MORE GPUS ARE NOT ALWAYS BETTER: SCALING PRETRAINING WITH FINITE DATA

Anonymous authors
Paper under double-blind review

ABSTRACT

We study pretraining when compute is abundant but unique training data are finite, requiring repeated training on the same data. We first revisit infinite-compute pretraining (Kim et al., 2025) and show that its frontier depends critically on choices previously fixed or studied only in isolation. In particular, optimizing batch size together with epoch counts reveals a U-shaped relationship between validation loss and batch size, favoring smaller batches than typical pretraining, likely because small-batch gradient noise provides implicit regularization. We further find that training recipes, previously optimized under a single-pass assumption, can be tuned, e.g., repacking documents into new sequences every epoch, weight tying, and positive embedding weight decay.

We then relax the infinite-compute assumption. In particular, smaller-than-typical batch sizes can require prohibitively long sequential training, motivating to treat GPUs and time as separate resources. Tighter deadlines favor larger batches, and latency-oriented parallelism can trade additional GPUs for shorter training. Nevertheless, even under a fixed time budget, a smaller batch with fewer GPUs can achieve lower validation loss than a larger batch with more GPUs, overturning the conventional preference for using as many GPUs as possible under a deadline. Together, our results identify underexplored factors in repeated-data training and provide concrete guidance on how training recipes and resource allocation should change from single-pass pretraining.

1 INTRODUCTION

Most language-model pretraining assumes a single pass over a large, diverse web corpus, with compute as the binding resource and data treated as effectively unlimited (Kaplan et al., 2020; Hoffmann et al., 2022). However, the supply of high-quality human-written data is finite and grows at a physically limited rate (Villalobos et al., 2024), so continued scaling increasingly requires training repeatedly on the same data (Muennighoff et al., 2023). Prior work asks how much performance can be extracted from fixed data when compute is effectively unlimited (Kim et al., 2025; Xu et al., 2026).

We revisit this setting and show that the infinite-compute frontier depends critically on choices previously fixed or studied only in isolation. In particular, jointly optimizing batch size and training horizon reveals a U-shaped relationship between validation loss and batch size, with the optimum at smaller batches than in conventional pretraining. The gradient noise from smaller batches, which can be undesirable in single-pass training, likely provides useful implicit regularization against repeated-data overfitting. Batch tuning extends beneficial repetitions from 8 to 20 for a 1.5B model and from 40 to 112 for a 153M model, both with substantial loss improvements (Figure 1(a), blue).

We further find that training recipes, which are typically optimized under a single-pass assumption, can be adapted specifically for repeated-data training. We evaluate three simple practices. REPACK—repacking documents into new sequences each epoch—creates fresh sequences even when the underlying data remain fixed, acting as a form of repeated-data regularization. Tying the input and output embeddings and applying positive weight decay to the shared parameters provide additional gains. Together, these choices improve validation loss and shift the best checkpoint from epoch 20 to 32 for the 1.5B model and from epoch 112 to 240 for the 153M model (Figure 1(a), blue vs. orange).

We next move beyond the infinite compute to consider compute resources, as compute is not truly infinite in practice. In particular, the improved frontier can require prohibitively long sequential

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.

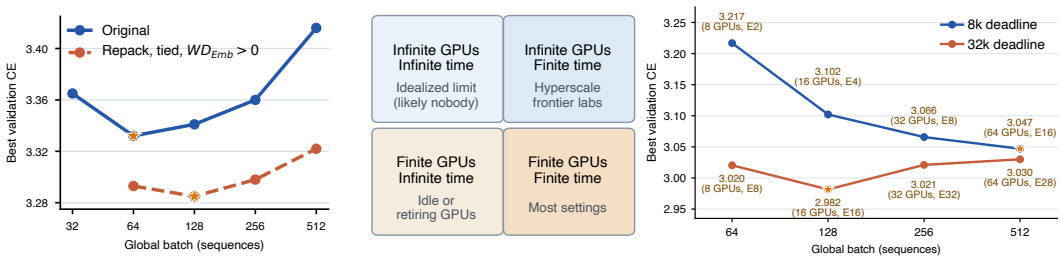

Figure 1: **(a)** Batch-tuned infinite-compute frontier for a 153M model on the 1B-token data pool. Batch tuning and the repetition-tuned recipe progressively improve the validation loss. **(b)** Separating GPU availability from training time yields four distinct compute regimes. **(c)** An example with a 1.5B model on the 1B-token pool: even when time-constrained, a smaller batch size with fewer GPUs can achieve lower validation loss than a larger batch size with more GPUs when compute is abundant.

