lo, Alaaeldin El-Nouby, Hadi Pouransari, Alexander Toshev, Stephanie Wang, Dirk Groeneveld, Luca Soldaini, Pang Wei Koh, Jenia Jitsev, Thomas Kollar, Alex Dimakis, Yair Carmon, Achal Dave, Ludwig Schmidt, and Vaishaal Shankar. DataComp-LM: In search of the next generation of training sets for language models. In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (eds.), *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024. URL http://papers.nips.cc/paper%5Ffiles/paper/2024/hash/19e4ea30dded58259665db375885e412-Abstract-Datasets%5Fand%5FBenchmarks%5FTrack.html.

Jia LI, Edward Beeching, Lewis Tunstall, Ben Lipkin, Roman Soletskyi, Shengyi Costa Huang, Kashif Rasul, Longhui Yu, Albert Jiang, Ziju Shen, Zihan Qin, Bin Dong, Li Zhou, Yann Fleureau, Guillaume Lample, and Stanislas Polu. Numinamath. [https://huggingface.co/AI-MO/NuminaMath-CoT](https://github.com/project-numina/aimo-progress-prize/blob/main/report/numina_dataset.pdf), 2024.

Liquid AI. LFM2 technical report. *CoRR*, abs/2511.23404, 2025. doi: 10.48550/ARXIV.2511.23404. URL https://doi.org/10.48550/arXiv.2511.23404.

Liquid AI. Introducing LFM2.5: The next generation of on-device AI, 2026. URL https://www.liquid.ai/blog/introducing-lfm2-5-the-next-generation-of-on-device-ai. Accessed: 2026-08-20.

Jiang Liu, Jialian Wu, Xiaodong Yu, Yusheng Su, Prakamya Mishra, Gowtham Ramesh, Sudhanshu Ranjan, Chaitanya Manem, Ximeng Sun, Ze Wang, Pratik Prabhanjan Brahma, Zicheng Liu, and Emad Barsoum. Instella: Fully open language models with stellar performance. *CoRR*, abs/2511.10628, 2025. doi: 10.48550/ARXIV.2511.10628. URL https://doi.org/10.48550/arXiv.2511.10628.

Shayne Longpre, Le Hou, Tu Vu, Albert Webson, Hyung Won Chung, Yi Tay, Denny Zhou, Quoc V. Le, Barret Zoph, Jason Wei, and Adam Roberts. The flan collection: Designing data and methods for effective instruction tuning, 2023. URL https://arxiv.org/abs/2301.13688.

Ilya Loshchilov and Frank Hutter. SGDR: stochastic gradient descent with warm restarts. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL https://openreview.net/forum?id=Skq89Scxx.

Anton Lozhkov, Raymond Li, Loubna Ben Allal, Federico Cassano, Joel Lamy-Poirier, Nouamane Tazi, Ao Tang, Dmytro Pykhtar, Jiawei Liu, Yuxiang Wei, Tianyang Liu, Max Tian, Denis Kocetkov, Arthur Zucker, Younes Belkada, Zijian Wang, Qian Liu, Dmitry Abulkhanov, Indraneil Paul, Zhuang Li, Wen-Ding Li, Megan Risdal, Jia Li, Jian Zhu, Terry Yue Zhuo, Evgenii Zheltonozhskii, Nii Osae Osae Dade, Wenhao Yu, Lucas Krauß, Naman Jain, Yixuan Su,

Xuanli He, Manan Dey, Edoardo Abati, Yekun Chai, Niklas Muennighoff, Xiangru Tang, Muhtasham Oblokulov, Christopher Akiki, Marc Marone, Chenghao Mou, Mayank Mishra, Alex Gu, Binyuan Hui, Tri Dao, Armel Zebaze, Olivier Dehaene, Nicolas Patry, Canwen Xu, Julian McAuley, Han Hu, Torsten Scholak, Sebastien Paquet, Jennifer Robinson, Carolyn Jane Anderson, Nicolas Chapados, Mostofa Patwary, Nima Tajbakhsh, Yacine Jernite, Carlos Mu noz Ferrandis, Lingming Zhang, Sean Hughes, Thomas Wolf, Arjun Guha, Leandro von Werra, and Harm de Vries. Starcoder 2 and the stack v2: The next generation, 2024. URL https://arxiv.org/abs/2402.19173.

Kairong Luo, Zhenbo Sun, Xinyu Shi, Shengqi Chen, Bowen Yu, Yunyi Chen, Chenyi Dang, Hengtao Tao, Hui Wang, Fangming Liu, Kaifeng Lyu, and Wenguang Chen. Pcmind-2.1-kaiyuan-2b technical report, 2025a. URL https://arxiv.org/abs/2512.07612.

Kairong Luo, Haodong Wen, Shengding Hu, Zhenbo Sun, Zhiyuan Liu, Maosong Sun, Kaifeng Lyu, and Wenguang Chen. A Multi-Power law for loss curve prediction across learning rate schedules. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025b. URL https://openreview.net/forum?id=KnoS9XxIlK.

Kairong Luo, Zhenbo Sun, Haodong Wen, Xinyu Shi, Jiarui Cui, Chenyi Dang, Kaifeng Lyu, and Wenguang Chen. How learning rate decay wastes your best data in curriculum-based LLM pretraining. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=T5wkZJqzkz.

Rabeeh Karimi Mahabadi, Sanjeev Satheesh, Shrimai Prabhumoye, Mostofa Patwary, Mohammad Shoeybi, and Bryan Catanzaro. Nemotron-cc-math: A 133 billion-token-scale high quality math pretraining dataset. *arXiv preprint arXiv:2508.15096*, 2025.

Meta AI. MobileLLM: Optimizing sub-billion parameter language models for on-device use cases, 2024a. URL https://github.com/facebookresearch/MobileLLM. Accessed: 2026-08-20.

Meta AI. Llama 3.2: Revolutionizing edge ai and vision with open, customizable models, 2024b. URL https://ai.meta.com/blog/llama-3-2-connect-2024-vision-edge-mobile-devices/.

Meta AI. Llama 3.2 model card, 2024c. URL https://github.com/meta-llama/llama-models/blob/main/models/llama3_2/MODEL_CARD.md. Accessed: 2026-08-20.

Meta AI. MobileLLM-R1 official recipe, 2025. URL https://github.com/facebookresearch/MobileLLM-R1. Accessed: 2026-08-20.

Paulius Micikevicius, Dusan Stosic, Neil Burgess, Marius Cornea, Pradeep Dubey, Richard Grisenthwaite, Sangwon Ha, Alexander Heinecke, Patrick Judd, John Kamalu, Naveen Mellempudi, Stuart Oberman, Mohammad Shoeybi, Michael Siu, and Hao Wu. FP8 formats for deep learning. *CoRR*, abs/2209.05433, 2022. doi: 10.48550/ARXIV.2209.05433. URL https://doi.org/10.48550/arXiv.2209.05433.

Niklas Muennighoff, Luca Soldaini, Dirk Groeneveld, Kyle Lo, Jacob Morrison, Sewon Min, Weijia Shi, Evan Pete Walsh, Oyvind Tafjord, Nathan Lambert, Yuling Gu, Shane Arora, Akshita Bhagia, Dustin Schwenk, David Wadden, Alexander Wettig, Binyuan Hui, Tim Dettmers, Douwe Kiela, Ali Farhadi, and et al. Olmoe: Open mixture-of-experts language models. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=xXTkbTBmqq.

Subhabrata Mukherjee, Arindam Mitra, Ganesh Jawahar, Sahaj Agarwal, Hamid Palangi, and Ahmed Awadallah. Orca: Progressive learning from complex explanation traces of gpt-4, 2023. URL https://arxiv.org/abs/2306.02707.

Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay 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*, pp. 1–15. ACM, November 2021. doi: 10.1145/3458817.3476209. URL https://doi.org/10.1145/3458817.3476209.

NVIDIA. Nvidia data agreement for model training. https://huggingface.co/datasets/nvidia/Nemotron-Pretraining-Dataset-sample/blob/main/LICENSE.md, 2025. [Accessed 03-12-2025].

NVIDIA Corporation. Megatron core, February 2026a. URL https://github.com/NVIDIA/Megatron-LM/releases/tag/core_v0.16.0. Official release published 2026-02-26; Git tag core_v0.16.0, commit 3bec9aa97dda898d16ff5a89bac0ed2b6682b172.

NVIDIA Corporation. Using FP8 and FP4 with transformer engine, 2026b. URL https://docs.nvidia.com/deeplearning/transformer-engine/user-guide/examples/fp8_primer.html. Transformer Engine 2.17.0 documentation, accessed 2026-08-09.

Beijing Academy of Artificial Intelligence. Chinese corpus internet usage agreement. https://data.baai.ac.cn/resources/agreement/cci_usage_aggrement.pdf, 2023. [Accessed 03-12-2025].

OpenBMB Team. MiniCPM5-1B-Base model card, 2026. URL https://huggingface.co/openbmb/MiniCPM5-1B-Base.

Andrei Panferov, Erik Schultheis, Soroush Tabesh, and Dan Alistarh. Quartet II: Accurate LLM pretraining in NVFP4 by improved unbiased gradient estimation. *CoRR*, abs/2601.22813, 2026. doi: 10.48550/ARXIV.2601.22813. URL https://arxiv.org/abs/2601.22813.

Keiran Paster, Marco Dos Santos, Zhangir Azerbayev, and Jimmy Ba. Openwebmath: An open dataset of high-quality mathematical web text, 2023. URL https://arxiv.org/abs/2310.06786.

Guilherme Penedo. Finewiki, 2025. URL https://huggingface.co/datasets/HuggingFaceFW/finewiki. Source: Wikimedia Enterprise Snapshot API (https://api.enterprise.wikimedia.com/v2/snapshots). Text licensed under CC BY-SA 4.0 with attribution to Wikipedia contributors.

Guilherme Penedo, Anton Lozhkov, Hynek Kydlíček, Loubna Ben Allal, Edward Beeching, Agustín Piqueres Lajarín, Quentin Gallouédec, Nathan Habib, Lewis Tunstall, and Leandro von Werra. Codeforces cots. https://huggingface.co/datasets/open-r1/codeforces-cots, 2025.

Renjie Pi, Grace Lam, Mohammad Shoeybi, Pooya Jannaty, Bryan Catanzaro, and Wei Ping. On data engineering for scaling llm terminal capabilities, 2026. URL https://arxiv.org/abs/2602.21193.

Zhenting Qi, Fan Nie, Alexandre Alahi, James Y. Zou, Himabindu Lakkaraju, Yilun Du, Eric P. Xing, Sham M. Kakade, and Hanlin Zhang. Evolm: In search of lost training dynamics for language model reasoning. In Danielle Belgrave, Cheng Zhang, Laura N. Montoya, Hsuan-Tien Lin, Razvan Pascanu, Piotr Koniusz, Marzyeh Ghassemi, Nancy Chen, Iván Vladimir Meza Ruíz, and Arturo Loaiza-Bonilla (eds.), *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2025, NeurIPS 2025, San Diego, CA, USA, December 2-7, 2025 / Mexico City, Mexico, November 30 - December 5, 2025*, 2025. URL http://papers.nips.cc/paper_files/paper/2025/hash/778cc31583604ebf246726055e9f737f-Abstract-Conference.html.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. Toolllm: Facilitating large language models to master 16000+ real-world apis, 2023.

Qwen Team. Qwen3.5: Towards native multimodal agents, February 2026. URL https://qwen.ai/blog?id=qwen3.5.

Morgane Rivière, Shreya Pathak, Pier Giuseppe Sessa, Cassidy Hardin, Surya Bhupatiraju, Léonard Hussenot, Thomas Mesnard, Bobak Shahriari, Alexandre Ramé, Johan Ferret, Peter Liu, Pouya Tafti, Abe Friesen, Michelle Casbon, Sabela Ramos, Ravin Kumar, Charline Le Lan, Sammy Jerome, Anton Tsitsulin, Nino Vieillard, Piotr Stanczyk, Sertan Girgin, Nikola Momchev, Matt Hoffman, Shantanu Thakoor, Jean-Bastien Grill, Behnam Neyshabur, Olivier Bachem, Alanna Walton, Aliaksei Severyn, Alicia Parrish, Aliya Ahmad, Allen Hutchison, Alvin Abdagic, Amanda Carl, Amy Shen, Andy Brock, Andy Coenen, Anthony Laforge, Antonia Paterson, Ben Bastian, Bilal Piot, Bo Wu, Brandon Royal, Charlie Chen, Chintu Kumar, Chris Perry, Chris Welty, Christopher A. Choquette-Choo, Danila Sinopalnikov, David Weinberger, Dimple Vijaykumar, Dominika Rogozinska, Dustin Herbison, Elisa Bandy, Emma Wang, Eric Noland, Erica Moreira, Evan Senter, Evgenii Eltyshev, Francesco Visin, Gabriel Rasskin, Gary Wei, Glenn Cameron, Gus Martins, Hadi Hashemi, Hanna Klimczak-Plucinska, Harleen Batra, Harsh Dhand, Ivan Nardini, Jacinda Mein, Jack Zhou, James Svensson, Jeff Stanway, Jetha Chan, Jin Peng Zhou, Joana Carrasqueira, Joana Iljazi, Jocelyn Becker, Joe Fernandez, Joost van Amersfoort, Josh Gordon, Josh Lipschultz, Josh Newlan, Ju-yeong Ji, Kareem Mohamed, Kartikeya Badola, Kat Black, Katie Millican, Keelin McDonell, Kelvin Nguyen, Kiranbir Sodhia, Kish Greene, Lars Lowe Sjösund, Lauren Usui, Laurent Sifre, Lena Heuermann, Leticia Lago, and Lilly McNealus. Gemma 2: Improving open language models at a practical size. *CoRR*, abs/2408.00118, 2024. doi: 10.48550/ARXIV.2408.00118. URL https://doi.org/10.48550/arXiv.2408.00118.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. WinoGrande: An adversarial winograd schema challenge at scale. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pp. 8732–8740. AAAI Press, 2020. doi: 10.1609/AAAI.V34I05.6399. URL https://doi.org/10.1609/aaai.v34i05.6399.

Maarten Sap, Hannah Rashkin, Derek Chen, Ronan Le Bras, and Yejin Choi. Social IQa: Commonsense reasoning about social interactions. In Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan (eds.), *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 4463–4473, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1454. URL https://aclanthology.org/D19-1454/.

Erik Schultheis and Dan Alistarh. LLMQ: Efficient lower-precision pretraining for consumer GPUs. *CoRR*, abs/2512.15306, 2025. doi: 10.48550/ARXIV.2512.15306. URL https://arxiv.org/abs/2512.15306.

Yikang Shen, Zhen Guo, Tianle Cai, and Zengyi Qin. JetMoE: Reaching Llama2 performance with 0.1m dollars. *CoRR*, abs/2404.07413, 2024a. doi: 10.48550/ARXIV.2404.07413. URL https://arxiv.org/abs/2404.07413.

Yikang Shen, Matthew Stallone, Mayank Mishra, Gaoyuan Zhang, Shawn Tan, Aditya Prasad, Adriana Meza Soria, David D Cox, and Rameswar Panda. Power scheduler: A batch size and token number agnostic learning rate scheduler. *arXiv preprint arXiv:2408.13359*, 2024b.

Xiaofeng Shi, Lulu Zhao, Hua Zhou, and Donglin Hao. Industrycorpus2. Hugging Face dataset, 2024. URL https://huggingface.co/datasets/BAAI/IndustryCorpus2.

Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism. *CoRR*, abs/1909.08053, 2019. URL http://arxiv.org/abs/1909.08053.

KaShun SHUM, Yuzhen Huang, Hongjian Zou, dingqi, YiXuan Liao, Xiaoxin Chen, Qian Liu, and Junxian He. Predictive data selection: The data that predicts is the data that teaches. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=tTVYR82Iz6.

Skywork-AI. Skywork community license. https://huggingface.co/datasets/Skywork/SkyPile-150B/blob/main/Skywork%20Community%20License.pdf, 2023. [Accessed 03-12-2025].

Dan Su, Kezhi Kong, Ying Lin, Joseph Jennings, Brandon Norick, Markus Kliegl, Mostofa Patwary, Mohammad Shoeybi, and Bryan Catanzaro. Nemotron-CC: Transforming Common Crawl into a refined long-horizon pretraining dataset. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2025, Vienna, Austria, July 27 - August 1, 2025*, pp. 2459–2475. Association for Computational Linguistics, 2025. URL https://aclanthology.org/2025.acl-long.123/.

Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc Le, Ed Chi, Denny Zhou, and Jason Wei. Challenging BIG-bench tasks and whether chain-of-thought can solve them. In *Findings of the Association for Computational Linguistics: ACL 2023*, pp. 13003–13051, Toronto, Canada, 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.findings-acl.824. URL https://aclanthology.org/2023.findings-acl.824/.

Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. CommonsenseQA: A question answering challenge targeting commonsense knowledge. In Jill Burstein, Christy Doran, and Thamar Solorio (eds.), *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 4149–4158, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1421. URL https://aclanthology.org/N19-1421/.

OpenThoughts-Agent Team. OpenThoughts-Agent. https://www.open-thoughts.ai/blog/agent, December 2025.

Evan Pete Walsh, Luca Soldaini, Dirk Groeneveld, Kyle Lo, Shane Arora, Akshita Bhagia, Yuling Gu, Shengyi Huang, Matt Jordan, Nathan Lambert, Dustin Schwenk, Oyvind Tafjord, Taira Anderson, David Atkinson, Faeze Brahman, Christopher Clark, Pradeep Dasigi, Nouha Dziri, Allyson Ettinger, Michal Guerquin, David Heineman, Hamish Ivison, Pang Wei Koh, Jiacheng Liu, Saumya Malik, William Merrill, Lester James Validad Miranda, Jacob Morrison, Tyler Murray, Crystal Nam, Jake Poznanski, Valentina Pyatkin, Aman Rangapur, Michael Schmitz, Sam Skjonsberg, David Wadden, Christopher Wilhelm, Michael Wilson, Luke Zettlemoyer, Ali Farhadi, Noah A. Smith, and Hannaneh Hajishirzi. 2 OLMo 2 furious (COLM's version). In *Second Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=2ezugTT9kU.

Liangdong Wang, Bo-Wen Zhang, Chengwei Wu, Hanyu Zhao, Xiaofeng Shi, Shuhao Gu, Jijie Li, Quanyue Ma, TengFei Pan, and Guang Liu. Cci3.0-hq: a large-scale chinese dataset of high quality designed for pre-training large language models, 2024a. URL https://arxiv.org/abs/2410.18505.

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, Tianle Li, Max Ku, Kai Wang, Alex Zhuang, Rongqi Fan, Xiang Yue, and Wenhu Chen. MMLU-Pro: A more robust and challenging multi-task language understanding benchmark. In *Advances in Neural Information Processing Systems*, volume 37, pp. 95266–95290. Curran Associates, Inc., 2024b. doi: 10.52202/079017-3018. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/ad236edc564f3e3156e1b2feafb99a24-Abstract-Datasets_and_Benchmarks_Track.html.

Tianwen Wei, Liang Zhao, Lichang Zhang, Bo Zhu, Lijie Wang, Haihua Yang, Biye Li, Cheng Cheng, Weiwei Lü, Rui Hu, Chenxia Li, Liu Yang, Xilin Luo, Xuejie Wu, Lunan Liu, Wenjun Cheng, Peng Cheng, Jianhao Zhang, Xiaoyu Zhang, Lei Lin, Xiaokun Wang, Yutuan Ma, Chuanhai Dong, Yanqi Sun, Yifu Chen, Yongyi Peng, Xiaojuan Liang, Shuicheng Yan, Han Fang, and Yahui Zhou. Skywork: A more open bilingual foundation model, 2023. URL https://arxiv.org/abs/2310.19341.

Kaiyue Wen, Xingyu Dang, Kaifeng Lyu, Tengyu Ma, and Percy Liang. Fantastic pretraining optimizers and where to find them ii: Hyperball optimization. *arXiv preprint arXiv:2606.16899*, 2026. URL https://arxiv.org/abs/2606.16899.

Yihao Xiao, Jialong Sun, Zitian Gao, Zeming Wei, Chutian Wang, Ran Tao, Jiaye Teng, and Bryan Dai. Hyperball may not be a free lunch, 2026. URL https://arxiv.org/abs/2607.22444.

Zijie Yan, Hongxiao Bai, Xin Yao, Dennis Liu, Tong Liu, Hongbin Liu, Pingtian Li, Evan Wu, Shiqing Fan, Li Tao, et al. Scalable training of mixture-of-experts models with megatron core. *CoRR*, abs/2603.07685, 2026. URL https://arxiv.org/abs/2603.07685. Technical report, version 2.

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jialong Tang, Jialin Wang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Ma, Jianxin Yang, Jin Xu, Jingren Zhou, Jinze Bai, Jinzheng He, Junyang Lin, Kai Dang, Keming Lu, Keqin Chen, Kexin Yang, Mei Li, Mingfeng Xue, Na Ni, Pei Zhang, Peng Wang, Ru Peng, Rui Men, Ruize Gao, Runji Lin, Shijie Wang, Shuai Bai, Sinan Tan, Tianhang Zhu, Tianhao Li, Tianyu Liu, Wenbin Ge, Xiaodong Deng, Xiaohuan Zhou, Xingzhang Ren, Xinyu Zhang, Xipin Wei, Xuancheng Ren, Xuejing Liu, Yang Fan, Yang Yao, Yichang Zhang, Yu Wan, Yunfei Chu, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, Zhifang Guo, and Zhihao Fan. Qwen2 technical report. *CoRR*, abs/2407.10671, 2024a. doi: 10.48550/ARXIV.2407.10671. URL https://doi.org/10.48550/arXiv.2407.10671.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report. *CoRR*, abs/2412.15115, 2024b. doi: 10.48550/ARXIV.2412.15115. URL https://doi.org/10.48550/arXiv.2412.15115.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jian Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report. *CoRR*, abs/2505.09388, 2025a. doi: 10.48550/ARXIV.2505.09388. URL https://doi.org/10.48550/arXiv.2505.09388.

Chen Yang, Ran Le, Yun Xing, Zhenwei An, Zongchao Chen, Wayne Xin Zhao, Yang Song, and Tao Zhang. Toolmind technical report: A large-scale, reasoning-enhanced tool-use dataset, 2025b. URL https://arxiv.org/abs/2511.15718.