training, as it favors smaller-than-typical batch sizes. For example, our fitted extrapolation suggests that, for a 70B model trained on 10T unique tokens, reaching the near-optimal stopping point could require over 20 years. Motivated by this, we model GPU availability and sequential training time as separate constraints (Figure 1(b)). Under short deadlines, larger batches process more data per optimizer step; as more time becomes available, progressively smaller batches become preferable until the unconstrained optimal $B^\star$ is reached. Nevertheless, even under a fixed time budget, a smaller batch with fewer GPUs can achieve lower validation loss than a larger batch with more GPUs, especially with longer deadlines (Figure 1(c)). This suggests that the common practice of maximizing parallelism with more GPUs under a fixed deadline may be suboptimal for repeated-data training.

In summary, we ask: *when unique data are fixed but compute resources vary, how should we train?* We answer this question in three stages:

1. We show that repeated-data training has its own statistical batch-size optimum (§3), which also motivates decomposing compute into GPUs and time.
2. We show that the training recipe itself can be adapted to repeated-data training by varying how data are presented across repetitions and regularizing the embeddings (§4).
3. Finally, we map the resulting optimal recipes to the available GPUs and time, showing that (1) although the unconstrained optimum can be very slow, we can jointly optimize repetitions, batch size, and compute configuration under GPU and time budgets, and (2) it can often overturn conventional decisions (§5).

2 Related Work

Data-infinite scaling. Conventional language-model scaling laws predict loss from model size, training tokens, and compute, then use these relationships to select compute-optimal model–data allocations (Kaplan et al., 2020; Hoffmann et al., 2022). They effectively assume an unlimited supply of fresh data, likely because available compute at the time had not yet reached the scale needed to exhaust the available stock of human-generated web data.

Finite-data, infinite-compute pretraining. As hardware improves and the stock of high-quality human-written text grows slowly, unique data increasingly becomes a binding resource (Villalobos et al., 2024). This has motivated repeated-data pretraining, in which models train on a fixed dataset until validation gains saturate or reverse because of overfitting (Muennighoff et al., 2023; Xue et al., 2023). At fixed compute, Fang et al. (2025) show that repeatedly training on filtered data can outperform training on a $10\times$ larger superset with proper regularization, e.g., weight decay tuning.

The infinite-compute, finite-data setting is studied most directly by two recent papers. Kim et al. (2025) fix the unique data, remove the compute constraint, and show that stronger regularization, especially weight decay tuning, substantially delays overfitting. Xu et al. (2026) build on this work and propose additional regularization methods and an interaction-aware scaling law. We revisit the same setting but optimize previously fixed recipe dimensions and then ask how the resulting statistical frontier changes under separate GPU and time constraints.

Batch size. Batch size optimization in conventional single-pass pretraining balances statistical efficiency and parallelism. At a fixed number of optimizer steps, larger batches process more tokens and are generally favored; at a fixed token budget, smaller batches take more steps and can be more token-efficient. Prior work finds diminishing returns to larger batches through critical batch size (McCandlish et al., 2018; Kaplan et al., 2020; Zhang et al., 2025; Merrill et al., 2025). However, reduced token efficiency alone does not imply worse loss under a fixed deadline, since larger batches process more tokens in the available time. Therefore, conventional recipes often increase batch size up to the critical batch size to expose parallelism.

For repeated-data training, batch size has received less attention. Kim et al. (2025) select a small batch using a single-pass ablation and then hold it fixed in their repeated-data scaling experiments; Xu et al. (2026) likewise use a fixed batch throughout.