Chen Yang, Guangyue Peng, Jiaying Zhu, Ran Le, Ruixiang Feng, Tao Zhang, Xiyun Xu, Yang Song, Yiming Jia, Yuntao Wen, Yunzhi Xu, Zekai Wang, Zhenwei An, Zhicong Sun, and Zongchao Chen. Nanbeige4.1-3b: A small general model that reasons, aligns, and acts, 2026. URL https://arxiv.org/abs/2602.13367.

Xi Ye, Fangcong Yin, Yinghui He, Joie Zhang, Howard Yen, Tianyu Gao, Greg Durrett, and Danqi Chen. Longproc: Benchmarking long-context language models on long procedural generation. *arXiv preprint arXiv:2501.05414*, 2025.

Hu Yiwen, Huatong Song, Jie Chen, Jia Deng, Jiapeng Wang, Kun Zhou, Yutao Zhu, Jinhao Jiang, Zican Dong, Yang Lu, Xu Miao, Xin Zhao, and Ji-Rong Wen. YuLan-mini: Pushing the limits of open data-efficient language model. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 5374–5400, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.268. URL https://aclanthology.org/2025.acl-long.268/.

Bowen Yu, Guanyu Feng, Huanqi Cao, Xiaohan Li, Zhenbo Sun, Haojie Wang, Xiaowei Zhu, Weimin Zheng, and Wenguang Chen. Chukonu: A fully-featured big data processing system by efficiently integrating a native compute engine into spark. *Proc. VLDB Endow.*, 15(4):872–885, 2021. doi: 10.14778/3503585.3503596. URL https://www.vldb.org/pvldb/vol15/p872-yu.pdf.

Yijiong Yu, Ziyun Dai, Zekun Wang, Wei Wang, Ran Chen, and Ji Pei. Opencsg chinese corpus: A series of high-quality chinese datasets for llm training, 2025. URL https://arxiv.org/abs/2501.08197.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. HellaSwag: Can a machine really finish your sentence? In Anna Korhonen, David Traum, and Lluís Màrquez (eds.), *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 4791–4800, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1472. URL https://aclanthology.org/P19-1472/.

Jintao Zhang, Haofeng Huang, Pengle Zhang, Jia Wei, Jun Zhu, and Jianfei Chen. SageAttention2: Efficient attention with thorough outlier smoothing and per-thread INT4 quantization. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 75097–75119. PMLR, 2025a. URL https://proceedings.mlr.press/v267/zhang25ae.html.

Peiyuan Zhang, Guangtao Zeng, Tianduo Wang, and Wei Lu. Tinyllama: An open-source small language model. *arXiv preprint arXiv:2401.02385*, 2024a.

Wanyue Zhang, Ziyong Li, Wen Yang, Chunlin Leng, Yinan Bai, Qianlong Du, Chengqing Zong, and Jiajun Zhang. Chinesewebtext 2.0: Large-scale high-quality chinese web text with multi-dimensional and fine-grained information, 2024b. URL https://arxiv.org/abs/2411.19668.

Yifan Zhang, Yifan Luo, Yang Yuan, and Andrew C Yao. Autonomous data selection with zero-shot generative classifiers for mathematical texts. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 4168–4189, Vienna, Austria, July 2025b. Association for Computational Linguistics. ISBN 979-8-89176-256-5. doi: 10.18653/v1/2025.findings-acl.216. URL https://aclanthology.org/2025.findings-acl.216/.

Zhengyan Zhang, Yuxian Gu, Xu Han, Shengqi Chen, Chaojun Xiao, Zhenbo Sun, Yuan Yao, Fanchao Qi, Jian Guan, Pei Ke, Yanzheng Cai, Guoyang Zeng, Zhixing Tan, Zhiyuan Liu, Minlie Huang, Wentao Han, Yang Liu, Xiaoyan Zhu, and Maosong Sun. Cpm-2: Large-scale cost-effective pre-trained language models. *AI Open*, 2:216–224, 2021a. ISSN 2666-6510. doi: 10.1016/j.aiopen.2021.12.003. URL https://www.sciencedirect.com/science/article/pii/S2666651021000310.

Zhengyan Zhang, Xu Han, Hao Zhou, Pei Ke, Yuxian Gu, Deming Ye, Yujia Qin, Yusheng Su, Haozhe Ji, Jian Guan, Fanchao Qi, Xiaozhi Wang, Yanan Zheng, Guoyang Zeng, Huanqi Cao, Shengqi Chen, Daixuan Li, Zhenbo Sun, Zhiyuan Liu, Minlie Huang, Wentao Han, Jie Tang, Juanzi Li, Xiaoyan Zhu, and Maosong Sun. Cpm: A large-scale generative chinese pre-trained language model. *AI Open*, 2:93–99, 2021b. ISSN 2666-6510. doi: 10.1016/j.aiopen.2021.07.001. URL https://www.sciencedirect.com/science/article/pii/S2666651021000019X.

Chuyue Zhou, Hongya Lyu, Xinle Lin, Hengyu Zhao, Junshao Guo, Xueren Zhang, Shuaikang Xue, Qiang Ma, Jie Zhou, Yudong Wang, and Zhiyuan Liu. Ultradata-math, 2026a. URL https://huggingface.co/datasets/openbmb/UltraData-Math.

Fan Zhou, Zengzhi Wang, Nikhil Ranjan, Zhoujun Cheng, Liping Tang, Guowei He, Zhengzhong Liu, and Eric P. Xing. Megamath: Pushing the limits of open math corpora. *CoRR*, abs/2504.02807, 2025. doi: 10.48550/ARXIV.2504.02807. URL https://doi.org/10.48550/arXiv.2504.02807.

Zhanpeng Zhou, Wenjie Zhou, Yufei Gu, Juqiu Wang, Guanwei Zhang, Ruobing Xie, Xianbiao Qi, Yushi Bai, Kaifeng Lyu, and Shunyu Yao. From lr to elr: A better heuristic for pretraining dynamics, 2026b. URL https://hy.tencent.ai/research/elr.

Jingwei Zuo, Maksim Velikanov, Ilyas Chahed, Younes Belkada, Dhia Eddine Rhayem, Guillaume Kunsch, Hakim Hacid, Hamza Yous, Brahim Farhat, Ibrahim Khadraoui, Mugariya Farooq, Giulia Campesan, Ruxandra Cojocaru, Yasser Djilali, Shi Hu, Iheb Chaabane, Puneesh Khanna, Mohamed El Amine Seddik, Ngoc Dung Huynh, Phuc Le Khac, Leen AlQadi, Billel Mokeddem, Mohamed Chami, Abdalgader Abubaker, Mikhail Lubinets, Kacper Piskorski, and Slim Frikha. Falcon-H1: A family of hybrid-head language models redefining efficiency and performance. *CoRR*, abs/2507.22448, 2025. doi: 10.48550/ARXIV.2507.22448. URL https://doi.org/10.48550/arXiv.2507.22448.

Appendices

Contents

1 Introduction **2**

2 Training Recipe **3**

2.1 RTX 5090 GPUs for Pretraining: Cost-Efficient and Accessible 4

2.2 MuonH and Effective Learning Rate Schedule Design 5

2.3 Curriculum Training and Data Selection 6

2.3.1 Multi-Source Curriculum Model Averaging 6

2.3.2 Dataset Benchmarking for Data Selection and Mixture Design 7

3 Evaluation **7**

3.1 End-to-End Evaluation 7

3.2 Cost-Saving Ablation of Each Component 7

3.3 Case Study: Post-Training Comparison between UD and CMA Pretraining 9

4 Conclusion **9**

A Data Benchmarking for Data Selection and Mixture Design **26**

B Limitations **26**

C Related Works **26**

C.1 Open-Recipe Language Models 26

C.2 Low-Cost Language Model Pretraining 27

D Training Recipe **28**

D.1 Training Infrastructure 28

D.1.1 RTX 5090 as a Cost-Effective GPU Choice 28

D.1.2 Hardware Setup and Tweaks 29

D.1.3 Efficient Training System 29

D.2 FP8 Mixed-Precision Training 31

D.3 Hyperball Optimization 32

D.3.1 Effective Learning Rate 32

D.3.2 Learning Rate Schedule Design 35

D.4 Curriculum Model Averaging 35

D.5 Post-Training Recipe 37

D.6 Data Recipe 38

D.6.1 How Do We Select a Good Data Recipe? 38

D.6.2 How Do We Preprocess Data and Reproduce the Shards? 39

E Cost Assumptions **40**

F Training Details **42**

F.1 Baselines, Benchmarks, and Evaluation Protocol 42

F.2 Production Training Details 43

G Scaling Ladder **44**

H Post-Training Details **47**

H.1 Shared Construction and Evaluation Protocol 47

H.2 GSM8K-Based SFT 48

H.3 Math&Code SFT with Replay 48

H.4 Tulu-3 Mixed-Domain SFT 49

I LR Schedule Analysis **51**

I.1 Details about Muon-ELR 51

I.2 Multi-Power Law for Effective Learning Rate Schedules 52

I.3 Details of the WSD Decay-Range Sweeps 54

J Scaling Checkpoint Ledger **54**

K Curriculum and CMA **56**

K.1 Curriculum Bucket Construction 56

K.2 Phase Transition and UD Control 57

K.3 Constant-LR Continuation and Checkpoint Averaging 57

L Data Recipe **58**

A Data Benchmarking for Data Selection and Mixture Design

Dataset benchmarking via proxy experiments. We build a proxy experiment pipeline to compare different data sources and different quality levels within each source. For large datasets with usable sample-level scores, we construct four candidate slices from different positions in each source's score ordering. Other eligible datasets contribute a random 4B-token slice. Each slice is tested in a 2,000-step training run starting from the same Qwen3-0.6B checkpoint. All runs use the same training configuration and the same schedule for mixing candidate data with replay data. We then evaluate the resulting checkpoints on a fixed 15-benchmark suite covering mathematics, code, Chinese, and general capabilities (Figure 13). Sampling rules and training settings are given in Section L.

Benchmark results guide source selection and mixture weights. We use the measured results to guide data allocation in Phase 2. For example, MegaMath-Web-Pro and MegaMath-Code rank top two in our proxy evaluation. The former performs strongly on Math and General, while the latter provides the strongest Code signal among the candidates (Table 25). Hence, we assign both a larger share in Phase 2 to strengthen the target capabilities. We also find that a dataset strong in one capability can lag behind in another. In particular, mathematics shows a weaker trade-off with general capabilities than coding does. We therefore increase the math share in Phase 2 when balancing different capabilities (Figure 18 and § L).

B Limitations

This section records limitations of the report's claims that can be improved in the future.

Chinese capability scope. The pretraining mixture includes Chinese data to provide a basic level of Chinese-language coverage, but Chinese capability is not a primary target of this release and was not the main axis used for optimization or headline model selection. We therefore omit Chinese benchmark scores from the release-facing model-comparison tables, while retaining the Chinese proxy axis used to audit data-mixture decisions. The headline evaluation remains a mathematics, code, reasoning, and knowledge study. Future versions should make the intended language targets explicit before tuning the mixture and evaluation protocol.

Overtrained dense regime. Puro-2B is a dense 2B-parameter model trained on approximately 1.4T tokens, or about 700 tokens per parameter. This is an overtrained, data-rich regime: it is useful for studying data and recipe choices and for obtaining a compact model with low inference cost, but it is not presented as compute-optimal. The chosen point balances RTX 5090 memory and communication limits, attainable benchmark quality, and the level of community support for a dense base model.

Scale-up and architecture scope. The Puro Cost Scaling Law is a fixed-2B, recipe-specific scale-down curve. A larger model or a larger world size would require a new communication and memory design; for example, a smaller vocabulary or another embedding/LM-head partition could be needed before scale-up. The current report therefore does not claim a model-size scale-up law. Especially when scaling up model size, the HBM memory capacity of RTX 5090 can be a bottleneck to deal with.

C Related Works

C.1 Open-Recipe Language Models

The core science and engineering of large-scale pretraining remain difficult to study when leading systems are either closed or release weights without the data and training recipe needed to reproduce the run (Luo et al., 2025a). The term "open model" therefore covers substantially different release practices. Widely used model families such as Qwen (Yang et al., 2024a;b; 2025a; Qwen Team, 2026), Gemma (Rivière et al., 2024; Gemma Team, 2025; 2026), Llama (Meta AI, 2024b), LFM (Liquid AI, 2025), and Falcon (Zuo et al., 2025) publish model weights and enough implementation information for evaluation and deployment. However, their exact pretraining corpora, sample

order, and complete training state are not released, so the community cannot reproduce the original training run. We refer to these releases as *open-weight*. In contrast, an *open-recipe* release aims to expose the model weights, training data or a reproducible data recipe, training and evaluation code, and sufficiently detailed configurations and intermediate artifacts to support scientific inspection and reproduction (Groeneveld et al., 2024; Walsh et al., 2025). Such releases turn a final checkpoint into a reproducible account of how the model was built, making controlled study possible for resource-limited research communities.

Open-recipe projects provide the broader experimental stack needed to study pretraining itself. Pythia releases ordered data and dense intermediate checkpoints; OLMo and OLMoE release data, training and evaluation code, checkpoints, and logs. Projects such as SmolLM3, Yulan-Mini-2.4B, and Instella provide public data mixtures and reproducible training configurations (Biderman et al., 2023; Groeneveld et al., 2024; Muennighoff et al., 2025; Bakouch et al., 2025; Yiwen et al., 2025; Liu et al., 2025). These resources allow other researchers to investigate learning dynamics, data selection, scaling, and optimization without relying on undocumented industrial pipelines. Open-recipe models can also be competitive: as shown in Table 2, models such as SmolLM3-3B, Yulan-Mini-2.4B, and Instella-3B match or surpass several open-weight models of comparable scale on parts of our evaluation, making this direction important beyond reproducibility alone.

This openness and capability have nevertheless often required substantial resources. The training cost of Yulan-Mini-2.4B is estimated at approximately USD 48K, while SmolLM3 was trained on 384 H100 GPUs and the first stage of Instella-3B on 128 MI300X GPUs. (Yiwen et al., 2025; Bakouch et al., 2025; Liu et al., 2025). These examples expose a remaining gap between openness and practical accessibility: a training pipeline may be fully reproducible yet remain too costly for small research groups to rerun. Addressing this gap requires jointly optimizing the data recipe, training algorithm, numerical precision, and hardware utilization under explicit cost accounting.

C.2 Low-Cost Language Model Pretraining

Training cost is meaningful only under an explicit accounting boundary. Token count and FLOPs alone capture only part of pretraining cost. End-to-end replication cost also depends on the workload required to reach a target quality, realized hardware throughput, and hardware price. We therefore consider cost jointly with model quality rather than using any single measure of compute or hardware efficiency as a proxy for low-cost training.

Low-precision training primarily reduces the cost of executing a fixed workload. FP8 mixed precision accelerates the matrix multiplications that dominate Transformer training while retaining higher precision where needed. DeepSeek-V3 shows that this design can remain stable at large scale (Micikevicius et al., 2022; DeepSeek-AI et al., 2024). Quartet II extends this direction to NVFP4 and improves its numerical behavior (Panferov et al., 2026). These methods improve execution efficiency for a given training workload, but do not by themselves reduce the token or nominal FLOP budget required to reach a target quality.

A complementary line of work reduces the training workload through model architecture, data selection, staged training, and systems optimization. TinyLlama, JetMoE, Yulan-Mini-2.4B, and Kaiyuan-2B explore different combinations of these techniques (Zhang et al., 2024a; Shen et al., 2024a; Yiwen et al., 2025; Luo et al., 2025a). Collectively, these projects show that useful small-model capability can be obtained under substantially smaller compute budgets than those of large-scale foundation models. The reported runs, however, generally rely on specialized data-center accelerators, so the hardware barrier for a small research group remains.

Hardware accessibility introduces a separate constraint. QLoRA reduces the memory required to finetune a pretrained model, leaving the original pretraining cost unchanged (Dettmers et al., 2023). LLMQ studies full-model pretraining on RTX 4090 GPUs using FP8, recomputation, sharding, and CPU offloading (Schultheis & Alistarh, 2025), while Quartet II measures the throughput benefit of NVFP4 on a single RTX 5090 (Panferov et al., 2026). The studies considered here establish the feasibility of relevant training operations on consumer GPUs, but focus primarily on system-level techniques or throughput characterization rather than on releasing a trillion-token-scale base model with an explicit replication-cost boundary.

PURO-2B lies at the intersection of these directions. It is a from-scratch 2B base model trained on more than 1.4T tokens using FP8 on a multi-node RTX 5090 cluster, with measured accelerator usage and an explicit replication-cost boundary. We therefore evaluate the training recipe and hardware platform end to end, relating system efficiency directly to the resulting model quality. Together, these components provide researchers with a reproducible and cost-accessible end-to-end pipeline for studying pretraining behavior and testing alternative methods at meaningful scales.

D TRAINING RECIPE

We organize the training recipe as follows. We first describe the infrastructure design that supports our training setup. We then present the pretraining recipe, focusing on optimization and data curriculum. Next, we explain how the training dataset is constructed. Finally, we present a post-training case study to demonstrate how our recipe enables end-to-end approach comparison.

D.1 TRAINING INFRASTRUCTURE

D.1.1 RTX 5090 AS A COST-EFFECTIVE GPU CHOICE

To make our recipe cost-efficient and the hardware accessible to the broader community, we made a non-standard infrastructure choice: we run all pretraining and post-training on a cluster equipped with consumer-grade GPUs (RTX 5090), instead of renting conventional data-center-oriented accelerators. Although data-center GPUs are generally expected to be more stable and compute-efficient, our cost analysis showed that RTX 5090 is a better fit for this workload.

Table 3: Peak hardware capability and cost efficiency of candidate NVIDIA GPUs.

GPU Model	Peak tensor performance (TFLOPS)[a]			Memory capacity	Memory bandwidth	TDP	Intra-node P2P BW (Bi-Di)	Single-GPU price[b]	Peak compute/$ (EFLOP/USD)		
	BF16	FP8	NVFP4						BF16	FP8	FP4
A100 SXM4	312	N/A	N/A	80 GB	2 TB/s	400 W	600 GB/s	$1.79/h	0.63	N/A	N/A
H200 SXM5	989.5	1979 [c]	N/A	141 GB	4.8 TB/s	700 W	900 GB/s	$4.00/h	0.89	1.78	N/A
RTX PRO 6000	503.8	1007.6	2015.2	96 GB	1.8 TB/s	600 W	128 GB/s [d]	$1.89/h	0.96	1.92	3.85
RTX 5090	209.5	419	1676	32 GB	1.8 TB/s	575 W	128 GB/s [d]	**$0.31/h**	**2.43**	**4.87**	**19.46**

[a] All performance numbers are from official NVIDIA documentation / datasheets / blog posts (Andersch et al., 2022; Corporation, 2024; 2025a;b) and refer to dense matrix operations with FP32 accumulation.
[b] The prices do not include CPU / storage / network, as they are marginal and often free on GPU rental platforms. A100, H200, and RTX PRO 6000 prices were collected from a public GPU rental platform on 16 Aug 2026. RTX 5090 has no widely available public rental channel due to NVIDIA EULA restrictions; we estimate its effective price by amortizing our hardware and electricity costs over five years. An 8-GPU node costs approximately 12,000 CNY ($1763) per month for our setup. Some private compute providers confirmed similar rental pricing for RTX 5090.
[c] Hopper FP8 Tensor Cores use a reduced-precision path for nominal FP32 accumulation, with 22 effective precision bits (Zhang et al., 2025a), which may affect numerical accuracy.
[d] PCIe Gen5 × 16 provides approximately 64 GB/s of theoretical bandwidth per direction, or approximately 128 GB/s of aggregate bidirectional bandwidth. Driver tweaks needed to achieve this bandwidth are discussed in Section D.1.2.

Table 3 shows the features and prices of different generations of NVIDIA GPUs. RTX 5090 has a clear gap in absolute peak throughput compared with data-center GPUs, especially because its Tensor Cores are artificially limited to half of actual peak performance when using FP32 accumulation. However, its much lower effective price gives substantially higher compute per dollar: its BF16 and FP8 cost efficiency is about 2.7× that of H200. Moreover, because policy and cost constraints made Blackwell data-center accelerators such as GB200 unavailable to us, RTX 5090 is also the most practical option with FP4 support.

RTX 5090 still has significant hardware limitations, including smaller memory capacity and the lack of NVLink for high-bandwidth GPU-to-GPU scaling. These drawbacks cannot be ignored. Nevertheless, our target model is small enough that, with the hardware configuration and parallel training setup described below, the RTX 5090 cluster reaches a mixed-precision effective MFU of approximately 73% under the stated precision-weighted peak convention. Section D.1.3 defines the convention and measurement boundary; this is not a BF16-only full-scale result. Its cost advantage therefore remains valid for our setting.

D.1.2 Hardware Setup and Tweaks

Our training cluster consists of multiple x86 GPU servers. Each server is equipped with dual CPUs, eight RTX 5090 GPUs (four in each CPU socket), and sufficient host memory. The performance numbers in Section D.1.1 are single-GPU metrics; for multi-GPU training, both intra-node P2P bandwidth and inter-node network bandwidth must be considered to sustain training throughput.

Intra-node bandwidth. Since RTX 5090 does not provide NVLink, the only intra-node interconnect is PCIe. NVIDIA disables PCIe P2P on non-data-center GPUs, which forces GPU communication traffic to be staged through host memory, also known as "ping-pong" transfer, which can substantially reduce communication efficiency. However, this restriction appears to be enforced by the NVIDIA GPU driver rather than the limitation of GPU hardware. We used a modified version of the open-source NVIDIA driver (aikitoria, 2026), along with the necessary platform configuration changes, including disabling IOMMU and PCIe ACS as well as adjusting NPS (NUMA Per Socket, also known as Sub-NUMA Clustering on some platforms) to enable GPU P2P on our servers.

On our hardware, enabling P2P improved one-way bandwidth from 31.5 GB/s to 56 GB/s. The former is bounded by half of the PCIe 5.0 $\times$16 bandwidth, i.e., 32 GB/s, because ping-pong transfer requires two host-memory copies; the latter approaches the 64 GB/s PCIe link limit. Bidirectional bandwidth increased from 32 GB/s to 111 GB/s, compared with theoretical limits of 128 GB/s. Communication latency decreased drastically from 14.3 μs to 0.4 μs. The eight-GPU `AllReduce` bandwidth over PCIe, measured as `busbw` reported by `nccl-tests`, increased from 14.75 GB/s to 27.34 GB/s. When the test was restricted to the four GPUs attached to a single CPU socket, the gain was larger, from 16.33 GB/s to 46.31 GB/s, about 2.8$\times$. Adding a PLX PCIe switch between the GPUs and the root complex might further improve P2P bandwidth toward the link limit; this is a prospective topology change, not a configuration used in the reported run.