We instead fix the unique-data pool and jointly tune the learning rate, weight decay, and repetitions for each batch size. We find that repeated-data training changes the conventional tradeoff: smaller batches can achieve lower validation loss, even under a fixed deadline.

3 Batch Size Shapes the Repeated-Data Frontier

We first revisit the unconstrained repeated-data frontier by jointly optimizing batch size, learning rate, weight decay, and epoch count, and reveal an interior batch-size optimum. Because batch size controls both statistical behavior and available parallelism, this later motivates separating GPU and time constraints in §5.

3.1 Experimental Setup

Models & data. We train dense decoder-only Transformers with 153M, 474M, and 1.5B parameters, using the olmo-core (Groeneveld et al., 2024) codebase. All experiments use DCLM-Baseline (Li et al., 2024) as the training corpus, with a 4,096-token context and document-disjoint training, validation, and test splits. This section uses a 1B-token unique-data pool; §4.2 expands the study to 0.333B-, 1B-, and 3B-token pools. Together, the measured grid covers D_{u}/N from 0.22 to 19.61. We train with AdamW (Loshchilov & Hutter, 2019) and a warmup-stable-decay (WSD) learning-rate schedule (Hu et al., 2024; Wen et al., 2024), with a 100M-token warmup and 10% terminal decay.

Hyperparameter tuning methodology. For a model with N parameters and D_{u} unique tokens, E passes yield $D_{\mathrm{train}} = ED_{\mathrm{u}}$ token presentations. We sweep global batches of 16–512 sequences for the 1.5B model and 32–512 for the smaller models.

For each (N, D_{u}, B), we tune the peak learning rate and weight decay through gradual adaptive coordinate search: we first select the pair for an earlier epoch, then proceed to the next epoch by starting from the pair selected for the previous epoch and including neighboring learning rates ($1/2\times$ and $2\times$) and a weight-decay value one level higher ($3\times$).

We seek the best training recipe at a fixed model size, rather than allowing parameter scaling or ensembling as an alternative route to lower loss (Kim et al., 2025). This choice reflects the practical importance of inference cost.

Evaluation. Our primary metric is next-token cross-entropy on a fixed 100M-token DCLM-Baseline validation split. Evaluation on downstream tasks are reported in §B.

3.2 Results

Batch size has an interior optimum. Figure 2 shows that batch-size tuning has a substantial effect. Relative to the best $B = 512$ run, the best $B = 64$ run reduces validation CE from 3.124 to 2.977 for the 1.5B model, from 3.136 to 3.093 for the 474M model, and from 3.416 to 3.332 for the 153M model. For all three model sizes, the complete sweep is U-shaped: validation CE is minimized near $B = 64$, with both larger and smaller batches performing worse.

This optimum is notable because $B = 64$ is substantially smaller than the batches used in public pretraining recipes at a similar scale: GPT-2, Cerebras-GPT, Pythia, and OLMo use global batches ranging from 512 to 2,048 sequences (Radford et al., 2019; Dey et al., 2023; Biderman et al., 2023; Groeneveld et al., 2024).

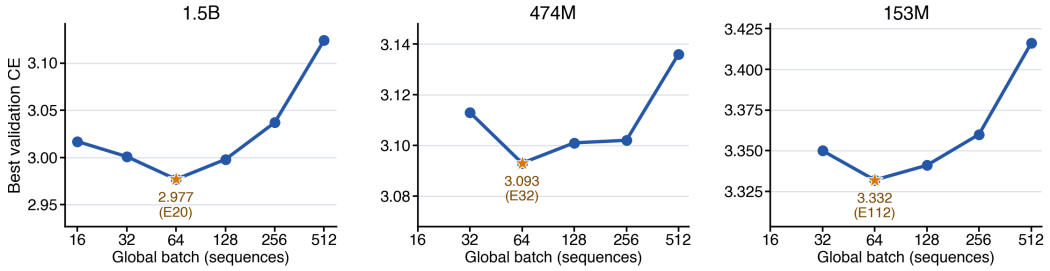

Figure 2: **Batch size has an interior optimum on the 1B-token pool.** At each (model size, batch size) pair, we tune (learning rate, weight decay, epoch count), and report the lowest validation CE.

The gap remains when compared with predictions from more recent research on batch-size scaling. For comparison, the single-pass laws of Bi et al. (2024) and Shuai et al. (2024) predict $B_{\text{opt}} = 0.2920 C_{\text{F}}^{0.3271}$ and $B_{\text{opt}} = 6.42 \times 10^{3} C_{\text{F}}^{0.102}$, respectively, with C_{F} in FLOPs and B in tokens. Our 1.5B model reaches its best observed repeated-data loss after 20 passes over the 1B-token pool, which is equivalent to 1.8×10^{20} FLOPs. At this compute scale, the two laws predict batches of 1.233M and 0.747M tokens, respectively—4.7 and 2.9 times our selected batch of 0.262M tokens (Table 1). This suggests that the batch size derived from single-pass training may be suboptimal for repeated-data regime, and the repeated-data optimum can be substantially smaller.

Table 1: **Batch-size comparison for the 1.5B model on the 1B-token pool.**

	C_{F} (equiv. E)	Batch size (million tokens)
Bi et al.	0.90×10^{19} ($E = 1$)	0.463
Bi et al.	7.20×10^{19} ($E = 8$)	0.914
Bi et al.	18.0×10^{19} ($E = 20$)	1.233
Shuai et al.	0.90×10^{19} ($E = 1$)	0.551
Shuai et al.	7.20×10^{19} ($E = 8$)	0.681
Shuai et al.	18.0×10^{19} ($E = 20$)	0.747
Ours	∞ (C_{F}: $E = 20$)	0.262

Optimization and generalization jointly shape the U-curve. Then, whey can't batch size too large or too small? Figure 3 compares training loss, validation loss, and their gap across batch sizes for the 1.5B model.

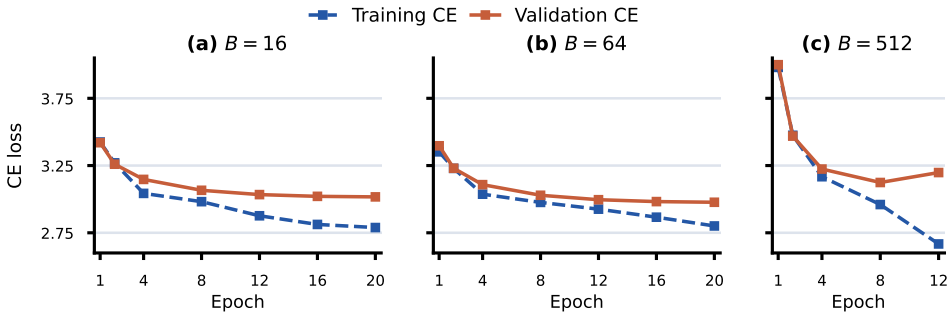

Figure 3: **Training and validation losses across batch sizes.** The large batch ($B = 512$) sees a larger train-validation gap and earlier validation degradation.

- $B = 512$ vs. $B = 64$: Early in training, $B = 512$ is under-optimized because it takes fewer optimizer steps per epoch. Later, however, its training loss drops below that of $B = 64$, so under-optimization can no longer explain the difference. Its validation loss still remains worse and eventually increases. This pattern is consistent with weaker stochastic regularization at large batch sizes (Keskar et al., 2017; Smith et al., 2020).
- $B = 16$ vs. $B = 64$: Their training losses are similar early in training. At later epochs, $B = 16$ achieves lower training loss but higher validation loss than $B = 64$. Thus, below $B = 64$, the additional optimizer steps eventually lead to overfitting: they continue to reduce training loss without improving held-out performance.

Together, the U-curve reflects a balance between optimization and generalization. Relative to conventional large batches, smaller batches benefit from stronger stochastic regularization and delay overfitting; below the optimum, additional optimizer steps stop improving the validation loss.