Inter-node bandwidth. To scale training beyond a single server, the inter-node network must provide bandwidth comparable to the intra-node communication path. We set up a 400 Gbps InfiniBand network among our servers, i.e., 100 GB/s bidirectional bandwidth: one dual-port Mellanox ConnectX-7 host card adapter (HCA) is installed in each server, with both ports connected to a 200 Gbps InfiniBand HDR switch. Half of the GPUs in the server could communicate with the HCA directly, while the other half must route through the inter-socket link to reach the HCA.

Even after PCIe P2P is enabled, non-data-center NVIDIA GPUs still do not support GPUDirect RDMA (shortened as GDR), another performance-critical feature for inter-node communication. GDR builds on P2P and allows a GPU to access remote GPU memory over RDMA NICs without staging through host memory. Through testing, we found that this restriction is also software-enforced: a minimal binary modification to the CUDA user-space driver can enable GDR on RTX 5090. Due to EULA constraints, we are not able to disclose the details in this report. When this configuration is unavailable, the same training recipe can be run with the stock NVIDIA driver, falling back to the standard communication path at lower inter-node bandwidth. In end-to-end tests, enabling GDR improved the 24-GPU (3 servers) `AllReduce` bandwidth, measured as `busbw`, from approximately 8.87 GB/s to 19.93 GB/s. We believe that if a more balanced network topology is used (e.g., one InfiniBand HCA on each socket), the performance gain could be even higher.

Caveats. These modifications should be used with caution. First, modifying drivers is unsupported by the vendor and must be done at the user's own risk. Second, these changes cannot exceed the physical PCIe bandwidth limit. Third, P2P and GDR should not be enabled blindly; the optimal setting depends on the actual hardware topology and platform capability. For example, if the PCIe root complex of a CPU has limited peer-switching capability (which is often the case on low-end models), heavy P2P traffic among many devices may become congested, and host-memory ping-pong transfer can be faster.

D.1.3 Efficient Training System

We build our training system on Megatron Core (Shoeybi et al., 2019), specifically release v0.16.0 (NVIDIA Corporation, 2026a), together with its Transformer Engine dependency. This software stack provides native support for blockwise FP8 training and the Muon optimizer, making it a

suitable basis for a model at our scale. Nevertheless, the combination of consumer GPUs, a mixed-precision training recipe, and our model architecture still requires careful system-level tuning.

Communication-aware parallelism. The limited communication bandwidth of RTX 5090 motivates us to use only parallel dimensions with modest communication demands. We combine data parallelism (DP), whose dominant collectives are gradient `ReduceScatter` and parameter `AllGather` around optimization, with pipeline parallelism (PP), which exchanges hidden-state tensors only at stage boundaries (Narayanan et al., 2021). We do not use tensor parallelism (TP), which introduces frequent collectives within every Transformer layer. Compared with data-center-class systems, our communication disadvantage is substantially more pronounced within a node. We therefore depart from Megatron Core's default rank ordering and use a `pp-dp` ordering that places each PP group on topologically closer GPUs within a node, thereby reducing sensitivity to communication latency.

Appropriate micro-batch size. FP8 kernels complete their arithmetic more quickly while introducing additional quantization operations, so they become memory-bound more readily than BF16 kernels. This effect is amplified by the relatively small hidden dimension of our model. Increasing the micro-batch size (MBS) improves the arithmetic intensity and Tensor Core utilization, but an excessive MBS exhausts device memory: DP does not partition activations; and the 1F1B pipeline schedule (Narayanan et al., 2021) helps control the number of simultaneously resident microbatches, but it does not eliminate the per-microbatch activation-memory growth caused by increasing MBS. We therefore extend Megatron-LM's analytical FLOP estimator to report the invocation shapes of GEMM and FlashAttention kernels. For each shape induced by our model configuration, we benchmark the corresponding Transformer Engine kernel and select the smallest MBS beyond which achieved FLOP/s no longer increases materially, analogous to locating the knee of a roofline curve.

Overcoming load imbalance. Although our model is relatively small, its vocabulary is large; consequently, the embedding and LM head account for a non-negligible fraction of both computation and memory. FP8 GEMMs accelerate the internal Transformer layers, further increasing their performance gap with the LM head. As a result, the LM head alone entails computation comparable to several Transformer layers, creating substantial pipeline imbalance. Guided by our extended FLOP estimator, which attributes theoretical work by layer, submodule, operator, and precision, we assign fewer Transformer layers to the stage containing the LM head. This balances pipeline computation at the cost of greater memory pressure on the first stage. Tensors assigned to Muon are preferably kept intact rather than sharded. We replace the zigzag placement policy of the layer-wise distributed optimizer in this version of Megatron Core, which over-concentrates the embedding and LM-head tensors, with a memory-aware allocation (DeepSeek-AI et al., 2026): Muon tensors are greedily assigned by memory footprint like bin packing, after which flattened Adam tensors fill the remaining per-device capacity as evenly as possible. This hybrid placement balances optimizer memory across GPUs and is important for fitting the training state into the 32 GB memory available on each RTX 5090.

These principles yield the same fastest configuration as an exhaustive enumeration of the candidate parallel strategies. Together with the hardware optimizations described above and standard Megatron Core options such as the `[T,H,D]` layout for packed-sequence attention, the Phase 1 production run sustains a median 238 TFLOP/s per GPU with global batch size 1536 on 24 GPUs across three nodes. The best configuration is MBS $= 2$, PP $= 2$ with layout `(18|10)`[5], and DP $= 12$. The measured throughput corresponds to an equivalent MFU[6] of approximately 73%. For larger-scale Phase 2 training, strong scaling is limited by Amdahl's law, and gradient synchronization accounts for a larger fraction of the step time. We consequently use PP $= 4$ with layout `(9|9|9|1)` and

[5]The PP layout lists the numbers of Transformer layers assigned to successive stages. In `(18|10)`, the first stage contains the embedding and the first 18 Transformer layers, while the second contains the remaining 10 layers and the LM head.

[6]Model FLOPs utilization (MFU) is conventionally defined as achieved model FLOP/s divided by the accelerator's peak FLOP/s (Korthikanti et al., 2023). For mixed-precision training, we weight the peak Tensor Core performance by the fraction of theoretical work at each precision. In our recipe, FP8 accounts for 72% of the theoretical Tensor Core operations and BF16 for the remaining 28%, giving an effective peak of $P_{\text{eff}} = (0.72/419 + 0.28/209.5)^{-1} \approx 327$ TFLOP/s per GPU.

DP $= 24$ on 96 GPUs, while still achieving median throughput 192 TFLOP/s per GPU. The run-level setup and final losses are reported in Table 7.

D.2 FP8 Mixed-Precision Training

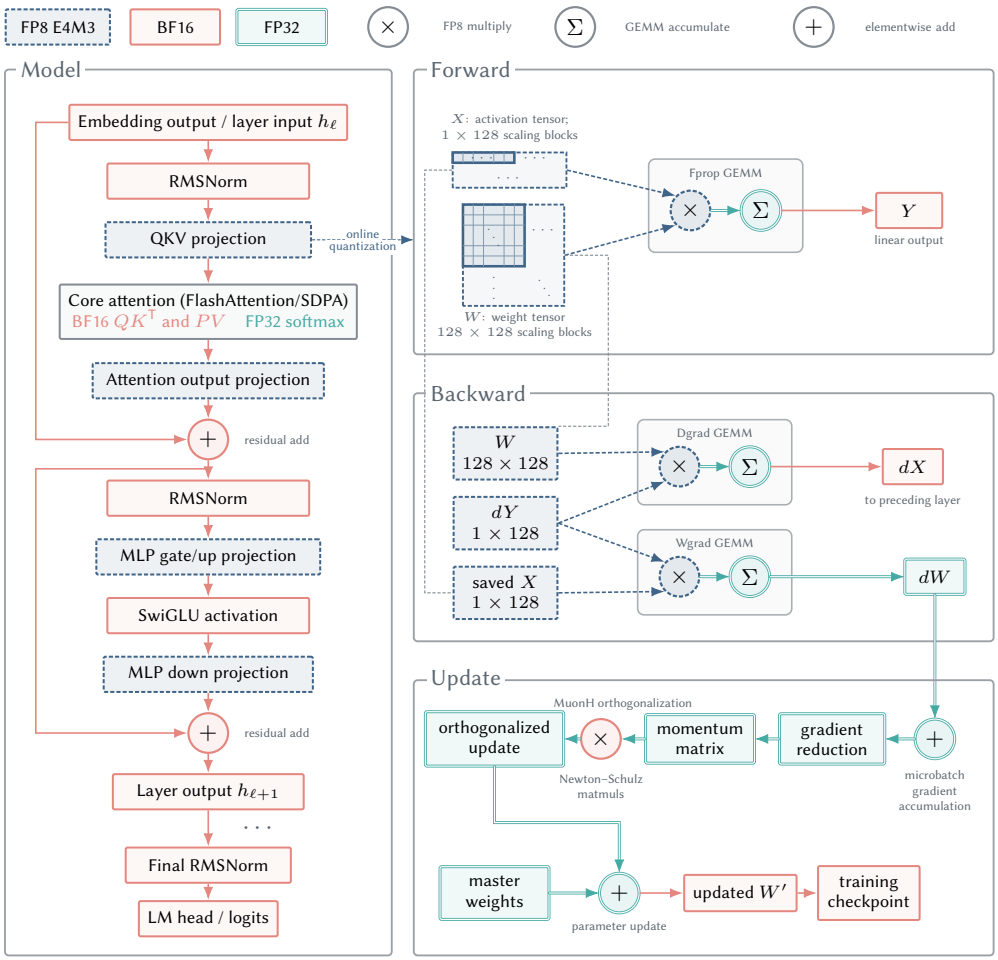


Figure 7: Precision flow for the FP8 recipe. Linear-layer GEMMs use blockwise E4M3 for operands and saved activations. The surrounding outputs, communication tensors, and checkpoint weights remain BF16, while gradient and optimizer-state accumulation remains FP32.

We use FP8 mixed precision from random initialization onward, without a BF16 warm-up stage or a later precision switch, using our training stack (NVIDIA Corporation, 2026a) and its blockwise FP8 support (NVIDIA Corporation, 2026b). The training state and numerically sensitive operations retain the standard BF16/FP32 mixed-precision path, while Transformer linear layers use E4M3 operands. FP8 is therefore an online compute and activation-storage format rather than the persistent dtype of the whole model.

E4M3 (Micikevicius et al., 2022) provides more mantissa bits than E5M2 but has a narrower dynamic range. With one scale for an entire tensor, a few outliers can enlarge the quantization interval and reduce the effective precision of ordinary values (DeepSeek-AI et al., 2024). We instead compute scales online for local groups: activations and activation gradients use one-dimensional groups of 128 consecutive values along the GEMM reduction dimension, while weights use two-dimensional 128×128 blocks (Yan et al., 2026). This fine-grained numerical design follows the recipe introduced and validated at scale by DeepSeek-V3 (DeepSeek-AI et al., 2024).

On our RTX 5090 (SM 120) (Corporation, 2025a), each logical block scale is constrained to a power of two and therefore has the exponent-only expressiveness of E8M0. Transformer Engine (NVIDIA

Corporation, 2026b) implements these blocks through Blackwell's native MXFP8 path. Thus, our logical block sizes follow DeepSeek-V3 (DeepSeek-AI et al., 2024), while the scaling-factor representation and execution path follow MXFP8; the fact that DeepSeek-V3 was trained on Hopper GPUs may account for this implementation difference.

The complete precision flow is shown in Figure 7. Here, Transformer linear layers include the QKV projections, attention output projection, and MLP projections; their Fprop, Dgrad, and Wgrad GEMMs all use the FP8 path (DeepSeek-AI et al., 2024). Core attention refers only to the FlashAttention/SDPA computation between these projections and remains in BF16 (Yan et al., 2026). A linear layer quantizes its BF16 inputs and weights immediately before GEMM and materializes a BF16 output for the surrounding model. Its quantized input is retained for Wgrad, reducing the dominant saved-activation footprint (Yan et al., 2026). FP8-accelerated operations account for 72% of the overall computation; the cost impact and net benefit of FP8 are analyzed in Figure 2.

The model-weight portion of each training checkpoint consequently remains BF16; the desired release or deployment representation is produced by a separate post-training conversion. In the matched 20-token-per-parameter scaling ladder, blockwise FP8 increases validation loss by 0.0031–0.0039 relative to BF16 across the five tested model sizes. The shared-shape fit maps this difference to 98.0% BF16-equivalent compute retention. At the 1.7B scale, however, blockwise FP8 improves median training throughput by $1.36\times$, yielding a quality-adjusted net gain of $1.34\times$. The complete setup, fitting procedure, and size-wise results are reported in Section 3.2.

D.3 Hyperball Optimization

In our recipe, we adopt Muon with the Hyperball wrapper (MuonH), introduced by Wen et al. (2026), as the training optimizer for selected weight matrices. We apply MuonH to approximately scale-invariant weights, including attention and MLP matrices, while embeddings, normalization layers, the language-model head, and the remaining parameters use AdamW. Following Wen et al. (2026), MuonH normalizes the Muon update and constrains each wrapped matrix to a fixed-radius sphere. More specifically, for each wrapped matrix W_t, MuonH fixes the radius at $R = \|W_0\|_F$ and normalizes the Muon update u_t as $\widehat{u}_t = u_t/\|u_t\|_F$. It then applies

$$\widetilde{W}_{t+1} = W_t - \eta_t R\widehat{u}_t, \qquad W_{t+1} = R\ \text{Normalize}(\widetilde{W}_{t+1}), \tag{1}$$

where $\text{Normalize}(A) = A/\|A\|_F$. The first operation normalizes the update scale, while the second projects the updated weight matrix back to its initial Frobenius radius. The illustration of the MuonH optimizer is shown in Figure 8.

We vary the learning rate η_t according to a two-phase learning rate schedule. The design of this schedule is motivated by an analysis of how the loss gap between MuonH and Muon arises from the perspective of effective learning rate, as we elaborate below.

D.3.1 Effective Learning Rate

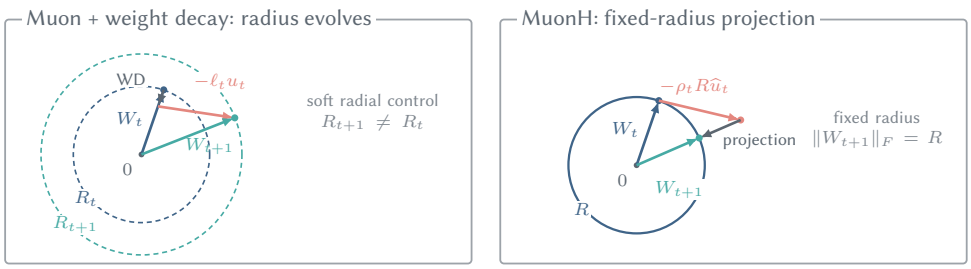
First, we briefly revisit the rationale behind Hyperball optimization and explain how it makes the effective LR schedule explicit (Wen et al., 2026). We then examine the loss gap between ordinary Muon and MuonH runs and find that the effective LR schedule strongly affects the loss convergence rate. This observation leads us to focus on learning rate schedule design for the MuonH optimizer.

Scale invariance motivates the effective learning rate. Hyperball is motivated by approximately scale-invariant parameter matrices. For a selected weight matrix W, we call the parameter approximately *scale invariant* when positively rescaling it has little effect on the model function or loss,

$$\mathcal{L}(cW) \approx \mathcal{L}(W), \qquad c > 0, \tag{2}$$

with the remaining parameters held fixed (Wen et al., 2026). Normalization in modern Transformers makes this approximation relevant for several attention and MLP weight matrices. In this regime, the update magnitude is naturally interpreted relative to the current weight scale. We therefore consider the matrix-wise effective LR, also called the *effective learning rate* (ELR). For ordinary Muon, with Muon update u_t, scalar learning rate η_t, and decoupled weight-decay coefficient λ,

$$W_{t+1} = \big(1 - \lambda\eta_t\big)W_t - \eta_t u_t, \tag{3}$$



	Muon + weight decay	**MuonH**
Weight radius	Evolves under weight decay and optimizer updates	Fixed at $R = \|W_0\|_F$ after each step
Displacement norm	$\eta_t \|u_t\|_F$ (state dependent)	$\eta_t R$ (scheduled)
Effective LR	$\eta_t \|u_t\|_F / \|W_t\|_F$ (indirect)	η_t (explicit)

Figure 8: Effective-LR control for one matrix. Ordinary Muon (left) applies a scalar learning rate to a state-dependent update norm, so both its effective LR and matrix radius emerge indirectly. MuonH (right) normalizes the update, prescribes a displacement of length $\rho_t R$, and projects the result back to the fixed-radius sphere.

and, ignoring the radial shrinkage from weight decay, the effective LR is

$$\rho_t(W) = \frac{\eta_t \|u_t\|_F}{\|W_t\|_F}. \tag{4}$$

Thus, ordinary Muon's effective LR is not determined by the scalar learning rate alone, because both the weight norm and the Muon-update norm evolve during training.

MuonH makes the effective LR schedule explicit. Under the Hyperball update in Equation (1), the pre-projection displacement $\eta_t R\widehat{u}_t$ has norm $\eta_t R$, while the wrapped matrix W_t has norm R. Its effective LR is therefore $\rho_t(W) = \eta_t \|u_t\|_F / \|W_t\|_F = \eta_t R / R = \eta_t$, numerically equal to the Hyperball weight learning rate, and the subsequent projection restores the fixed radius. Thus, ordinary Muon induces its effective LR indirectly through the scalar learning rate and evolving norms, whereas *MuonH directly controls the effective LR schedule through its learning rate schedule.* This explicit control is the property of Hyperball that is most relevant to our subsequent schedule design.

In our experiments, we use different learning rates for the MuonH-wrapped and AdamW-optimized parameter groups. We define a base learning rate schedule η_t^{base} for the AdamW parameter groups and set the Hyperball weight learning rate to a fixed multiple, $\eta_t^H = m\,\eta_t^{\text{base}}$. For MuonH-wrapped matrices, η_t^H is numerically equal to the prescribed effective LR. In the production run, we use $m = 10$, while the remaining parameter groups follow η_t^{base} directly with AdamW.

Matching the effective LR schedule also aligns validation loss. The MuonH optimizer can make training more efficient than the ordinary Muon optimizer, and we find that the efficiency gap can be attributed to the difference in their effective LR schedule. We compare three 170M-parameter BF16 runs with the same architecture, batch size, sequence length, seed, and evaluation protocol. The MuonH run uses a predefined effective LR schedule ρ_t^H. The ordinary-LR run uses the base LR schedule of the MuonH run as the LR schedule and does not compensate for the evolving weight and update norms. Then, as shown in Figure 9, the ordinary-LR Muon run produces an effective LR trace that decays rapidly early in training and finishes close to zero. The MuonH run instead follows the predefined linear-decay effective-LR schedule. Although the ordinary-LR Muon run achieves lower validation loss early in training, MuonH overtakes it near the end. The MuonH run finishes at a validation loss of 3.029, compared with 3.073 for ordinary-LR Muon.

For comparison, we conduct an effective-LR aligned run, which uses the ordinary Muon optimizer but adjusts its scalar learning rate online to follow the same effective LR curve,

$$\eta_t = \rho_t^H \frac{\|W_t\|_F}{\|u_t\|_F}, \tag{5}$$

so that the effective LR $\rho_t(W) = \rho_t^H$ without projecting W_{t+1} back to a fixed-radius sphere. As shown in Figure 9, effective-LR aligned Muon follows the prescribed linear post-warmup LR decay and reaches a final validation loss of 3.030, close to MuonH at 3.029. Thus, in this selected diagnostic, *ordinary Muon largely recovers the MuonH loss curve when its effective LR schedule is aligned*. This comparison supports the effective-LR schedule as an important factor in MuonH's training behavior. It also suggests that better control of the effective-LR schedule can improve loss convergence. Recent work also discusses related behavior from the perspective of angular update size (Zhou et al., 2026b; Xiao et al., 2026).

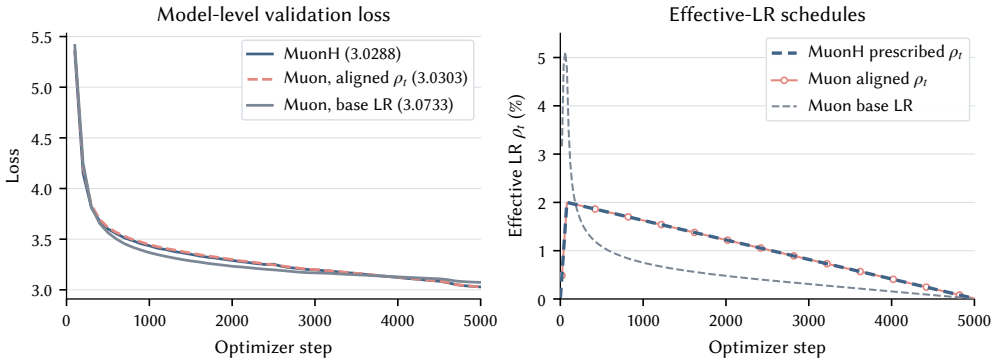


Figure 9: 170M BF16 comparison of MuonH, effective-LR aligned Muon, and ordinary-LR Muon. **Left:** validation loss. **Right:** effective LR traces for the selected MLP down-projection matrices. The training run with aligned-effective-LR and the normal Muon optimizer follows the predefined MuonH effective LR schedule. This run reaches a similar final loss, whereas ordinary-LR Muon produces a faster early decay of its induced effective LR and finishes with higher loss.

Effective LR provides a more informative descriptor of the loss trend. The experiments above show a close relationship between effective LR and loss dynamics. We further test whether effective LR provides a more informative descriptor than the raw optimizer learning rate. We apply the Multi-Power Law (MPL), a LR-schedule scaling law that predicts loss-curve shape from the LR schedule, to both the effective-LR schedule and the raw learning rate schedule (Luo et al., 2025b). For the two ordinary-Muon runs above, we fit the post-warmup validation curves using either the scalar learning rate or the induced effective LR, while holding out the final 20% of validation targets. As shown in Figure 16, using the effective LR reduces the mean held-out RMSE across the two runs from 0.0265 to 0.0210. The improvement is driven mainly by the base-Muon run; for effective-LR aligned Muon, the two LR signals are comparably predictive. This small diagnostic therefore supports effective LR as a more informative descriptor for schedule-dependent training dynamics. The MPL formulation and fitting results are provided in Section I.2.

Effective LR MPL helps interpret the late-stage crossover. In the selected 170M diagnostic, MuonH and effective-LR aligned Muon converge more slowly at the beginning but outperform ordinary Muon near the end of training. This pattern is consistent with the learning rate analysis in Section I. In particular, the MPL fit (Luo et al., 2025b) provides a simple diagnostic relationship: *a decrease in learning rate can induce an approximately proportional reduction in loss, with the loss decrease taking effect over subsequent training steps.* Applying this view to the effective LR, ordinary Muon decays its effective LR more aggressively early in training, which is consistent with its faster early loss reduction but leaves less effective LR decay for the later stage. MuonH and aligned Muon instead follow a more gradual, approximately linear effective LR decay, giving up some early convergence speed while retaining more decay toward the end of training. This view provides additional motivation for using a linear-decay schedule in our production design.

Together, these diagnostics motivate treating effective LR as the primary hyperparameter for MuonH-wrapped weights. They also suggest that the placement of its decay can affect where loss reduction occurs during training. We therefore next study the practical design of the LR schedule for the production run.

D.3.2 Learning Rate Schedule Design

Continual training motivates an open-ended schedule followed by a long terminal decay. In practice, the final token budget may not be known in advance (Hu et al., 2024; Shen et al., 2024b; Allal et al., 2025). We therefore prefer an open-ended schedule that can be extended before the final decay. To study how long this decay should be, we use WSD as a controlled setting and vary the effective peak LR and training horizon. Following the schedule-shape comparison above, we use linear decay in these experiments. For each setting, we sweep the decay ratio from 0.2 to 1.0 in increments of 0.2. Full experimental settings are provided in Section I.3.

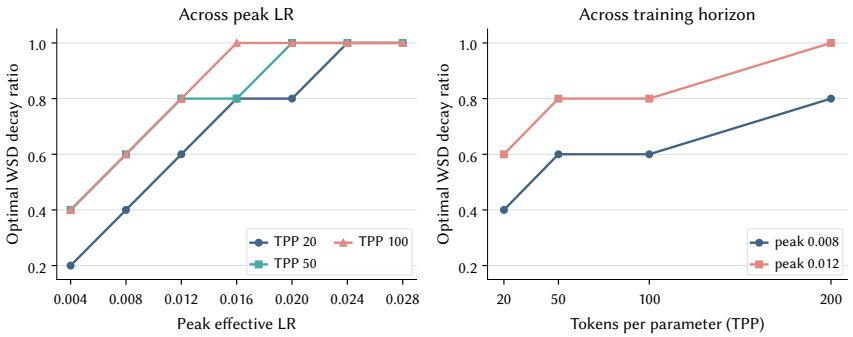


Figure 10: **Best WSD decay ratio on the evaluated grid. Left:** optimal WSD decay ratios gradually increase until saturated, as the peak effective LR increases across different TPPs. **Right:** optimal WSD decay ratios increase until saturated, as the TPP increases across peak learning rates.

Higher peak LRs favor longer decay. As shown in the left panel of Figure 10, the best tested decay ratio does not decrease as the effective peak LR increases from 0.004 to 0.028, at TPP 20, 50, and 100. At TPP 20, the best ratio increases from 0.2 to 1.0 over this peak-LR range; at TPP 50 and 100, it increases from 0.4 to 1.0. These results show that a larger peak LR generally favors a longer decay.

Longer training horizons also tend to favor longer decay. We observe a similar trend when increasing the training horizon. At effective peak LR 0.008, the best tested decay ratio increases from 0.4 at TPP 20 to 0.8 at TPP 200. At peak LR 0.012, it increases from 0.6 to 1.0.

Together, these observations motivate the production effective-LR schedule shown in the bottom panel of Figure 14. Here, we report the Hyperball weight LR, which is numerically equal to the effective LR for wrapped matrices, rather than the base LR. Phase 1 uses the open-ended power schedule in Equation (10), decreasing from approximately 5×10^{-2} to 1.04×10^{-2} over 438.8B. Its open-ended tail allows training to continue as the token budget or data distribution changes (Shen et al., 2024b). Phase 2 starts near 10^{-2} and applies a long linear decay over 960.0B, following the WSD observations that larger peak LRs and longer training horizons generally favor longer decay. Thus, the Phase 1 schedule supports continual training, while the WSD sweeps guide the decay range used in Phase 2.

D.4 Curriculum Model Averaging

When open-source datasets provide usable sample-level scores, we rank samples within each source and construct a mixture-preserving curriculum that places more preferred samples later in training. This ordering is designed to improve the data efficiency of higher-quality samples. However, a conventional terminal learning rate decay can work against this objective: *examples presented near the end of the curriculum are also processed when parameter updates have become small* (Luo et al., 2026). Curriculum Model Averaging (CMA) (Luo et al., 2026) addresses this conflict by combining an ascending data curriculum with constant-LR training and averaging late checkpoints. Our production recipe adopts this principle in Phase 2. We first follow the scheduled learning rate trajectory, then resume from a selected late checkpoint, hold the base learning rate fixed, and average six checkpoints from the continuation. The complete data and optimization flow is summarized in Figure 4,

with implementation details provided in Sections K.1 and K.3. We next describe the curriculum construction, the selected constant-LR continuation with checkpoint averaging, and the corresponding ablation results.

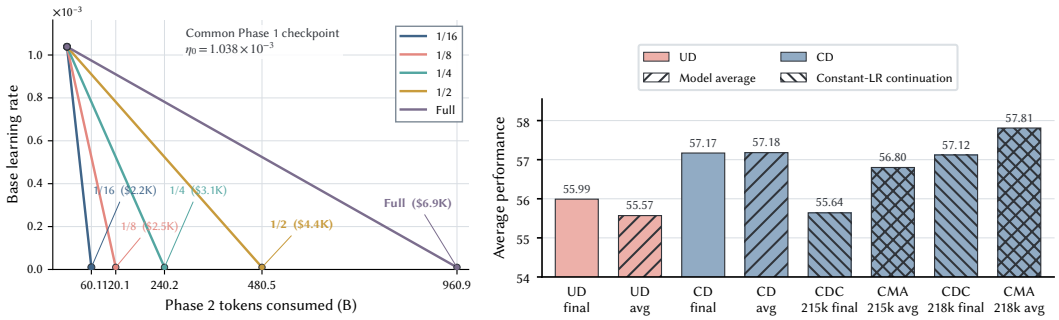
For convenience, we name the Phase 2 variants as follows. **Uniform–Decay (UD)** globally reshuffles the Phase 2 pool and follows the scheduled learning rate decay. **Curriculum–Decay (CD)** presents the same pool in component-local curriculum order while retaining the same LR decay. **CDC** adds a constant-LR continuation to CD and reports the final checkpoint. **Curriculum Model Averaging (CMA)** is the production variant: it averages the last several checkpoints from the CDC recipe.

The data curriculum arranges the order within each component while keeping the target mixture weight. We first establish a reproducible order separately for every retained component. If a component provides a usable sample-level quality signal, we sort its examples according to the source-specific interpretation of its score, placing less preferred score ranges earlier and more preferred ranges later. Components without a usable score instead follow a fixed random order. We then assign each example a normalized within-component rank according to its position in this ordered sequence, measured by cumulative token mass. Intuitively, this rank indicates which source-local score range an example falls into. For example, a rank of 0.25 means that approximately one quarter of the component's tokens appear earlier in its ordered sequence.

We divide the normalized rank range into 376 intervals, with each curriculum bucket formed by taking the corresponding interval from every component. Each bucket contains approximately 2.5B tokens, drawing roughly $1/376$ of tokens from every component and preserving the selected cross-component mixture. As training advances through the buckets, scored components move toward their more preferred score ranges, while unscored components advance through their fixed random orders. Since scores are interpreted only within their own sources and are not compared across components, this construction is not a global quality ranking. The scalable approximation used to construct the 376 buckets and the exact transition schedule are given in Sections K.1 and K.2 and Table 21.

The canonical CMA run uses the best-performing late-stage continuation with model averaging. We compare three late-stage training strategies: averaging checkpoints directly from the terminal-decay trajectory, and two constant-LR continuations resumed from steps 215,000 and 218,000. Among these configurations, the continuation from step 218,000 gives the strongest observed performance and is therefore selected for the production model. For this continuation, training resumes from step 218,000 with the base learning rate fixed at 4.08×10^{-5}. With the production Hyperball multiplier $m = 10$, the Hyperball weight LR is therefore 4.08×10^{-4}, which is also the effective LR for the MuonH-wrapped matrices. During the late stage of this continuation, we save checkpoints at approximately regular intervals and form the released model by equally averaging the parameters of six such checkpoints. Neighboring checkpoints are typically separated by 100 optimizer steps, or approximately 0.63B tokens under the production batch configuration. Full resume, checkpoint-spacing, and averaging details are provided in Section K.3.

Ablation study on curriculum, model averaging, and constant-LR continuation. We conduct ablation experiments to examine curriculum ordering, constant-LR late-stage training, and model averaging. All runs use the same Phase 2 data pool and evaluation benchmarks. The results are summarized in Figure 11b, and the continuation and averaging settings are provided in Section K.3. First, *curriculum ordering improves over uniform ordering* by 1.18 points without model averaging (CD: 57.17 versus UD: 55.99) and by 1.61 points when comparing the model averages of the corresponding decay trajectories (CD model average: 57.18 versus UD model average: 55.57). Since UD and CD use the same data pool and LR schedule, these comparisons show that curriculum ordering itself contributes to the improvement. Second, *model averaging alone gives little benefit under LR decay*: it changes the UD endpoint by -0.42 points and the CD endpoint by $+0.01$ points without constant-LR continuation. Finally, the two CDC controls score 55.64 and 57.12 for the 215k and 218k continuations, respectively, when evaluated at their unaveraged final checkpoints. With six-checkpoint averaging, the corresponding branches reach 56.80 and 57.81. Model averaging therefore improves the two constant-LR continuations by 1.16 and 0.69 points, respectively, in contrast to its limited benefit under LR decay. This suggests that model averaging is more useful when the model continues learning from the late high-quality data with a constant LR. We therefore select



(a) The five independent Phase 2 base learning rate schedules used in the UD points of Figure 2b. All runs load the same Phase 1 checkpoint and therefore share the same starting base rate. They differ only in decay horizon and all decay linearly to 10^{-5} for base LR.

(b) Endpoint evaluation scores for eight Phase 2 configurations. *final* means using the final checkpoint, and *avg* means computing the checkpoint average as discussed in Section K.3. CDC/CMA *215k* and *218k* mean continual training starting from step 215k and 218k. The vertical axis is the same 15-benchmark performance defined in Table 2. The underlying endpoint scores are reported in Table 20.

Figure 11: **Phase 2 budget schedules and endpoint comparison.** The schedule panel gives the common Phase 1 starting point and the independent UD Phase 2 horizons. The endpoint panel compares the UD, CD, CDC, and CMA configurations.

the 218k averaged branch as the production CMA endpoint. It combines curriculum ordering, late constant-LR continuation and six-checkpoint averaging.

Part of the observed advantage of CD over UD ordering could be attributed to benchmark contamination in the later-stage data, as discussed in Section B. But the remaining advantage is observed after post-training in Section 3.3, and the contamination alone is unlikely to explain the persistent improvement. We report the exact base LR and Hyperball effective LR, checkpoint IDs, and the model-export procedure in Section K.3.

Cost-efficiency estimation. We further use the Puro cost scaling law to estimate the overall efficiency improvement. Relative to UD, the fitted scaling law estimates cost-efficiency factors of approximately $1.65\times$ for CD and $2.40\times$ for the full CMA recipe (Section J). These values are estimated from the fitted cost–performance relation at matched performance.

D.5 Post-Training Recipe

This section specifies the supervised fine-tuning procedure used to compare the Uniform–Decay (UD) and Curriculum Model Averaging (CMA) checkpoints. The motivation and interpretation of this comparison are discussed in Section 3.3. This section focuses on the construction, controls, and evaluation protocol required to reproduce the experiments.

Data construction and matched controls. For each materialized SFT dataset, we initialize one run from the UD checkpoint and one run from the CMA checkpoint. Within each data-order repetition, the two runs use identical examples, example order, tokenizer, loss mask, optimizer, learning rate schedule, and training horizon. The initialization checkpoint is therefore the only changed factor in each paired comparison.

We use MuonH for SFT, with a cosine base learning rate schedule from 1×10^{-5} to 1×10^{-7}. Matrix parameters managed by MuonH use a $10\times$ multiplier for the Hyperball weight learning rate. The GSM8K-based experiment uses a global batch size of 160. The batch configurations for the other two settings are reported in Section H.

Evaluation protocol. Each UD–CMA pair is evaluated at the same predefined training steps. The two mathematics-oriented settings use `lm-evaluation-harness` on GSM8K with a fixed ChatML prompt and greedy decoding. The Tulu-3 mixed-domain setting uses the adapted Open-

Compass protocol for the 15-task suite and evaluates instruction following separately with IFEval. The answer-extraction rules, task-level aggregation, and checkpoint steps are specified in Section H.

D.6 Data Recipe

Domain	Phase 1		Phase 2	
	Tokens	Share	Tokens	Share
English	321.31B	73.2%	557.52B	59.4%
Math	31.56B	7.2%	171.29B	18.3%
Chinese	51.34B	11.7%	88.48B	9.4%
Code	34.62B	7.9%	108.11B	11.5%
SFT / instruction	–	0.0%	12.66B	1.3%
Total	438.84B	100.0%	938.06B	100.0%

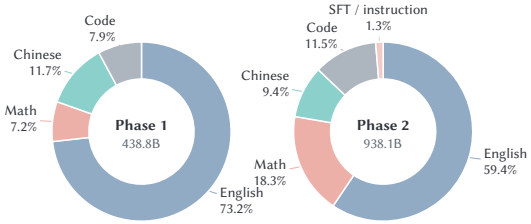


Figure 12: Domain composition of the two pretraining phases. The left table reports materialized token counts and shares for Phase 1 and Phase 2, and the right panel visualizes the same composition. Phase 1 emphasizes broad coverage, while Phase 2 allocates a larger share to mathematics and introduces instruction-like data. The Phase 2 token budget excludes Phase 1 replay. See Section D.6.2 for details.

Our goal is to select a compact set of high-quality and diverse data sources while preserving broad domain coverage. Candidate datasets differ in both quality and capability profile, so source-level metadata alone is not sufficient for data selection. We therefore use *proxy benchmarking* to compare candidate data slices under a shared protocol. As illustrated in Figure 13, each run starts from the same Qwen3-0.6B checkpoint and follows the same continuation-training schedule. After continuation training, we evaluate the resulting checkpoint on a fixed 15-benchmark suite. The score vector from this evaluation serves as the capability profile of the candidate slice. These score vectors can guide our data recipe heuristically. The following three paragraphs describe how we construct fine-grained slices, select and allocate data sources, and balance different capabilities in the final mixture. Detailed settings are provided in Section L.

D.6.1 How Do We Select a Good Data Recipe?

Fine-grained data slice selection. We construct candidate data slices according to source size and the availability of sample-level quality scores. We apply three sampling rules. First, if a dataset contains more than 50B tokens and provides usable score labels, we sort its examples by score and sample four local slices around the 0th, 25th, 50th, and 75th percentile positions. Second, for datasets between 5B and 50B tokens, as well as larger datasets without usable score labels, we randomly sample one 4B-token slice. Third, we do not evaluate the remaining smaller datasets. For each large scored source, the four slices form a within-source quantile benchmark. Because these slices come from different positions in the same score ordering, they reveal quality differences that a single source-level result would hide. For example, the `q00` slice of DCLM-Dedup ranks third in average proxy performance, while its `q25` slice ranks eleventh. This result shows that different regions of the same source can provide substantially different training value.

Feature-guided data selection. We use the measured capability profiles to guide both source selection and data allocation. MegaMath-Web-Pro performs strongly on Math and General, while MegaMath-Code provides the strongest Code signal among the measured candidates. We therefore assign both sources larger shares in Phase 2 to strengthen general and target capabilities. The quantile results also guide within-source filtering. Because the top-scoring region of DCLM-Dedup performs much better than its lower-ranked regions, we retain 34B tokens from this region in Phase 2. This selection corresponds to approximately the top 5% of DCLM-Dedup. FineWeb-Edu-CN ranks highest on Chinese capability, so we also increase its share to preserve Chinese performance. The proxy results provide useful evidence for increasing, reducing, or filtering different data sources. Here, we take heuristics from these capability profiles and still choose the concrete mixing ratios manually.

Capability balance across domains. The proxy results show that different datasets have substantially different capability profiles. A source that performs well on one group of benchmarks may

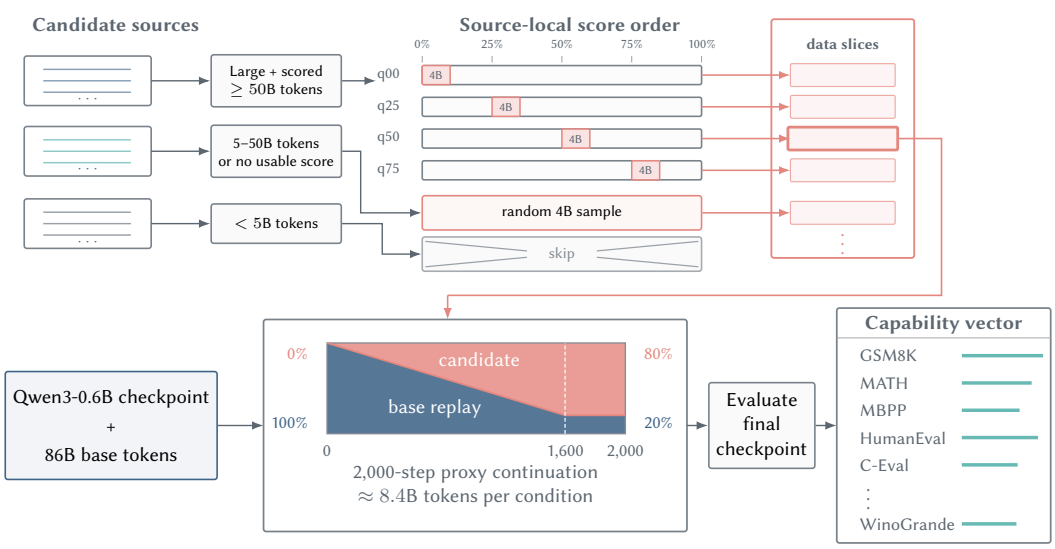


Figure 13: Proxy benchmarking protocol. Candidate slices are first constructed according to corpus size and the availability of sample-level quality scores. Large scored sources contribute four within-source quantile slices, while other eligible sources contribute one random 4B-token slice. Each retained slice is used for a 2,000-step continuation run from the same Qwen3-0.6B checkpoint. During continuation training, the candidate-data ratio gradually rises to 80%, while replay from the base mixture decreases accordingly. The final checkpoint is evaluated on the same 15-benchmark suite to obtain a capability vector. These vectors guide the fine-grained slice selection, source allocation, and capability balancing described below.

perform less well on others. For example, FineWeb-Edu-CN ranks first on Chinese capability but is weaker on Math, Code, and General (Section L). We therefore need to balance broad general knowledge with more specialized reasoning capabilities rather than maximize a single benchmark dimension. To make these trade-offs more explicit, we apply PCA to the capability vectors. As shown in Figure 18, General aligns mainly with PC1, while Code points in the opposite direction. Math aligns mainly with PC2 and remains slightly positive on PC1. In this projection, Code shows a stronger trade-off with General than Math does (Section L). This observation informs the roles of the two pretraining phases. Phase 1 preserves broad coverage, with English accounting for 73.2% of 438.8B (Table 4 and fig. 12). Phase 2 then adds more specialist data, but allocates a slightly larger share to mathematics than to code. In this way, the two-phase recipe strengthens the target reasoning capabilities while retaining broad general coverage.

D.6.2 How Do We Preprocess Data and Reproduce the Shards?

Data preprocessing pipeline. The pipeline turns the candidate collection into fixed training shards in four stages. *Within-source deduplication* removes repeated documents from web components such as DCLM-Dedup and FineWeb-Edu-EN. We do not deduplicate across sources because our analysis found little additional reduction in data volume. *Proxy benchmarking* evaluates each eligible candidate source or within-source slice under the shared protocol described above and produces a multidimensional capability profile. *Heuristic filtering* combines these profiles with domain coverage, language balance, and token-budget constraints to determine the retained portion and mixture weight of each source. *Final materialization* freezes the source revisions, retention rules, mixture weights, and random seeds. It then merges the retained components into fixed shards that are read sequentially during training, without online mixing or sampling. In addition, we also use PreSelect fastText (SHUM et al., 2025) to evaluate the Nemotron-HQ (Mahabadi et al., 2025) and Nemotron-HQ-Synthetic datasets. Apart from this step, we do not perform additional compute-intensive data scoring and primarily rely on existing quality scores.

Kaiyuan-Spark preprocessing framework. We implement the recipe with Kaiyuan-Spark, a Spark-based data preprocessing framework (Luo et al., 2025a). The framework is designed for effi-

cient large-scale data processing. For compute-intensive operations such as MinHash deduplication, it uses native C++ kernels through the Chukonu integration. This implementation is inherited from Chukonu, whose original Spark benchmark reports an approximately $2.5\times$ speedup over the corresponding JVM implementation (Yu et al., 2021). The framework also supports the curriculum bucket materialization required by the Phase 2 design. Given a fixed component pool and mixture, it assigns normalized progress within each component, aligns the resulting slices into token-sized buckets, and materializes their order using recorded random seeds. The same component pool can therefore produce either an ordered curriculum stream or a uniformly ordered stream by changing only the materialization order. These two data streams support the UD, CD, and CMA recipes in Phase 2 (Section D.4).

Two-phase data and transition schedule. All training data are materialized as static shards and manifests before training. Production runs therefore do not rely on online scoring or dynamic mixture sampling. The Phase 1-to-Phase 2 transition spans approximately 43.9B training tokens. During this transition, the sampling weight of the Phase 1 replay mixture decreases *linearly*, while that of the Phase 2 mixture increases *linearly*. The transition therefore consumes approximately 21.9B tokens from the Phase 1 replay data and 21.9B tokens from the beginning of the Phase 2 data pool. The full Phase 2 component pool contains approximately 938.1B tokens, including the 21.9B tokens consumed during the transition. After the transition, the remaining 916.1B Phase 2 tokens are consumed without Phase 1 replay. Counting the transition, the Phase 2 training stage consumes approximately 960.0B tokens in total: 21.9B tokens of Phase 1 replay and 938.1B tokens from the Phase 2 pool. Together with Phase 1 training, the complete pretraining run consumes approximately 1.4T tokens.

E Reproduction-Cost Assumptions

Table 5: Reference rental rates used to convert reported or estimated accelerator usage to reproduction cost. The main figures report USD; RMB values are retained for ledger traceability and use an exchange rate of USD:CNY=6.8067 as of 1 July 2026.

Reference accelerator	USD/GPU-hour	RMB/GPU-hour
H200	4.00	27.23
H100	3.25	22.12
A100 80GB	1.79	12.18
A100 40GB	1.29	8.78
A800 80GB	1.79	12.18
RTX 5090	0.31	2.08

This section specifies how every coordinate and the comparison-model frontier in Figure 1 are constructed. The plotted points are determined by the benchmark scores, public source disclosures, and cost-accounting rules described below.

Construction of the cost–performance figure. The coordinates of different models are summarized in Table 6. Our Puro-2B includes a UD ($4.4K) endpoint and the canonical CMA ($6.9K) endpoint. The cost accounting of these two runs is summarized in Table 19. More complete results of the Puro Cost Scaling Law, including different UD training variants and CD/CMA endpoints, are summarized in Figure 11b. The implementation of CMA can be found in Figure 4. The checkpoint lineage and scale-down run details of Puro Cost Scaling Law are provided in Tables 19 and 20, and the fitted scaling law can be found in Figure 2b.

Performance coordinate. For model m, the vertical coordinate is the unweighted arithmetic mean

$$P_m = \frac{1}{15} \sum_{b \in \mathcal{B}} s_{m,b}, \tag{6}$$

where $s_{m,b}$ is the percentage score on benchmark b. The benchmark set $\mathcal{B}$ contains GSM8K, MATH, sanitized-MBPP, HumanEval, MMLU, MMLU-Pro, ARC-Challenge, ARC-Easy, BoolQ, CommonsenseQA, HellaSwag, PIQA, SocialIQA, WinoGrande, and BBH. No benchmark is weighted by its number of examples. All 15 scores must be present for a plotted model.

Table 6: Cost and performance coordinates used in Figure 1. GPU-hours and costs are rounded to the nearest unit for presentation; USD costs use the exchange rate in Table 5, and the figure uses the unrounded coordinates. Avg score is the unweighted mean over the 15 tasks used by the figure.

Model	Evidence	Price basis	GPU-hours	Cost (RMB)	Cost (USD)	Avg score
Qwen2-1.5B	6ND, model tokens	H100 equivalent	25,939	573,817	84,302	55.14
Qwen2.5-1.5B	6ND, family tokens	H100 equivalent	66,700	1,475,530	216,776	60.73
Qwen3-1.7B-Base	6ND, model tokens	H100 equivalent	147,261	3,257,664	478,597	65.27
Gemma-2-2B	6ND, model tokens	H100 equivalent	12,560	277,860	40,821	48.60
Gemma-3-1B-PT	6ND, model tokens	H100 equivalent	4,812	106,460	15,640	25.34
Llama-3.2-3B	Reported GPU-hours	H100	460,000	10,176,017	1,495,000	52.17
LFM2.5-1.2B-Base	6ND, family tokens	H100 equivalent	78,828	1,743,808	256,190	53.54
Falcon-H1-Deep-1.5B-Base	6ND, family tokens	H100 equivalent	11,189	247,519	36,364	63.53
Instella-3B	6ND, model tokens	H100 equivalent	30,851	682,470	100,265	55.20
OLMoE-1B-7B-0125	Reported count/time	H100	61,440	1,359,162	199,680	48.70
Yulan-Mini-2.4B	6ND, reported MFU	A100-80GB rate	27,073	329,857	48,461	58.80
SmolLM3-3B-Base	Reported count/time	H100	221,184	4,892,983	718,848	65.85
MobileLLM-R1-950M-base	Reported count/time	A100-80GB rate	36,864	449,151	65,987	54.23
PURO-2B	Measured GPU-hours	RTX 5090	22,514	46,905	6,891	57.81

Cost evidence hierarchy. We assign the horizontal coordinate (reproduction cost) using the following ordered rules. Reported monetary cost is used directly when it is available. Otherwise, reported GPU-hours are converted using the corresponding RMB/GPU-hour rate in Table 5. If a report gives only GPU count G and elapsed training time T, we first compute $H = GT$ GPU-hours, with $T = 24d$ for a duration of d days. In all these cases, the converted cost is

$$C_m = H_m r_a, \tag{7}$$

where r_a is the reference RMB/GPU-hour price for accelerator a. The audit ledger retains this RMB value; Figure 1 divides it by 6.8067 to report the horizontal coordinate in USD.

If neither monetary cost nor accelerator-hours are reported but a usable training-token count is available, we estimate training compute with $C = 6ND$ (Kaplan et al., 2020; Hoffmann et al., 2022). For parameter count N, token count D, accelerator BF16 peak throughput F_a in TFLOP/s, and model FLOPs utilization (MFU) η, the estimate is

$$H_m = \frac{6N_m D_m}{3600 \times 10^{12} \times \eta F_a}, \qquad C_m = H_m r_a. \tag{8}$$

For token-only entries, we use an H100-equivalent estimate with $F_a = 989.5$ TFLOP/s and $\eta = 0.70$. Model-specific token counts are preferred; when only a family-level training budget is disclosed, that use is marked as an estimate in the coordinate table. Models lacking both accelerator usage and a usable token count receive no cost coordinate and are omitted from the figure. As we posit an ideal MFU assumption in our accounting, we estimate the cost lower bound of reproducing our counterpart models in most cases.

Public sources for Table 6. The Qwen rows use the public training-token disclosures in the Qwen2, Qwen2.5, and Qwen3 technical reports (Yang et al., 2024a;b; 2025a). Gemma-2-2B and Gemma-3-1B-PT use Google's Gemma report and model-card disclosures for the corresponding model sizes and token budgets (Rivière et al., 2024; Google, 2024; Gemma Team, 2025; Google, 2025). Llama-3.2-3B uses the official Llama 3.2 model card, which reports the 3B checkpoint's H100 GPU-hours directly (Meta AI, 2024c). LFM2.5-1.2B-Base uses Liquid AI's LFM2.5 release note for the disclosed family-level training budget (Liquid AI, 2026). The Falcon-H1-Deep-1.5B-Base row uses the Falcon-H1 technical report for the token budget and H100 training pool disclosure (Zuo et al., 2025). Instella-3B uses the Instella report and model card for the two base-pretraining stages (Liu et al., 2025; AMD, 2025). OLMoE-1B-7B-0125 uses the OLMoE paper's H100 count and elapsed-time disclosure (Muennighoff et al., 2025). SmolLM3-3B-Base uses the Hugging Face training report for the reported H100 count and duration (Bakouch et al., 2025).

Special cases. Our PURO-2B point uses measured active-training time rather than $6ND$: 6,009.46 GPU-hours for Phase 1 plus 16,504.95 GPU-hours for Phase 2. Its local RTX 5090 rate is $12{,}000/(8 \times 30 \times 24) = 2.0833$ RMB/GPU-hour, giving $22{,}514.41 \times 2.0833 = 46{,}905.03$ RMB,

or $6,891. Yulan-Mini-2.4B uses Equation (8) with its reported 51.57% MFU and the 312-TFLOP/s A800 BF16 peak (Yiwen et al., 2025). Its GPU-hours are priced with the unified A100 80GB rental rate in place of the rarely published public A800 rental price. MobileLLM-R1 reports 128 GPUs but not their model (Meta AI, 2025). We use the A100 80GB GPUs disclosed by the original MobileLLM training-cost report as the MobileLLM-series reference (Meta AI, 2024a). Two 4–5 day pretraining phases and two 1–2 day mid-training phases imply 30,720–43,008 accelerator-hours. We price this range with the A100 80GB rate and use the midpoint as its plotted coordinate.

Pareto frontier. To construct the dashed comparison frontier, we first remove PURO-2B. A comparison model m is Pareto-optimal if there is no other comparison model j such that

$$C_j \leq C_m, \qquad P_j \geq P_m, \tag{9}$$

with at least one strict inequality. Operationally, at an identical cost we retain only the highest-scoring point, sort the remaining points by increasing cost, and retain a point only when its score is strictly larger than every lower-cost score seen so far. The retained comparison points are connected in cost order.

F Evaluation and Production Training Details

F.1 Baselines, Benchmarks, and Evaluation Protocol

Baseline models. We compare PURO-2B with language models of comparable size, primarily in the 1B–3B parameter range. For a fair comparison, we consistently evaluate their pretrained or base versions. We divide the baselines into two groups according to the scope of their public releases.

Open-weight models release their weights but not the complete training data or recipe. We include Qwen2-1.5B, Qwen2.5-1.5B, Qwen3-1.7B-Base, and Qwen3.5-2B-Base from the Qwen series (Yang et al., 2024a;b; 2025a; Qwen Team, 2026). The Gemma baselines are Gemma-2-2B, Gemma-3-1B-PT, and Gemma-4-E2B-Base (Rivière et al., 2024; Gemma Team, 2025; 2026). We also evaluate Llama-3.2-3B (Meta AI, 2024b), LFM2.5-1.2B-Base (Liquid AI, 2025), and the standard and deeper 1.5B base variants of Falcon-H1 (Zuo et al., 2025).

Open-recipe models additionally release training data and training artifacts to support studying or reproducing the training process. We include Instella-3B (Liu et al., 2025), OLMoE-1B-7B-0125 (Muennighoff et al., 2025), Yulan-Mini-2.4B (Yiwen et al., 2025), SmolLM3-3B-Base (Bakouch et al., 2025), and MiniCPM5-1B-Base (OpenBMB Team, 2026).

Evaluation benchmarks. Our evaluation covers 15 benchmarks in three groups: mathematics, code, and general reasoning and knowledge. For mathematics, we use GSM8K (Cobbe et al., 2021) and the original MATH benchmark (Hendrycks et al., 2021b). For code generation, we use sanitized-MBPP (Austin et al., 2021) and HumanEval (Chen et al., 2021). For reasoning and knowledge, we use MMLU (Hendrycks et al., 2021a), MMLU-Pro (Wang et al., 2024b), ARC-Challenge and ARC-Easy (Clark et al., 2018), BoolQ (Clark et al., 2019), CommonsenseQA (CSQA) (Talmor et al., 2019), HellaSwag (Zellers et al., 2019), PIQA (Bisk et al., 2020), SocialIQA (SIQA) (Sap et al., 2019), WinoGrande (Sakaguchi et al., 2020), and Big-Bench Hard (BBH) (Suzgun et al., 2023). Together, these benchmarks cover mathematics, coding, academic knowledge, reading comprehension, and commonsense reasoning.

Evaluation protocol. We evaluate every model with OpenCompass (Contributors, 2023), fixing the checkpoint and tokenizer revisions, dataset snapshots, benchmark-specific prompts, few-shot examples, generation settings, and answer postprocessors. GSM8K, MATH, sanitized-MBPP, HumanEval, MMLU-Pro, and BBH use generation-based (GEN) evaluation: the model produces a free-form response, from which the final answer or executable code is extracted. We use greedy decoding with fixed maximum output lengths and stopping rules. The remaining nine benchmarks use perplexity-based (PPL) evaluation, which ranks candidate answers by their token log-likelihoods. For GSM8K, a shared deterministic postprocessor prioritizes explicit final-answer markers and boxed answers, preserves signs and fractions, and uses the final numeric expression only when no explicit answer is present.

Score aggregation. For tasks supporting both cloze formulation (CF) and multiple-choice formulation (MCF), we evaluate both formulations for each model and use the better-performing formulation when computing the reported aggregate, following the OLMES (Gu et al., 2025) convention. This selection is performed independently for each model. All scores in the performance table are percentages. Throughout the performance, cost and scaling analyses, overall performance P is the unweighted arithmetic mean of all 15 benchmark scores in Table 2.

F.2 Production Training Details

Production run setup. Our canonical Puro-2B run consists of two phases. Both phases use sequence length 4,096, global batch size 1,536, and micro-batch size 2. Selected matrix weights use MuonH with zero weight decay, while the remaining parameters use AdamW with weight decay 0.1; both phases use blockwise E4M3 FP8. The LR schedule below corresponds to the base learning rate, and the Hyperball weight LR is 10 times the base learning rate. Phase 1 trains on 438.8B with 24 GPUs and reaches validation loss 2.730; its median measured throughput is 238 TFLOP/s per GPU. From Phase 1 to Phase 2, we extend our compute resources, from 24 to 96 GPUs. Phase 2 resumes from this checkpoint, consumes 960.0B and continues to 1.4T cumulative tokens, and ends at validation loss 2.488 with median throughput 192 TFLOP/s per GPU. The learning rate decays from 5×10^{-3} to 1.04×10^{-3} in Phase 1 and from 1.04×10^{-3} to 1×10^{-5} in Phase 2, matching the power-then- linear schedule shown in Figure 14. The Phase 2 token ledger is summarized in Table 19; the exact transition and pool accounting is frozen in Table 21. The materialized Phase 2 pool already includes the early transition range, while Phase 1 replay is an additional consumed stream.

Table 7: The resulting audit table combines the GPU cluster, configuration, base learning rate schedule, and throughput.

Phase	Phase tokens	Cumulative	GPUs	TP/PP/DP	LR schedule	TFLOP/s/GPU
Phase 1	438.84B	438.84B	24	1/2/12	$5.00 \times 10^{-3} \rightarrow 1.04 \times 10^{-3}$ (power)	238
Phase 2	959.99B	1.40T	96	1/4/24	$1.04 \times 10^{-3} \rightarrow 1.00 \times 10^{-5}$ (linear)	192

Table 8: Optimizer parameter groups used in both production pretraining phases. Learning rate entries are relative to the base schedule.

Parameter group	Optimizer	Weight decay	Learning rate
Selected attention and MLP matrices	MuonH	0	10× base schedule
Embeddings, normalization layers, LM head, and other parameters	AdamW	0.1	Base schedule

Phase 1 learning rate schedule. The scalar learning rate is the base rate. The MuonH-controlled matrix groups apply an optimizer multiplier of 10, while norm-variant parameters use their separately configured scalar optimizer. With optimizer step k, the Phase 1 base schedule is

$$\eta_{\text{base}}(k) = \begin{cases} 5 \times 10^{-3}\, k/1000, & 0 \le k \le 1000, \\ 5 \times 10^{-4} + 4.5 \times 10^{-3} \left(1 + \dfrac{k - 1000}{1000}\right)^{-1/2}, & k > 1000. \end{cases} \tag{10}$$

Thus $p = 1/2$, $\tau = 1{,}000$ steps, warmup is 1,000 steps (1,536,000 samples), and the 5×10^{-4} minimum is an asymptotic floor. The power decay shape follows the power schedule (Shen et al., 2024b) and supports continual training in the first phase.

Phase 2 learning rate schedules. Each Phase 2 run loads the same Phase 1 endpoint and therefore starts from the same base learning rate, namely the terminal Phase 1 value $\eta_0 = 1.04 \times 10^{-3}$. We reset the Phase 2-local schedule coordinate at that shared checkpoint, so $s = 0$ always means the Phase 1 endpoint, not the run's recorded global optimizer step. Thus, for every run j, $\eta_{\text{base},j}(0) = \eta_0$; the runs differ only in their linear decay-sample budgets. With decay-sample budget S_j and terminal rate η_{min}, the base schedule is

$$\eta_{\text{base},j}(s) = \eta_0 + (\eta_{\text{min}} - \eta_0)\, \frac{s}{S_j}, \qquad 0 \le s \le S_j. \tag{11}$$

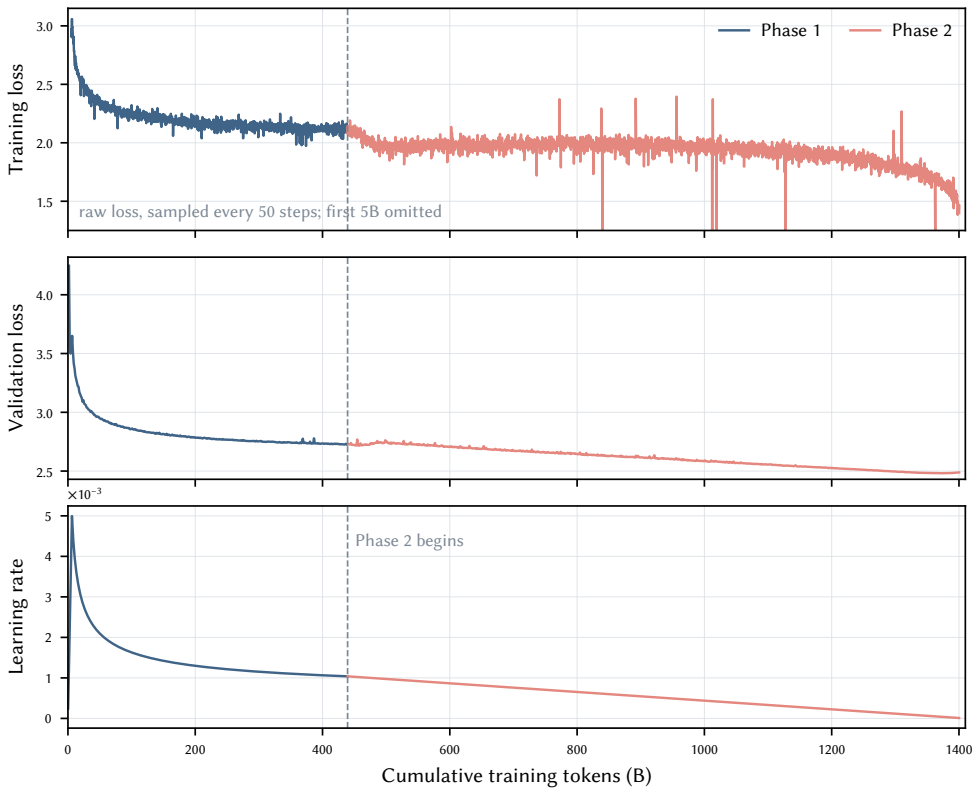


Figure 14: Concatenated production traces. Training loss is raw (not smoothed) and sampled every 50 optimizer steps; the first 5B tokens are omitted from that panel to keep the later trajectory legible. Every logged validation event is retained. The vertical line marks the Phase 1/Phase 2 run boundary; the Phase 2 run begins with the distribution transition. The learning rate schedule corresponds to the base LR and the Hyperball weight LR is 10 times base LR.

Here s is Phase 2 samples consumed after the common checkpoint. As in Phase 1, MuonH-controlled matrix groups multiply each base rate by 10. The five schedules share the same starting rate and $\eta_{\min} = 10^{-5}$, but have different horizons of approximately 60.1, 120.1, 240.2, 480.5, and 960.9B tokens, as shown in Figure 11a. These are the UD scaling points; the matched CD and CMA endpoints use the full Phase 2 budget. Our canonical CMA run uses the curriculum order and replaces the LR decay in the final steps (starting from step 218,000, lasting the last 29B tokens) of pretraining with constant-LR training. The details can be found in Section K.

G MuonH Scaling-Ladder Analysis

Experimental setup. We construct a five-model ladder spanning model scales from 0.17B to 1.7B. Every model is trained from scratch for 20 tokens per scaling parameter (TPP=20), following the compute-optimal token-to-parameter ratio reported in prior work (Hoffmann et al., 2022). All runs use the same data distribution, tokenizer, sequence length of 4,096, global batch size of 512, validation set, and linear learning rate decay with a 1% warmup. The validation set uses a subset of the Nemotron-CC corpus (Su et al., 2025; Basant et al., 2025). The exact architecture and token budgets are shown in Table 10. We compare two paired recipes: (i) blockwise-FP8 MuonH against a tuned blockwise-FP8 Muon baseline to evaluate the complete optimizer recipes, and (ii) blockwise-FP8 MuonH against BF16 MuonH to isolate numerical precision. MuonH uses a base LR of 10^{-2} and a Hyperball multiplier of 2, hence a Hyperball weight LR of 2×10^{-2}. The updated ordinary-Muon baseline fixes weight decay at 0.8 and uses ordinary LRs 5×10^{-3}, 4×10^{-3}, 4×10^{-3}, 3×10^{-3}, and 3×10^{-3} from 0.17B through 1.7B. The 0.17B and 0.33B choices are anchored by the reported LR/weight-decay searches; the larger points are conservative size-scaled extrapolations. The search results that motivate the first two LR/weight-decay anchors are tabulated in Table 9. The

two precision runs use identical MuonH hyperparameters. Because the optimizer pair uses tuned, non-identical learning rate schedules, this is a complete-recipe comparison rather than an isolated hyperball-constraint ablation. These two experiment results are summarized in Figure 5.

Table 9: Completed LR–weight-decay search results used to select the updated blockwise-FP8 Muon baseline anchors. The search stages are labeled initial or extended. Bold rows mark the best completed setting at each anchor size.

Model	Search stage	Ordinary LR	Weight decay	Final validation loss
0.17B	initial	2×10^{-3}	0.1	3.413030
0.17B	initial	2×10^{-3}	0.2	3.406015
0.17B	initial	2×10^{-3}	0.4	3.391572
0.17B	initial	2×10^{-3}	0.8	3.375170
0.17B	initial	3×10^{-3}	0.1	3.414363
0.17B	initial	3×10^{-3}	0.2	3.405061
0.17B	initial	3×10^{-3}	0.4	3.383846
0.17B	initial	3×10^{-3}	0.8	3.361907
0.17B	initial	4×10^{-3}	0.1	3.413549
0.17B	initial	4×10^{-3}	0.2	3.398726
0.17B	initial	4×10^{-3}	0.4	3.381147
0.17B	initial	4×10^{-3}	0.8	3.357248
0.17B	extended	4×10^{-3}	1.6	3.352910
0.17B	**extended**	$\mathbf{5 \times 10^{-3}}$	**0.8**	**3.349810**
0.17B	extended	5×10^{-3}	1.6	3.360720
0.17B	extended	6×10^{-3}	0.8	3.356287
0.17B	extended	6×10^{-3}	1.6	3.370632
0.33B	initial	2×10^{-3}	0.2	3.156636
0.33B	initial	2×10^{-3}	0.4	3.142686
0.33B	initial	2×10^{-3}	0.8	3.122845
0.33B	initial	3×10^{-3}	0.2	3.150765
0.33B	initial	3×10^{-3}	0.8	3.116963
0.33B	extended	3×10^{-3}	1.6	3.117189
0.33B	**initial**	$\mathbf{4 \times 10^{-3}}$	**0.8**	**3.114583**
0.33B	extended	4×10^{-3}	1.6	3.130430
0.33B	initial	5×10^{-3}	0.8	3.120008
0.33B	extended	5×10^{-3}	1.6	3.139534

Compute-equivalent fitting. For each run, we use the standard dense-transformer approximation

$$C = 6ND, \tag{12}$$

where D is the number of training tokens and $N = D/20$ is the scaling parameter count implied by the ladder construction. To compare a baseline b and variant v, we jointly fit the ten observations with

$$L_g(C) = L_\infty + A_g \left(\frac{C}{C_0} \right)^{-\alpha}, \qquad g \in \{b, v\}, \tag{13}$$

sharing L_∞ and α while allowing a group-specific amplitude. The resulting horizontal multiplier is

$$\kappa_{v \leftarrow b} = \left(\frac{A_b}{A_v} \right)^{1/\alpha}. \tag{14}$$

Thus, the variant requires $C/\kappa_{v \leftarrow b}$ to match the fitted baseline loss at compute C. This restricted shared-shape fit estimates a horizontal efficiency shift; it is not used to select a learning rate schedule or to predict the absolute loss of the 2B production run.

As shown in Section 3.2, MuonH has a fitted compute-equivalent multiplier of $1.19\times$ relative to the updated Muon baseline. Leaving out one model size at a time gives 1.17–$1.28\times$, showing that the estimate is not determined by one endpoint. Applying this horizontal factor to the 1.68×10^{22}-FLOP budget of a 2B/1.4T-token run gives 1.41×10^{22} FLOPs at matched Muon loss, corresponding to

Scale	L	d_{model}	d_{ff}	H_Q	H_{KV}	d_{ch}	d_{QK}	d_V	MBS	Steps	Tokens	$C = 6ND$
0.17B	28	512	1536	8	8	64	128	128	4	1700	3.57B	3.81×10^{18}
0.33B	28	768	2304	16	8	64	128	128	4	3100	6.50B	1.27×10^{19}
0.60B	28	1024	3072	16	8	128	128	128	2	5400	11.32B	3.85×10^{19}
1.09B	28	1536	4608	16	8	128	128	128	1	10200	21.39B	1.37×10^{20}
1.70B	28	2048	6144	16	8	128	128	128	1	16400	34.39B	3.55×10^{20}

Table 10: Architecture and training setup of the MuonH scaling ladder. All models use a sequence length of 4,096, global batch size 512, 28 layers, and 20 training tokens per scaling parameter. H_Q and H_{KV} denote query and grouped-query KV heads; d_{ch} is the configured attention channel width (`kv_channels`), and d_{QK} and d_V are the configured QK and value head dimensions.

16.1% less compute. This transfer assumes that the optimizer's horizontal efficiency factor persists beyond the 20-TPP ladder; we therefore report it as an estimated compute-equivalent saving rather than an absolute large-run loss prediction. The leave-one-out range measures sensitivity to individual ladder points, not uncertainty from the selected functional form.

Precision and throughput decomposition. To compare training cost without including evaluation, checkpoint, or experiment-tracking intervals, we convert theoretical compute to GPU-hours using

$$H = \frac{C}{3600 \times 10^{12} \times \bar{p}}, \tag{15}$$

where $\bar{p}$ is the median per-GPU throughput in TFLOP/s over training-step history. Because H is measured in aggregate GPU-hours, the world-size factor cancels. Section 3.2 reports the matched validation loss gap across all five sizes but shows throughput only for the 1.7B run. Smaller models have smaller GEMMs and need not expose the throughput behavior relevant to the 2B production configuration, so their throughput ratios are omitted from the main figure rather than treated as part of the production proxy.

At fixed theoretical compute, blockwise FP8 is consistently 0.0031–0.0039 higher in validation loss than BF16 across the ladder. A BF16-versus-FP8 fit in theoretical-compute space estimates 98.0% effective-compute retention; leave-one-size-out fits range from 97.6–98.1%. This corresponds to a 2.0% nominal-compute penalty at matched quality. At 1.7B, median throughput increases by $1.36\times$. Using this largest-scale measured throughput ratio as a proxy for the 2B setting and accounting for the fitted precision penalty, we obtain a ladder-derived, quality-adjusted speedup estimate of $1.34\times$, corresponding to an estimated 25.2% reduction in GPU-hours at matched quality. The FP8 speedup is a ladder-derived estimate. It combines the five-size quality fit with the measured 1.7B throughput ratio, then uses that ratio as a proxy for the 2B production setting.

H Post-Training Details

This appendix provides the data accounting, matched training controls, evaluation details, and per-run results for the post-training experiments in Sections 3.3 and D.5. The GSM8K-based SFT provides a targeted mathematics comparison, the Math&Code SFT with replay tests whether the difference persists after a longer and broader SFT process, and the Tulu-3 mixed-domain SFT examines transfer beyond mathematics.

H.1 Shared Construction and Evaluation Protocol

Dataset construction and accounting. The three SFT settings use the same conversation schema, Qwen3-1.7B tokenizer, Qwen ChatML format, 4,096-token context, assistant-only loss, and document-isolated packing. A packed sequence may contain multiple conversations, but each conversation retains an independent attention context. We split every materialized dataset into 90% training data and 10% validation data, then train for two epochs. The tables below report assistant-supervised token counts before packing, so they exclude runtime-added EOD labels and padding.

We use *UD-based* to denote an SFT model initialized from the *Uniform–Decay (UD)* checkpoint and *CMA-based* to denote one initialized from the *Curriculum Model Averaging (CMA)* checkpoint. Within each setting and repetition, the two models use the same materialized examples, data order, tokenizer, optimizer, learning rate schedule, and training budget. The initialization checkpoint is the only difference within each paired comparison.

Each setting uses three matched SFT data orders, which are labeled as runs 1–3 in the tables. These comparisons concern the complete CMA recipe and do not isolate the contributions of curriculum ordering, late constant-LR training, or checkpoint averaging.

Before materialization, duplicate examples are removed within the relevant source or shard, and empty sequences are discarded. For the GSM8K-oriented mixtures, training examples sharing a 13-gram with a GSM8K test question are removed before SFT. The token counts reported below are computed after these filtering steps and before packing.

Training configuration. We continue to use the MuonH optimizer, as in pretraining (Wen et al., 2026). The base learning rate follows a cosine schedule from 1×10^{-5} to 1×10^{-7}. MuonH-managed matrix parameters use a $10\times$ multiplier for their Hyperball weight learning rate. The UD-based and CMA-based models use identical optimizer settings within every SFT setup.

The GSM8K-based experiment runs on one node with eight RTX 5090 GPUs. It uses tensor parallelism 1, pipeline parallelism 2, micro-batch size 1, global batch size 160, and sequence length 4,096. The Math&Code and Tulu-3 settings use training durations and batch configurations adapted to their data budgets. These configurations remain matched between the UD-based and CMA-based models within each setting.

Evaluation protocol. Evaluation checkpoints are fixed before inspecting the benchmark results. The UD-based and CMA-based models are evaluated at the same training steps within each setting. The reported endpoint is step 172 for GSM8K-based SFT, step 2,431 for Math&Code SFT with replay, and step 449 for Tulu-3 mixed-domain SFT.

For the two mathematics-oriented settings, we evaluate all 1,319 GSM8K test examples using zero-shot ChatML prompting and greedy generation. We use a flexible answer extraction procedure to accommodate variation in final-answer formatting. The answer extractor first truncates any continuation that begins a new user turn. It prioritizes explicitly marked final answers, then checks standard GSM8K and boxed-answer formats, and finally uses the last numeric token as a fallback. Exact numeric equivalence after normalization determines correctness.

This flexible extraction reduces score differences caused only by answer formatting. We also use an explicit-answer-only variant as a sensitivity check and compare the sets of questions solved by the two models. For correct-answer sets C and U from the CMA-based and UD-based models, respectively, we report

$$\mathrm{IoU}(C, U) = \frac{|C \cap U|}{|C \cup U|}.$$

We additionally report C-only and U-only counts, which measure questions solved exclusively by the CMA-based and UD-based models.

For Tulu-3 mixed-domain SFT, we evaluate the final models on the same 15-task benchmark suite used in the base-model scorecard. The suite includes MMLU-Pro and BBH, and its primary summary is an unweighted macro-average over the 15 task scores. IFEval is reported separately as an instruction-following diagnostic. All broad-evaluation numbers are averaged across the same three SFT repetitions.

H.2 GSM8K-Based SFT

Data and training budget. The GSM8K-based setting emphasizes GSM-oriented mathematics data and does not include pretraining replay. The dataset contains 191,767 source conversations and 41.42M assistant-supervised tokens. After packing, it contains 15,292 training sequences and reaches its final evaluation point after 172 optimizer steps.

Table 11: Composition of the GSM8K-based SFT dataset. Token shares are computed from assistant-supervised tokens before packing. Small deviations from the target shares arise from indivisible records.

Component	Included source families	Supervised tokens	Token share
GSM-oriented math	GSM8K train; Tulu grade-filtered math; Tulu OpenMath GSM; OpenMathInstruct augmented GSM8K; MetaMathQA GSM rephrased	37.28M	89.998%
Instruction	PersonaHub instruction following	2.07M	5.000%
General	WildChat; NoRobots	2.07M	5.001%
Total		41.42M	100.000%

Per-run results and answer analysis. The endpoint means and per-run differences are reported in Table 12. The table also reports IoU, C-only, and U-only counts to characterize the overlap between the solved-question sets. The explicit-answer-only evaluation is included as a sensitivity check for the extraction procedure.

Table 12: Paired GSM8K results for the GSM8K-based SFT setting. UD and CMA are accuracies in percent, and Δ is CMA minus UD in percentage points. C-only and U-only count questions solved exclusively by the CMA-based and UD-based models.

Run	Step	UD	CMA	Δ	IoU	C-only	U-only
1	172	67.32	69.14	+1.82	.754	138	114
2	172	67.10	69.22	+2.12	.721	160	132
3	172	66.26	67.63	+1.36	.730	147	129
Mean	172	66.89	68.66	+1.77	–	–	–

H.3 Math&Code SFT with Replay

Data and training budget. The Math&Code mixture contains mathematics data, code data, and a small replay component from the final part of the Phase 2 curriculum. It contains 2,014,933 source conversations and 530.86M assistant-supervised tokens. After packing, it contains 194,530 sequences and reaches its final evaluation point after 2,431 optimizer steps.

The replay examples occupy 4.96% of the source conversations but only 2.46% of the assistant-supervised tokens. The dataset is a legacy materialized mixture whose replay slots were replaced in place. Since its original manifest did not preserve component-level token totals, we recomputed the values below with the same tokenizer and target mask used by the SFT loader. Its packing manifest also records 697 retained overlength rows.

Table 13: Composition of the Math&Code SFT with replay dataset. Assistant-supervised token counts are recomputed from the materialized JSONL using the training tokenizer and target mask.

Component	Included source families	Supervised tokens	Token share
Mathematics	OpenMathInstruct math/GSM variants; MetaMathQA GSM/MATH variants; GSM8K train	430.97M	81.183%
Code	OpenCodeInstruct generic/algorithmic; Magicoder Evol-Instruct	86.85M	16.360%
Curriculum replay	Samples from the final Phase 2 curriculum parquet, formatted as SFT continuation data	13.04M	2.456%
Total		530.86M	100.000%

Per-run results. The paired endpoint scores and set-overlap diagnostics are reported in Table 14. Absolute GSM8K scores should not be compared across the two SFT mixtures. The relevant comparison is the within-setting CMA-minus-UD difference.

Table 14: Paired endpoint GSM8K results for Math&Code SFT with replay. UD and CMA are accuracies in percent, and Δ is CMA minus UD in percentage points.

Run	UD	CMA	Δ	IoU	C-only	U-only
1	73.24	76.50	+3.26	.792	136	93
2	75.06	76.27	+1.21	.798	120	104
3	74.00	75.59	+1.59	.786	129	108
Mean	74.10	76.12	+2.02	–	–	–

H.4 Tulu-3 Mixed-Domain SFT

Data and training budget. The Tulu-3 mixed-domain mixture covers general data, mathematics, code, instruction following, and safety. It contains 264,612 source conversations and 82.25M assistant-supervised tokens.

After packing, the dataset contains 39,914 sequences. The UD-based and CMA-based models are each trained for two epochs and evaluated at the predefined final step 449. This setup uses the same three matched SFT repetitions as the mathematics experiments.

Table 15: Composition of the Tulu-3 mixed-domain SFT dataset. Category shares are defined by assistant-supervised tokens before packing.

Component	Included source families	Supervised tokens	Token share
General	FLAN; WildChat; NoRobots; OASST; Aya; TableGPT; SciRIFF	37.01M	45.001%
Mathematics	Persona Math; grade-filtered math; OpenMath GSM8K; NuminaMath; intermediate algebra	20.56M	24.999%
Code	Evol CodeAlpaca HumanEval; PersonaHub code	8.22M	10.000%
Instruction	PersonaHub instruction following	8.22M	10.000%
Safety	WildGuardMix; WildJailbreak; CoCoNot	8.22M	10.000%
Total		82.25M	100.000%

Broad capability and instruction-following results. Table 16 reports the scores for all 15 tasks and the corresponding CMA–UD differences. The 15-task macro-average is unweighted and excludes IFEval. IFEval is reported separately because it uses a dedicated instruction-following criterion. The

task-level differences are heterogeneous, so the aggregate values should be read together with the full table.

Table 16: Tulu-3 mixed-domain SFT results averaged across three repetitions. UD and CMA are percentages, and Δ is CMA minus UD in percentage points. The 15-task macro-average excludes IFEval, which is reported separately as an instruction-following diagnostic.

Benchmark	UD	CMA	Δ
GSM8K	59.92	59.23	-0.68
MATH	22.39	26.18	$+3.79$
HumanEval	42.27	40.65	-1.63
MBPP	50.97	50.84	-0.13
MMLU	53.97	55.80	$+1.83$
MMLU-Pro	25.73	26.11	$+0.38$
ARC-C	63.28	67.23	$+3.95$
ARC-E	80.90	83.60	$+2.70$
BoolQ	74.18	69.57	-4.61
CommonsenseQA	66.09	68.69	$+2.59$
HellaSwag	43.69	50.94	$+7.25$
PIQA	68.79	69.01	$+0.22$
SIQA	61.05	63.19	$+2.13$
WinoGrande	51.62	52.46	$+0.84$
BBH	44.58	43.52	-1.06
15-task macro-average	53.96	55.13	$+1.17$
IFEval	41.71	43.07	$+1.36$

I MuonH and Learning Rate Schedule Diagnostics

We provide more detailed diagnostics and fitting results for learning rate schedule diagnostics in this section.

I.1 Details about Muon-ELR

MuonH diagnostic traces. To make the optimizer comparison more concrete, this appendix plots Figure 15, the learning rate and update diagnostics for the FC2[7] matrix in the 170M-parameter BF16 experiment used for Figure 9. The aligned and base Muon runs record per-group effective learning rates, weight norms, and raw update norms. The MuonH run records the shared base schedule rather than these per-group fields, so its effective LR target is reconstructed as $10\eta_t^{\text{base}}$ from the configured Hyperball multiplier. Its weight norm is therefore not drawn as an inferred measurement; projection keeps it at the prescribed initial radius.

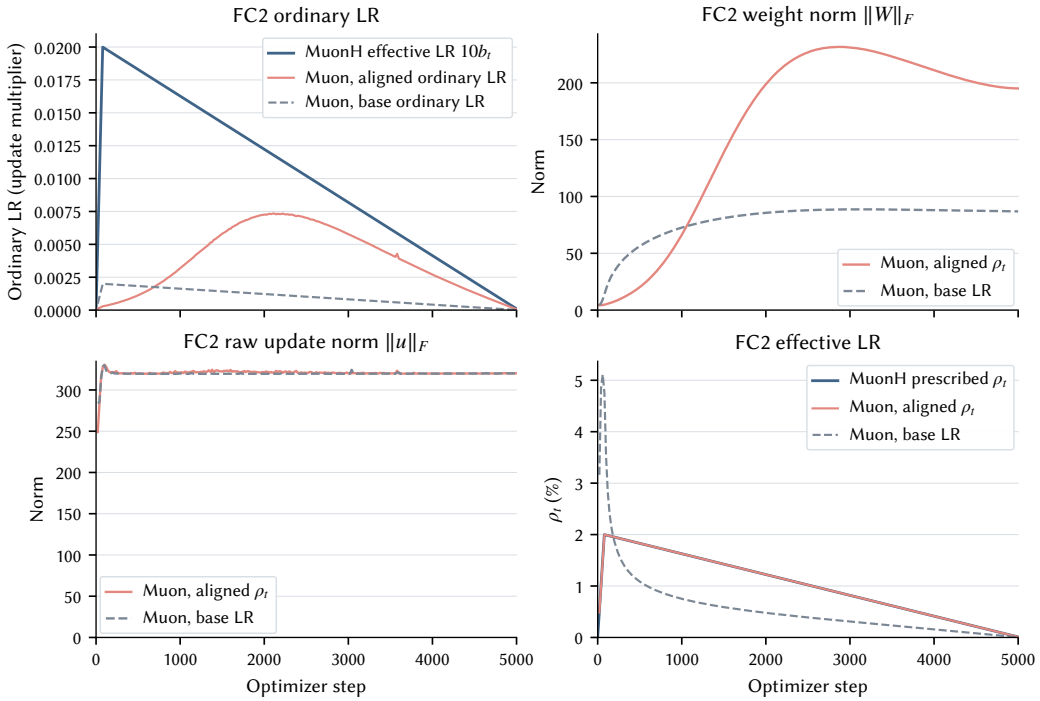


Figure 15: FC2 represents the second layer of the FFN. Here we present the FC2 diagnostics for the BF16 MuonH comparison in Figure 9. The upper-left panel shows the ordinary LR, namely the scalar coefficient that multiplies the ordinary-Muon optimizer update; the lower-right panel shows the induced effective LR. The MuonH curves in both panels are reconstructed from the shared schedule, while the norm panels show the aligned/base Muon traces.

Matched effective LR across schedules and weight decay. To extend the single linear-schedule diagnostic in Figure 9 and its FC2 view in Figure 16, we compare MuonH with effective-LR-aligned Muon (Muon-ELR) across three LR schedules and six constant weight-decay values. All comparisons use the same 170M BF16 model training setup by default. At weight decay 0.1, Muon-ELR closely matches the MuonH endpoint loss for each schedule (Table 17). Under the linear schedule, the same pattern persists across the weight-decay sweep (Table 18). The remaining endpoint gaps are at most 0.006. These results support the view that matching the effective-LR schedule explains most of the observed MuonH convergence advantage.

WSD with a short decay range underperforms cosine and linear schedules. For MuonH at fixed weight decay 0.1, we compare linear, cosine, and WSD schedules in Table 17. Linear and cosine are

[7] FC2 represents the second layer of FFN.

Table 17: Endpoint validation loss for MuonH and Muon-ELR across LR schedules at fixed weight decay 0.1. Losses are rounded to three decimals.

LR schedule	MuonH	Muon-ELR
Linear	3.029	3.030
Cosine	3.040	3.042
WSD	3.055	3.057

relatively close, with endpoint losses of 3.029 and 3.040, whereas WSD reaches 3.055. Thus WSD is 0.026 worse than linear and 0.015 worse than cosine in this experiment. The WSD run uses a terminal linear-decay region covering 20% of the training horizon, so this result motivates treating the decay ratio as an important variable.

Table 18: Endpoint validation loss for MuonH and Muon-ELR across constant weight decay under the linear schedule.

Weight decay	MuonH	Muon-ELR
0.1	3.029	3.030
0.2	3.028	3.025
0.4	3.029	3.023
0.8	3.032	3.026
1.6	3.037	3.037

I.2 Multi-Power Law for Effective Learning Rate Schedules

Preliminary. The Multi-Power Law (MPL) was introduced to predict loss curves across learning rate schedules from a small number of training runs (Luo et al., 2025b). We adapt the same construction to an effective learning rate signal for one selected weight group. Let q_t denote either the scalar optimizer learning rate coefficient η_t or the induced effective LR ρ_t. We define its post-warmup trapezoidal exposure as

$$S_q(t) = \sum_{k=t_{\text{warm}}+1}^{t} \frac{q_{k-1} + q_k}{2} \Delta t_k,$$

and let S_w denote the corresponding exposure accumulated during warmup under the same choice of q. The resulting model is

$$\begin{aligned} L(t) &= L_0 + A\left[S_q(t) + S_w\right]^{-\alpha} + B \sum_{t_{\text{warm}} < k \le t} \Delta q_k \, G(x_k(t)), \qquad \Delta q_k = q_k - q_{k-1}, \\ x_k(t) &= C q_k^{-\gamma} \left[S_q(t) - S_q(k-1)\right], \qquad\qquad G(x) = 1 - (1+x)^{-\beta}. \end{aligned} \tag{16}$$

The intuition is to decompose loss evolution into an exposure-controlled baseline and a schedule-shape correction. The first power-law term is Chinchilla-like: it treats the cumulative q-exposure $S_q(t) + S_w$ as an effective optimization-time coordinate and captures the training progress expected from that accumulated exposure. This term alone, however, does not distinguish schedules that accumulate similar exposure through different decay patterns.

The summation term accounts for the additional loss reduction induced by decreasing the learning rate signal. For a decaying schedule, $\Delta q_k < 0$. A decrease at step k therefore contributes a negative correction

$$B \Delta q_k G(x_k(t)) = -B\left(q_{k-1} - q_k\right) G(x_k(t))$$

to the loss at a later step t. This benefit is not assumed to appear instantaneously. Instead, $x_k(t)$ measures the exposure accumulated after the change, rescaled by the post-change signal level q_k, while $G(x_k(t))$ represents the fraction of the eventual loss reduction that has been realized by time t. As subsequent exposure accumulates, $G(x)$ approaches one, with the unrealized fraction $1 - G(x) = (1+x)^{-\beta}$ decaying according to a power law. Finally, we approximate the total schedule-shape

correction by superposing the delayed responses induced by all preceding changes in q. Thus, MPL combines one power law for cumulative training progress with a collection of power-law responses to reductions in the effective learning rate signal.

A useful analytical simplification assumes that every reduction is fully realized as soon as it occurs, corresponding to $G(x) = 1$. The response-weighted sum then telescopes. With $q_{\text{warm}} = q_{t_{\text{warm}}}$, we obtain

$$\begin{aligned} L_{G=1}(t) &= L_0 + A\left[S_q(t) + S_w\right]^{-\alpha} + B \sum_{t_{\text{warm}} < k \le t} \Delta q_k \\ &= L_0 + A\left[S_q(t) + S_w\right]^{-\alpha} + B\left(q_t - q_{\text{warm}}\right). \end{aligned} \tag{17}$$

For a decaying schedule and $B > 0$, the final term is non-positive and lowers the predicted loss relative to the accumulated-exposure baseline. This instantaneous-response approximation discards when the individual reductions occurred: beyond their contribution to $S_q(t)$, the correction retains only the net decrease from q_{warm} to q_t. We therefore use it as a simplified interpretation of the full response model rather than as a claim that learning rate reductions take effect instantaneously in actual training.

We test this transfer on the two BF16 ordinary-Muon runs already used in Figures 9 and 15. For each run, we fit the post-warmup validation curve using either $q_t = \eta_t$ or $q_t = \rho_t$, with the final 20% of validation targets held out. Because the two signals have different units, each is normalized by its own post-warmup peak before fitting; this only rescales the fitted constants. The fit follows the three-stage MPL protocol: power-only, power plus a schedule-drop term, and the full response above. As a simpler comparison, we also fit the $G(x) = 1$ model in Equation (17); unlike the formal response, this simplification retains only the current change from the warmup endpoint.

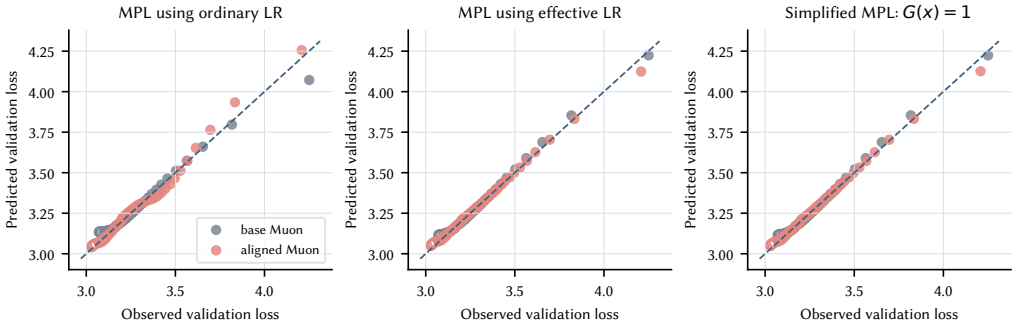


Figure 16: MPL transfer to an effective learning rate signal for the FC2 group (FFN down-projection weight). All panels compare predicted with observed validation loss; filled markers are fitting targets and hollow markers are the final 20% holdout. **Left:** The formal MPL fit using the ordinary LR. **Center:** The formal MPL fit using the induced effective LR. **Right:** The simplified $G(x) = 1$ model in Equation (17), also using the induced effective LR. The two signals are normalized separately, and colors distinguish base and effective-LR-aligned Muon.

The effective LR coordinate gives a lower mean held-out RMSE in this small FC2 diagnostic (0.0210 versus 0.0265 for ordinary LR), with the largest gain for base Muon (0.0270 versus 0.0422). For aligned Muon, the two representations are comparably predictive (0.0149 and 0.0108). The simplified $G(x) = 1$ effective LR fit has a mean held-out RMSE of 0.0207, with 0.0270 for base Muon and 0.0144 for aligned Muon, matching the vanilla MPL of effective LR on these two curves.

Then, from the perspective of Equation (17), the loss decreasing is strongly related to the effective LR decreasing. And as shown in Figure 9, the ordinary Muon run results in an LR schedule that decays aggressively in the early phase, while the MuonH and aligned Muon runs use linear LR decay. The early effective LR decay in the ordinary Muon run results in the loss of potential to decay further near the end of training. Thus, the MuonH and aligned Muon runs finally surpass the ordinary run, which is not by coincidence from this diagnostic.

I.3 Details of the WSD Decay-Range Sweeps

All WSD sweeps use a 0.6B decoder-only model with the Qwen3-0.6B architecture. We train in BF16 with sequence length $4{,}096$, global batch size 512. The decay phase uses a linear LR schedule. For the peak-LR sweep, we evaluate effective peak LRs from 0.004 to 0.028 in increments of 0.004 at TPP 20, 50, and 100. For each peak LR, the decay ratio ranges from 0.2 to 1.0 in increments of 0.2. For the training-horizon sweep, we extend selected peak-LR settings to TPP 200. TPP 20, 50, 100, and 200 correspond to 11.32, 28.31, 56.62, and 113.25B training tokens, respectively. The effective peak LRs 0.008, 0.012, and 0.016 have measurements through TPP 200. For every run, we use the validation loss at the final step to select the best decay ratio among the tested settings.

J Puro-2B Scaling Checkpoint Ledger

The following ledger freezes the identifiers and accounting coordinates used by Figure 2(b). Phase 1 is shared by all rows. Phase 2 tokens are schedule consumption, so the production Phase 2 pool is not added a second time when the early transition range is listed. The endpoint comparison is visualized in Figures 4 and 11b.

2948 2947 2946 2945 2944 2943 2942 2941 2940 2939 2938 2937 2936 2935 2934 2933 2932 2931 2930 2929 2928 2927 2926 2925 2924 2923 2922 2921 2920 2919 2918 2917 2916

Table 19: Checkpoint lineage for the five UD points in the Puro Cost Scaling Law, the CD endpoint, and the canonical CMA endpoint. GPU-hours are total active-training GPU-hours and costs use the normalized RTX 5090 rate and are reported in USD. Here, *SMA6* (simple moving average) denotes the equal-weight average of model parameters from six late checkpoints on the step-218,000 constant-LR continuation; optimizer states are not averaged (details in Section K.3).

Released checkpoint ID	Phase 2 tokens	Total tokens	Phase 2 ordering	Model averaging	Total GPU-h	Cost (USD)
Puro-2B-UD-Phase2-1of16	60.00B	498.83B	UD global reshuffle	No	7,041	$2.16K
Puro-2B-UD-Phase2-1of8	120.00B	558.83B	UD global reshuffle	No	8,073	$2.47K
Puro-2B-UD-Phase2-1of4	240.00B	678.83B	UD global reshuffle	No	10,136	$3.10K
Puro-2B-UD-Phase2-1of2	480.00B	918.83B	UD global reshuffle	No	14,262	$4.37K
Puro-2B-UD	959.99B	1.40T	UD global reshuffle	No	22,514	$6.89K
Puro-2B-CD	959.99B	1.40T	CD component-local curriculum; decay final	No	22,514	$6.89K
Puro-2B-CMA	959.99B	1.40T	CMA curriculum + late constant LR	Yes (SMA6)	22,514	$6.89K

Table 20: 15-benchmark score sheet for the UD, CD, CDC, and CMA branches(detailed definitions in Section D.4). Scores are percentages, and Avg15 is the unweighted arithmetic mean over the 15 displayed benchmarks. The canonical Puro-2B row(CMA 218k avg) is the released six-checkpoint SMA endpoint.

Model	GSM8K	MATH	MBPP	HumanEval	MMLU	MMLU-Pro	ARC-C	ARC-E	BoolQ	CSQA	HSwag	PIQA	SIQA	WinoGrande	BBH	Avg15
UD 1/16	47.23	21.78	45.53	18.90	49.71	20.67	61.02	79.19	75.11	65.68	62.28	75.30	60.18	59.51	37.66	51.98
UD 1/8	48.22	24.64	46.69	25.00	51.37	22.29	63.39	79.54	77.16	64.78	61.25	75.90	60.24	59.98	38.09	53.24
UD 1/4	42.30	26.02	48.64	23.78	53.96	23.83	66.44	81.83	74.83	65.27	62.59	76.17	63.10	60.22	41.20	54.01
UD 1/2 ($4.4K)	56.03	27.86	49.81	20.73	54.82	24.02	66.78	84.66	77.34	67.65	63.62	75.84	62.90	59.91	41.14	55.54
UD decay avg	55.72	26.70	50.58	18.90	56.23	24.87	68.81	84.13	79.27	67.16	63.32	77.69	62.64	60.30	37.29	55.57
UD full	55.72	26.94	50.19	20.73	56.22	25.55	70.17	84.83	79.48	67.49	63.30	77.58	62.64	60.38	38.75	56.00
CD decay final	57.77	29.68	52.14	25.00	57.11	25.56	73.56	85.89	76.24	68.88	63.38	75.46	64.43	60.93	41.45	57.17
CD decay avg	57.32	29.30	51.75	26.22	57.12	26.50	73.56	86.07	76.18	69.29	62.96	75.63	64.33	60.69	40.80	57.18
CDC 215k final	54.74	27.60	46.30	22.56	56.28	25.44	73.90	84.83	73.52	67.49	62.46	74.92	62.85	61.33	40.39	55.64
CMA 215k avg	55.95	28.18	50.19	25.00	57.05	26.09	75.93	86.07	74.80	67.73	62.72	75.41	62.69	61.48	42.78	56.80
CDC 218k final	57.47	29.22	50.19	30.49	56.45	25.12	73.90	84.13	78.10	67.49	63.02	75.08	63.51	61.48	41.19	57.12
CMA 218k avg ($6.9K)	59.67	30.30	52.92	31.10	57.44	25.81	74.24	86.07	76.85	67.81	63.21	75.52	63.61	60.62	42.05	57.81

K Curriculum Construction and Model Averaging Details

This appendix provides the construction and training details for the Phase 2 curriculum. The construction overview and component ablations of CMA are reported in Section D.4.

K.1 Curriculum Bucket Construction

We construct 376 curriculum buckets, each containing approximately 2.5B tokens. The goal is to place higher-quality data later while keeping the source mixture approximately stable. Each bucket should therefore receive about $1/376$ of the tokens from every component, taken from corresponding positions in their quality orderings. We partition by token count rather than document count because documents vary in length.

Grouping documents to reduce preprocessing cost. An exact construction would sort every document within each component and compute cumulative token counts to determine the bucket boundaries. To reduce this overhead, we first group documents into *internal bins* and compute cumulative counts across bins rather than individual documents. These bins serve as intermediate groups for preprocessing, instead of the final curriculum buckets.

For each scored component, approximate quantiles define up to 256 score intervals, ordered from less to more preferred according to the source's quality criterion. We further divide each interval into 64 deterministic hash shards. Each *interval–shard pair* forms an internal bin, ordered first by score interval and then by shard index. The score intervals establish the quality order, while hashing provides a reproducible subdivision for finer token accounting. Repeated scores may produce fewer intervals, and missing scores are placed at the least-preferred end. For components without usable scores, we instead use $4{,}096$ hash shards in shard-index order, without interpreting this order as a quality ranking.

Assigning documents to curriculum buckets. The token count of each bin determines the interval it occupies within its component's normalized token range. We also need a position for each document within that interval, since a bin may overlap multiple curriculum buckets. We obtain this position from the same hash used for shard selection. Let $h(x) \in [0, 1)$ be the normalized deterministic hash of document x. With K shards, $\lfloor Kh(x) \rfloor$ selects the shard, and the remainder $u_{c,j(x)}(x) = Kh(x) - \lfloor Kh(x) \rfloor$ gives the document's fractional position within it. Here, K is 64 for scored components and 4096 for unscored components.

Let $T_{c,j}$ be the token count of bin j in component c, and let $T_c = \sum_j T_{c,j}$. The bin's normalized starting position is

$$p_{c,j} = \frac{\sum_{i<j} T_{c,i}}{T_c}. \tag{18}$$

We place document x within its bin's token interval as

$$p(x) = p_{c,j(x)} + u_{c,j(x)}(x) \frac{T_{c,j(x)}}{T_c}. \tag{19}$$

Thus, actual token counts determine the bin's starting position and width, while the hash remainder determines the document's offset within it. This avoids computing an exact cumulative-token position for every document.

Finally, we divide the normalized range into 376 equal intervals and assign each document to a bucket:

$$b(x) = \min \{375,\ \lfloor 376\, p(x) \rfloor\}. \tag{20}$$

Documents assigned to the same bucket are mixed across components. Each bucket receives approximately the same token fraction from every component, while later buckets contain more-preferred data from the scored components. The proportions are approximate because documents remain intact and their within-bin positions are hash-based. Scores are interpreted only within each component and are not compared across sources.

K.2 Phase Transition and UD Control

Phase 2 begins with a transition from the Phase 1 mixture to the ordered Phase 2 data. The transition combines a deterministic stable-hash sample of Phase 1 replay with the earliest approximately 2.34% of the Phase 2 curriculum. The Phase 2 portion is already included in the component pool, so only the Phase 1 replay is added when computing the total token count. Table 21 gives the token counts.

Table 21: Approximate token counts for the Phase 2 transition and training schedule. The Phase 2 transition portion is included in the component pool; the complete schedule adds only the Phase 1 replay.

Data portion	Tokens
Phase 1 replay	21.93B
Phase 2 transition portion	21.93B
Total transition	43.87B
Remaining Phase 2 data	916.13B
Phase 2 component pool	938.06B
Complete Phase 2 schedule	959.99B

Table 22 reports the token-share ranges across curriculum buckets after the transition. The group names follow the preprocessing manifest. In this manifest, the code-only MegaMath-Code partition is grouped under Mathematics, whereas Tables 4 and 23 report it under Code.

Table 22: Token-share ranges across Phase 2 curriculum buckets after the transition. Groups follow the preprocessing manifest.

Group	Token-share range
Chinese	9.25%–9.54%
Code	6.37%–8.73%
English	58.32%–59.79%
Mathematics	22.38%–22.98%
Instruction-formatted	1.30%–1.40%

We shuffle the transition data with seed `20260601` and the subsequent curriculum buckets with seed `20260602`. The UD stream is constructed by globally reshuffling the same Phase 2 component pool with seed `20260609`. Thus, CD and UD contain the same token multiset at the component-pool level but differ in presentation order.

K.3 Constant-LR Continuation and Checkpoint Averaging

We resume constant-LR training from steps $215{,}000$ and $218{,}000$ of the curriculum run. The production model uses the continuation from step $218{,}000$; the comparison is reported in Section D.4.

For this continuation, we fix the base LR at 4.08×10^{-5} rather than following the remaining LR decay. With the production MuonH multiplier $m = 10$, the weight LR for MuonH matrices is 4.08×10^{-4}. This is also their effective LR, as defined in Section D.3.

We form the production model by equally averaging six late checkpoints. The first five are taken every 100 optimizer steps, from step $222{,}100$ through $222{,}500$, and the sixth is taken at step $222{,}569$. The averaging window spans 469 optimizer steps, corresponding to approximately 2.95B tokens. A 100-step interval corresponds to approximately 0.63B tokens under the production batch configuration.

Only model parameters are averaged, not optimizer states. The released model is therefore the equal-weight average of these checkpoints rather than the unaveraged final checkpoint. The reference CMA method evaluates several averaging rules and uses an exponential moving average over the final six checkpoints by default (Luo et al., 2026). Our implementation instead uses equal weights.

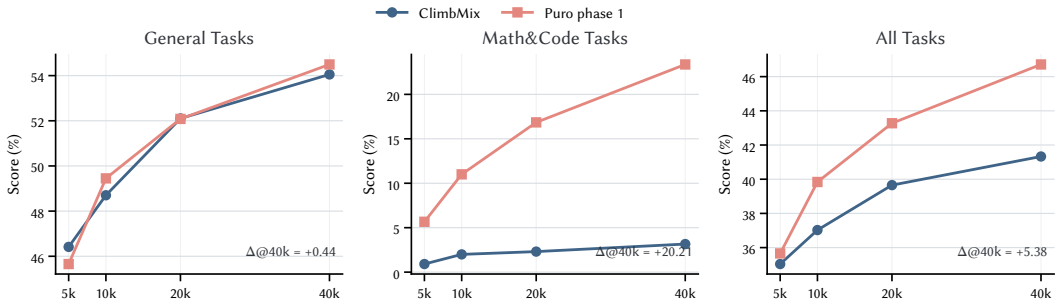


Figure 17: Matched data-efficiency ladder: Puro Phase 1 and ClimbMix achieve similar scores on General-6, while Puro's advantage on Math&Code grows with the training budget and widens its lead on Combined-8.

L Data Recipe Details

A supplementary data-efficiency comparison. We compare the Puro Phase 1 dataset with the ClimbMix dataset (Diao et al., 2025) using Qwen3-1.7B proxy runs. Both mixtures include Nemotron-CC datasets, so this is a comparison of complete mixtures with partially overlapping source families. Our proxy experiments use BF16 training with a context length of 4,096 tokens and a global batch size of 512. We use budgets of 10.49B, 20.97B, 41.94B, and 83.89B tokens, corresponding to training runs ranging from 5k to 40k steps. As shown in Figure 17, the three panels report average scores on general tasks (MMLU, ARC-C, ARC-E, SIQA, LAMBADA, and WinoGrande), code and math tasks (GSM8K and MBPP), and all eight tasks combined. General performance is nearly matched between the two mixtures, with a maximum absolute gap of 0.76 percentage points. The difference is larger in math and code, where Puro exceeds ClimbMix by 4.76 points at 5k steps and 20.21 points at 40k steps. We attribute this difference to the higher proportion of code and math data in the Puro dataset. Puro's overall average is higher by 0.62 points at 5k steps, and this gap grows to 5.38 points at 40k steps. We therefore interpret this ladder as supplementary evidence of differences in capability profiles, with stronger math and code coverage in Puro. Since this comparison does not cover the full training pipeline and the two datasets have different characteristics, we do not report an improvement in compute efficiency in this case.

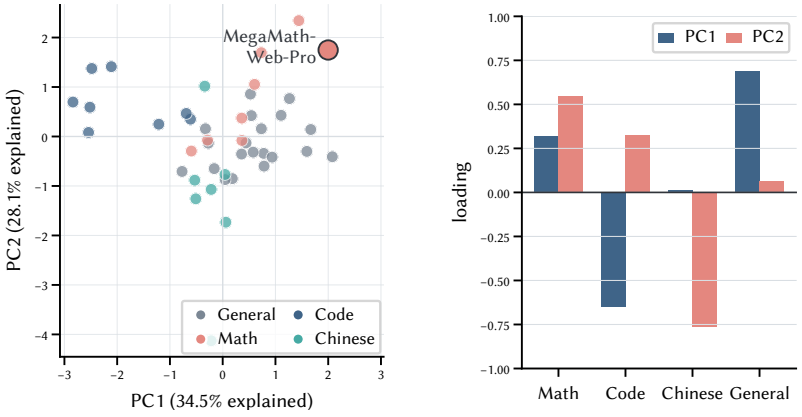
Component accounting and license scope. Table 23 reports the materialized data components used in the two pretraining phases. Its token counts come from the component-level materialization records and therefore reflect the sampled, tokenized components actually used by the training recipe. The table identifies the upstream dataset or source family and states the current public license or terms that we could verify. A missing license is reported as "to verify" or "not specified". Nemotron-CC-v2 and the Synthetic-Code partition of Nemotron-Pretraining-Code-v1 use NVIDIA's data agreement, which permits model training but prohibits redistribution of the raw dataset (NVIDIA, 2025). The merged Chinese web component uses the same CCI, SkyPile-150B, WanJuan1.0, IndustryCorpus, and WuDaoCorpus2.0 source families documented by the Kaiyuan recipe and inherits their respective terms (Wang et al., 2024a; Wei et al., 2023; He et al., 2023; 2024; Shi et al., 2024; Zhang et al., 2021a;b; of Artificial Intelligence, 2023; Skywork-AI, 2023). Due to these terms, we also release our data preprocessing framework for data construction.

Proxy measurement protocol. We use a Qwen3-0.6B model for proxy experiments. The proxy experiment starts from a checkpoint trained on 86B tokens on the same broad base mixture. Then it receives approximately 8.4B continuation tokens. It corresponds to 2,000 steps with a sequence length of 4,096 and a global batch size of 1,024. The continuation differs only in its candidate source or within-source slice. We evaluate the final checkpoint on a suite of benchmarks and aggregate evaluation scores into Math, Code, Chinese, and General axes. Math combines GSM8K and MATH, Code combines MBPP and HumanEval, Chinese combines C-Eval and CMMLU, and General summarizes the remaining knowledge and commonsense tasks.

We adopt a continuous data schedule in the proxy experiments. Let $r(t)$ denote the candidate fraction at continuation step t. The candidate share follows

$$r(t) = \min\left(0.8,\ 0.8 \times \frac{t}{1600}\right), \qquad 0 \le t \le 2000,$$

so the candidate ramps from 0% to 80% during the first 1,600 steps and remains at 80% for the final 400 steps. This schedule controls the distribution shift to avoid abrupt changes in distribution for measurement. The resulting feature vector depends on the experiment setup, including proxy scale, continuation schedule, and evaluation suite.



(a) Candidate profiles (b) PCA loadings

Figure 18: Proxy evidence used in recipe design: **(a)** Candidate profiles on the first two principal components and **(b)** the capability contrasts exposed by the PCA loadings. PCA is descriptive; the comparisons do not establish causal gains.

Proxy Capability Dimensions The raw proxy export contains 39 source or score-quantile slices evaluated on 15 benchmarks. We use four capability dimensions to make the benchmark panel interpretable: Math averages GSM8K and MATH; Code averages MBPP and HumanEval; Chinese averages C-Eval and CMMLU; and General averages the remaining nine English knowledge and commonsense tasks. The raw percentage averages and their within-panel ranks over the 39 data slices are reported in Table 25. For the PCA visualization, each benchmark b is first converted to a population z-score across the 39 candidates,

$$z_{i,b} = \frac{x_{i,b} - \mu_b}{\sigma_b}.$$

The PCA visualization averages the corresponding z-score columns within each dimension and then standardizes the four aggregate axes once more before fitting PCA. The first two components explain 34.5% and 28.1% of the standardized four-axis variance. Panel (b) shows the resulting loading contrasts: PC1 primarily contrasts General with Code, while PC2 primarily contrasts Math with Chinese. These directions summarize co-variation in this panel.

Table 23: Dataset-family accounting for the stationary Phase 1 and Phase 2 component pools.

Dataset / family	Domain	P1 tokens	P2 tokens	Materialization	Upstream source	Declared upstream terms[a]
FineWeb-Edu-CN	Chinese	19.73B	34.79B	Both: Score-filtered	opencsg/Fineweb-Edu-Chinese-V2.1 (Yu et al., 2025)	OpenCSG Community License and Apache 2.0
FineWeb-Edu-Chinese V2.2	Chinese	16.50B	28.11B	P1: Score-filtered; P2: No additional sampling	opencsg/Fineweb-Edu-Chinese-V2.2 (Yu et al., 2025)	Apache 2.0 metadata; web-source terms also apply
ChineseWebText2.0 (quality-filtered subset)	Chinese	11.05B	17.50B	Both: Score-filtered	CASIA-LM/ChineseWebText2.0 (Zhang et al., 2024b)	Apache 2.0 metadata
Deduplicated merged Chinese web	Chinese	–	5.77B	P2: Score-filtered	CCI{,2,3}, SkyPile-150B, WanJuan1.0, IndustryCorpus{,2}, and WuDaoCorpus2.0 (Wang et al., 2024a; Wei et al., 2023; He et al., 2023; 2024; Shi et al., 2024; Zhang et al., 2021a;b)	Mixed upstream terms; see appendix license note
Baidu-Baike	Chinese	1.19B	1.19B	Both: No additional sampling	mohamedah/baidu_baike	MIT metadata; upstream content rights are not clarified
FineWiki-CN	Chinese	1.10B	1.10B	Both: No additional sampling	HuggingFaceFW/finewiki (Penedo, 2025)	CC BY-SA 4.0; older Wikipedia content may also be GFDL
UNDL ZH-EN Aligned	Chinese	1.75B	–	P1: No additional sampling	bot-yaya/undl_zh2en_aligned	MIT metadata; underlying UN document terms also apply
Alpaca-Zh	Chinese	0.01B	0.01B	Both: No additional sampling	hfl/alpaca_zh_51k (Cui et al., 2023)	Apache 2.0
MegaMath-Code	Code	–	42.77B	P2: No additional sampling	LLM360/MegaMath (Zhou et al., 2025)	ODC-By 1.0
Nemotron Synthetic Code	Code	17.09B	25.64B	Both: Random subsample	nvidia/Nemotron-Pretraining-Code-v1 (Synthetic-Code) (Basant et al., 2025)	NVIDIA Data Agreement for Model Training; gated, no raw-data redistribution
Swallow-Code-v2	Code	14.12B	9.30B	Both: Score-filtered	tokyotech-llm/swallow-code-v2 (Fujii et al., 2025)	Apache 2.0 and original code licenses
CoderForge Trajectories	Code	–	15.78B	P2: No additional sampling	togethercomputer/CoderForge-Preview (Ariyak et al., 2026)	No dataset-level license declared; per-row licenses are provided
StackExchange	Code	–	5.54B	P2: Random subsample	togethercomputer/RedPajama-Data-1T (Computer, 2023)	CC BY-SA 2.5/3.0/4.0

Continued on next page

Table 23: Dataset-family accounting for the stationary Phase 1 and Phase 2 component pools. (Continued)

Dataset / family	Domain	P1 tokens	P2 tokens	Materialization	Upstream source	Declared upstream terms[a]
Python Code Large	Code	–	4.37B	P2: No additional sampling	ajibawa-2023/Python-Code-Large	MIT metadata; upstream code provenance undocumented
Python-Edu	Code	3.41B	–	P1: No additional sampling	HuggingFaceTB/smollm-corpus (Lozhkov et al., 2024; Ben Allal et al., 2024)	ODC-By 1.0 and original code licenses
Codeforces-CoTs	Code	–	2.93B	P2: No additional sampling	open-r1/codeforces-cots (Penedo et al., 2025)	CC BY 4.0 metadata; upstream Codeforces terms also apply
GitHub Top Code	Code	–	1.55B	P2: No additional sampling	ronantakizawa/github-top-code	MIT metadata; original repository licenses apply
Jupyter Agent	Code	–	0.23B	P2: No additional sampling	jupyter-agent/jupyter-agent-dataset (Colle et al., 2025)	Apache 2.0; referenced source terms also apply
LongCodeU CU-DFA	Code	–	0.02B	P2: No additional sampling	longcodeu/longcodeu-dataset (Ye et al., 2025)	Apache 2.0
Nemotron HQ	English	107.51B	447.97B	Both: Score-filtered	nvidia/Nemotron-CC-v2 (Su et al., 2025; Basant et al., 2025)	NVIDIA Data Agreement for Model Training; gated, no raw-data redistribution
Nemotron HQ Synthetic	English	97.67B	39.64B	Both: Score-filtered	nvidia/Nemotron-CC-v2 (Su et al., 2025; Basant et al., 2025)	NVIDIA Data Agreement for Model Training; gated, no raw-data redistribution
FineWeb-Edu-EN	English	59.79B	–	P1: Score-filtered	HuggingFaceTB/smollm-corpus (Ben Allal et al., 2024)	ODC-By 1.0; Common Crawl terms also apply
Cosmopedia-v2	English	27.41B	27.41B	Both: No additional sampling	HuggingFaceTB/smollm-corpus (Ben Allal et al., 2024)	ODC-By 1.0
DCLM-Dedup	English	–	33.77B	P2: Score-filtered	mlfoundations/dclm-baseline-1.0 (Li et al., 2024)	CC BY 4.0; Common Crawl terms also apply
ArXiv	English	28.93B	–	P1: No additional sampling	togethercomputer/RedPajama-Data-1T (Computer, 2023)	Metadata CC0 1.0; article content has author-selected licenses
FineWiki-EN	English	–	8.74B	P2: No additional sampling	HuggingFaceFW/finewiki (Penedo, 2025)	CC BY-SA 4.0; older Wikipedia content may also be GFDL
UltraData-Math	Math	–	118.33B	P2: No additional sampling	openbmb/UltraData-Math (Zhou et al., 2026a)	Apache 2.0 on the current dataset card
SwallowMath-v2	Math	9.99B	13.32B	Both: Random subsample	tokyotech-llm/swallow-math-v2 (Fujii et al., 2025)	Apache 2.0
Nemotron-CC-Math	Math	11.48B	2.47B	Both: Score-filtered	nvidia/Nemotron-CC-Math-v1 (Basant et al., 2025)	NVIDIA Data Agreement/Open Data terms; production revision to pin

Continued on next page

Table 23: Dataset-family accounting for the stationary Phase 1 and Phase 2 component pools. (Continued)

Dataset / family	Domain	P1 tokens	P2 tokens	Materialization	Upstream source	Declared upstream terms[a]
MegaMath-Web-Pro	Math	–	13.45B	P2: No additional sampling	LLM360/MegaMath (Zhou et al., 2025)	ODC-By 1.0
OpenWebMath	Math	–	13.23B	P2: No additional sampling	open-web-math/open-web-math (Paster et al., 2023)	ODC-By 1.0; Common Crawl terms also apply
FineMath	Math	10.10B	–	P1: No additional sampling	HuggingFaceTB/finemath (Allal et al., 2025)	ODC-By 1.0
AutoMathText	Math	–	8.71B	P2: No additional sampling	math-ai/AutoMathText (Zhang et al., 2025b)	CC BY-SA 4.0
STEM Reasoning Complex	Math	–	1.21B	P2: No additional sampling	galaxyMindAiLabs/stem-reasoning-complex	Apache 2.0
NuminaMath-CoT	Math	–	0.44B	P2: No additional sampling	AI-MO/NuminaMath-CoT (LI et al., 2024)	Apache 2.0
FineProofs SFT	Math	–	0.13B	P2: No additional sampling	lm-provers/FineProofs-SFT	Apache 2.0
Nemotron Terminal Corpus	SFT / instruction	–	6.27B	P2: No additional sampling	nvidia/Nemotron-Terminal-Corpus (Pi et al., 2026)	CC BY 4.0 on the current dataset card; production revision to pin
JiuZhang3.0 PT-CoT	SFT / instruction	–	3.58B	P2: No additional sampling	ToheartZhang/JiuZhang3.0-Corpus-PT-CoT	Not specified on the upstream dataset card
Tulu-3 SFT	SFT / instruction	–	0.64B	P2: No additional sampling	allenai/tulu-3-sft-mixture (Lambert et al., 2024)	ODC-By 1.0 mixture; stricter subset terms may apply
ToolMind	SFT / instruction	–	0.53B	P2: No additional sampling	Nanbeige/ToolMind (Yang et al., 2025b)	Apache 2.0
ToolBench	SFT / instruction	–	0.47B	P2: No additional sampling	tuandunghcmut/toolbench-v1 (Qin et al., 2023)	Apache 2.0 metadata
ToolMind-Web-QA	SFT / instruction	–	0.43B	P2: No additional sampling	Nanbeige/ToolMind-Web-QA (Yang et al., 2026)	Apache 2.0
Multi-Turn Long Context	SFT / instruction	–	0.22B	P2: No additional sampling	TreeAILab/Multi-turn_Long-context_Benchmark_for_LLMs (Li et al., 2025)	CC BY 4.0
SlimOrca	SFT / instruction	–	0.20B	P2: No additional sampling	Open-Orca/SlimOrca (Mukherjee et al., 2023; Longpre et al., 2023)	MIT
LongAlpaca	SFT / instruction	–	0.12B	P2: No additional sampling	Yukang/LongAlpaca-12k (Chen et al., 2023)	No license declared on the current dataset card

Continued on next page

Table 23: Dataset-family accounting for the stationary Phase 1 and Phase 2 component pools. (Continued)

Dataset / family	Domain	P1 tokens	P2 tokens	Materialization	Upstream source	Declared upstream terms[a]
Organic CoT	SFT / instruction	–	0.11B	P2: No additional sampling	CodonProject/Organic-Reasoning-195k (ingested as A03HCY/Organic-CoT-Reasoning-SFT)	Apache 2.0
OpenThoughts Agent	SFT / instruction	–	0.11B	P2: No additional sampling	open-thoughts/OpenThoughts-Agent-v1-SFT (Team, 2025)	Apache 2.0

[a] Token counts are tokenizer-dependent materialized exposure, not necessarily unique upstream content; a family can aggregate overlapping configurations. They exclude the Phase 2 transition and replay. The code-only MegaMath-Code partition is reported under Code although its raw scheduling manifest used the broader math label. Terms describe current public metadata unless otherwise noted.

Table 24: Raw proxy benchmark scores (percentage points) before per-task standardization. Dataset labels follow the canonical family names in Table 23. Source qualifiers remain in the Slice column. The redundant DCLM-Dedup-50B probes are omitted because they duplicate DCLM-Dedup in the production recipe.

Source	Slice	GSM8K	MATH	MBPP	HE	CEval	CMMLU	MMLU	ARC-C	ARC-E	BoolQ	CSQA	HSwag	PIQA	SIQA	WinoG
Algebraic Stack Train	random	0.38	0.96	3.11	5.49	29.78	30.37	30.82	28.14	61.20	62.32	47.91	38.71	65.61	43.14	52.41
ArXiv	random	0.68	1.04	1.95	5.49	29.06	30.53	30.85	26.44	58.91	62.26	47.50	38.39	66.92	43.86	52.49
AutoMathText	random	0.61	1.44	6.61	4.88	29.58	30.47	31.03	29.15	65.08	57.40	49.63	38.70	67.30	44.98	52.57
Cosmopedia-v2	random	0.38	1.54	5.06	5.49	30.97	30.35	31.24	27.12	63.14	53.09	47.83	40.09	67.68	44.11	52.80
DCLM-Dedup	q00	0.91	0.96	1.95	4.88	29.39	31.04	32.03	29.83	63.67	62.32	51.02	40.69	68.72	44.42	53.20
DCLM-Dedup	q25	0.91	1.34	3.50	7.32	28.73	30.71	30.95	26.78	61.02	60.49	49.30	40.63	68.66	44.68	53.59
DCLM-Dedup	q50	0.38	0.88	1.56	4.88	29.75	30.42	30.59	28.47	61.20	60.34	49.47	40.70	67.85	44.32	54.30
DCLM-Dedup	q75	1.14	0.92	2.72	5.49	30.79	30.68	30.11	28.14	58.02	61.77	48.65	40.10	68.55	43.45	52.88
FLAN	random	0.61	0.84	1.17	3.05	29.28	30.49	30.37	28.14	60.14	60.15	47.91	39.46	67.19	43.76	52.17
PES2O	random	0.45	0.66	2.72	5.49	29.37	30.84	31.48	26.78	60.32	56.67	47.26	38.12	66.27	43.04	53.20
FineMath	random	1.59	1.74	3.89	3.05	28.70	30.67	30.84	26.78	61.55	62.72	47.83	38.64	67.36	44.42	50.43
FineWeb-Edu-CN	q00	0.83	0.56	1.17	6.10	31.83	32.44	30.76	27.46	60.49	55.69	48.24	38.36	67.79	43.65	52.72
FineWeb-Edu-CN	q25	0.76	0.60	4.28	4.88	29.61	31.24	30.59	29.49	60.49	56.12	48.48	38.53	67.25	43.96	53.28
FineWeb-Edu-CN	q50	0.38	0.76	3.11	5.49	29.05	31.16	30.92	28.47	61.38	52.51	49.30	39.27	67.41	44.32	54.14
FineWeb-Edu-CN	q75	0.99	0.90	2.33	6.71	28.51	30.39	30.44	29.15	61.02	55.38	47.91	38.48	67.85	43.76	52.64
FineWeb-Edu-EN	q00	0.61	1.06	2.72	4.88	29.85	30.58	32.48	30.85	66.84	59.88	49.63	39.90	68.99	43.04	52.33
FineWeb-Edu-EN	q25	0.38	0.68	2.72	5.49	30.00	30.24	31.51	25.76	61.55	58.69	49.47	40.29	68.12	45.39	52.88
FineWeb-Edu-EN	q50	0.68	1.56	2.72	4.88	29.34	30.16	30.77	29.83	60.32	61.25	50.12	41.32	68.06	45.14	52.49
FineWeb-Edu-EN	q75	0.45	1.14	2.33	5.49	30.16	30.56	31.54	26.78	61.20	55.11	49.63	40.92	69.15	44.83	52.49
FineWiki-EN	random	0.99	0.70	2.72	6.71	30.17	30.37	29.98	28.14	63.49	61.68	48.32	38.77	66.59	44.63	51.62
MegaMath-Code	random	1.14	1.30	13.23	10.37	29.64	30.60	31.14	28.81	61.90	62.05	46.76	38.53	66.49	43.19	51.70
MegaMath-Web	random	0.99	0.62	3.11	2.44	29.68	30.18	30.73	28.47	62.96	50.61	47.67	38.30	66.49	43.30	51.30
MegaMath-Web-Pro	random	3.03	1.90	3.50	5.49	29.93	30.53	31.86	30.51	63.14	62.78	49.80	39.64	67.03	44.47	51.54
Nemotron-CC-Math	random	1.82	3.74	2.72	6.71	29.13	30.48	31.36	29.83	62.08	61.99	48.98	38.96	66.70	43.91	53.28
Nemotron HQ	random (medium-high-quality)	0.61	1.32	1.56	6.10	29.33	30.55	30.82	27.46	59.79	58.35	50.04	40.72	68.34	44.83	53.04
Nemotron HQ	random	0.53	1.24	2.33	6.71	29.17	30.45	30.80	31.53	62.26	60.21	48.08	41.31	68.17	43.30	52.80
Nemotron HQ Synthetic	random	0.38	1.12	5.06	6.71	30.60	30.74	31.22	27.80	63.67	60.40	47.50	40.94	68.93	43.81	52.25
Nemotron Synthetic Code	random	0.53	0.70	7.39	10.37	29.22	30.45	30.32	25.42	60.85	57.25	46.36	38.18	66.54	44.22	51.07
OpenWebMath	random	0.53	1.18	3.11	4.88	29.27	30.65	31.09	29.83	61.90	53.55	47.99	38.48	67.95	44.98	52.96
StackExchange	random	0.45	0.54	5.84	4.88	29.24	30.17	30.64	29.83	60.85	55.41	47.75	39.14	67.30	43.91	52.96
Stack-v2-Smol	random	0.91	1.10	5.84	4.88	30.23	30.07	30.13	29.49	59.08	62.42	45.78	37.87	67.14	43.24	50.51
StarCoder Tokens	random	0.68	0.50	6.61	4.27	29.68	29.92	30.89	27.80	61.55	62.23	45.86	38.11	67.30	42.94	53.75
Swallow-Code-v2	q00	0.61	0.90	8.95	10.37	28.20	30.71	30.55	28.14	60.49	60.40	47.99	38.83	67.30	43.40	52.64
Swallow-Code-v2	q25	0.30	0.82	9.34	9.15	28.95	30.65	30.50	28.14	58.91	59.66	48.48	38.57	66.59	42.84	52.17
Swallow-Code-v2	q75	0.68	0.42	8.95	9.76	29.71	30.74	31.13	26.10	61.55	60.89	46.11	38.63	66.70	42.99	51.70
SwallowMath-v2	random	1.21	4.60	0.39	7.32	30.26	30.18	30.80	27.12	61.38	62.11	47.34	38.60	67.57	43.76	51.07
Deduplicated merged Chinese web	q00	0.53	0.76	2.33	4.27	30.65	30.96	30.77	27.46	59.61	59.05	47.91	38.33	67.79	44.37	52.72
Deduplicated merged Chinese web	q50	0.83	0.68	3.89	4.88	30.40	30.65	30.74	26.10	59.79	57.43	47.91	38.73	67.19	44.47	51.85

Table 24 – continued

Source	Slice	GSM8K	MATH	MBPP	HE	CEval	CMMLU	MMLU	ARC-C	ARC-E	BoolQ	CSQA	HSwag	PIQA	SIQA	WinoG
Deduplicated merged Chinese web	q75	0.45	0.60	2.72	4.88	29.74	31.03	30.82	27.12	60.14	57.74	47.26	38.97	67.14	44.06	51.93

Table 25: Raw proxy benchmark aggregates and descending ranks by data slice. Dataset labels follow the canonical family names in Table 23; source qualifiers remain in the Slice column. Math, Code, and CN average 2 benchmarks each; General (the English knowledge and commonsense axis) averages 9 benchmarks; Overall averages all 15 raw benchmark scores. The ranks are computed over the 39 displayed non-duplicate slices; rank 1 is highest.

Source	Slice	Math avg	Math rank	Code avg	Code rank	CN avg	CN rank	General avg	General rank	Overall avg	Overall rank
MegaMath-Code	random	1.22	5	11.80	1	30.12	17	47.84	17	34.46	1
MegaMath-Web-Pro	random	2.46	3	4.50	18	30.23	11	48.97	3	34.34	2
DCLM-Dedup	q00	0.94	14	3.42	35	30.21	14	49.54	1	34.34	3
FineWeb-Edu-EN	q00	0.83	19	3.80	29	30.21	14	49.33	2	34.24	4
Nemotron-CC-Math	random	2.78	2	4.71	13	29.80	30	48.57	7	34.11	5
Nemotron HQ Synthetic	random	0.75	24	5.88	6	30.67	4	48.50	8	34.08	6
Swallow-Code-v2	q00	0.76	22	9.66	2	29.45	38	47.75	21	33.97	7
AutoMathText	random	1.02	9	5.75	7	30.02	23	48.43	10	33.96	8
Nemotron HQ	random	0.89	15	4.52	16	29.81	29	48.72	5	33.93	9
FineWeb-Edu-EN	q50	1.12	7	3.80	29	29.75	34	48.81	4	33.91	10
DCLM-Dedup	q25	1.12	6	5.41	9	29.72	35	48.46	9	33.91	11
Swallow-Code-v2	q75	0.55	36	9.36	3	30.23	12	47.31	32	33.74	12
DCLM-Dedup	q50	0.63	30	3.22	37	30.09	21	48.58	6	33.67	13
Swallow-Code-v2	q25	0.56	34	9.24	4	29.80	32	47.32	31	33.67	14
FineWiki-EN	random	0.84	18	4.71	13	30.27	10	48.14	13	33.66	15
SwallowMath-v2	random	2.90	1	3.85	27	30.22	13	47.75	20	33.58	16
DCLM-Dedup	q75	1.03	8	4.11	22	30.73	3	47.96	14	33.56	17
FineWeb-Edu-EN	q25	0.53	37	4.11	22	30.12	18	48.18	11	33.54	18
Nemotron HQ	random (medium-high-quality)	0.96	11	3.83	28	29.94	25	48.15	12	33.52	19
StarCoder Tokens	random	0.59	32	5.44	8	29.80	31	47.83	18	33.47	20
FineWeb-Edu-EN	q75	0.80	21	3.91	26	30.36	9	47.96	15	33.45	21
Cosmopedia-v2	random	0.96	12	5.28	12	30.66	5	47.46	29	33.39	22
Algebraic Stack Train	random	0.67	28	4.30	20	30.07	22	47.81	19	33.36	23
FineMath	random	1.67	4	3.47	34	29.68	37	47.84	16	33.35	24
FineWeb-Edu-CN	q25	0.68	27	4.58	15	30.43	7	47.58	24	33.30	25
StackExchange	random	0.49	39	5.36	10	29.70	36	47.53	26	33.26	26
Nemotron Synthetic Code	random	0.61	31	8.88	5	29.84	28	46.69	38	33.26	27
Stack-v2-Smol	random	1.00	10	5.36	10	30.15	16	47.30	33	33.25	28
OpenWebMath	random	0.85	17	4.00	25	29.96	24	47.64	23	33.22	29
FineWeb-Edu-CN	q00	0.69	26	3.63	33	32.13	1	47.24	35	33.21	30

Table 25 – continued

Source	Slice	Math avg	Math rank	Code avg	Code rank	CN avg	CN rank	General avg	General rank	Overall avg	Overall rank
FineWeb-Edu-CN	q50	0.57	33	4.30	20	30.11	19	47.52	27	33.18	31
Deduplicated merged Chinese web	q00	0.65	29	3.30	36	30.80	2	47.56	25	33.17	32
FineWeb-Edu-CN	q75	0.94	13	4.52	16	29.45	39	47.40	30	33.10	33
ArXiv	random	0.86	16	3.72	32	29.80	33	47.51	28	33.09	34
Deduplicated merged Chinese web	q50	0.76	22	4.38	19	30.52	6	47.13	36	33.04	35
FLAN	random	0.72	25	2.11	39	29.89	27	47.70	22	32.98	36
Deduplicated merged Chinese web	q75	0.53	38	3.80	29	30.39	8	47.24	34	32.97	37
PES2O	random	0.56	35	4.11	22	30.11	19	47.02	37	32.84	38
MegaMath-Web	random	0.81	20	2.77	38	29.93	26	46.65	39	32.46	39