Table 2: **(a) Data-loader ablation** and **(b) weight-tying and embedding-decay ablation**. Best validation loss and its selected epoch are reported. Learning rate, weight decay, and epoch counts are tuned separately for each configuration.

Loader	$B = 64$	$B = 512$
FIXED	3.011 (E16)	3.205 (E8)
SHUFFLE	2.977 (E20)	3.124 (E8)
REPACK	**2.971 (E24)**	**3.064 (E24)**

Loader	Method	$B = 64$	$B = 512$
SHUFFLE	Untied	2.977 (E20)	3.124 (E8)
REPACK	Untied	2.971 (E24)	3.064 (E24)
REPACK	Tied, $\text{WD}_{\text{Emb}} = 0$	2.971 (E24)	3.061 (E20)
REPACK	Untied, $\text{WD}_{\text{Emb}} > 0$	2.951 (E32)	3.050 (E28)
REPACK	Tied, $\text{WD}_{\text{Emb}} > 0$	**2.946 (E32)**	**3.030 (E28)**

4 TRAINING CHOICES SHIFT THE REPEATED-DATA FRONTIER

In addition to batch size, we show that recipe details, typically optimized for single-pass training or chosen incidentally, can be tuned specifically for repeated-data training. We aim to improve the frontier across batch sizes rather than for the optimal batch size only, since we later show that larger-than-optimal batches can be preferred under time constraints (§5).

4.1 CONTROLLED RECIPE ABLATIONS

Data presentation across epochs. When a finite data is repeated, the data loader determines whether the model repeatedly encounters identical packed sequences or new combinations of documents and contexts. We compare three methods with progressively more cross-epoch variation:

1. FIXED: keeping packed sequences, minibatch composition, and minibatch order fixed across epochs, as in Kim et al. (2025);
2. SHUFFLE: keeping sequence boundaries fixed while reshuffling sequences across minibatches, producing fresh batches each epoch; and
3. REPACK: reshuffling documents and dynamically repacking them into fixed-length sequences, producing both fresh sequences and fresh batches each epoch.

These methods are equivalent in a single pass but expose models to different examples once data are repeated. Prior repeated-data studies use either FIXED (Kim et al., 2025) or fixed minibatch composition with shuffled minibatch order (Fang et al., 2025; Xu et al., 2026). We test whether the additional variation introduced by REPACK acts as repeated-data regularization.

Table 2(a) shows results for a 1.5B model with $B = \{64, 512\}$. FIXED clearly underperforms SHUFFLE in validation loss and exhibits a larger train-validation gap at $B = 512$. REPACK further improves on SHUFFLE at both batch sizes and reaches its best observed validation loss later in training, with a clearer benefit at $B = 512$ than at $B = 64$.

Tied input/output embeddings and embedding decay. Weight tying shares one parameter matrix between the input embeddings and output head and was originally proposed for parameter reduction (Press & Wolf, 2017). We ask whether tying and decay on the shared embedding affect repeated-data training. We fix the dataloder to REPACK and evaluate $\{\text{untied model, tied model}\} \times \{\text{no embedding decay, positive embedding decay}\}$.

Table 2(b) shows that weight tying alone yields little to no improvement, while embedding decay consistently improves validation loss. Combining weight tying and positive embedding decay yields the best results, improving the validation loss from from 2.971 to 2.946 at $B = 64$ and from 3.064 to 3.030 at $B = 512$, relative to the original setup of untied embeddings and no embedding decay. It also reaches its best validation loss later, shifting the selected epoch from 24 to 32 and 28, respectively.[1]

4.2 COMPLETE RESULTS WITH THE REPETITION-TUNED RECIPE

Complete original vs. repetition-aware recipe comparison. Figure 4 reports a complete comparison between the original (SHUFFLE, untied embeddings, and no embedding decay) and repetition-

[1]Weight tying reduces the 153M, 474M, and 1.5B models to approximately 102M, 372M, and 1.3B parameters, respectively; we retain the original names for consistency. Because the untied models contain more parameters, this difference does not explain the tied configuration's lower validation loss.

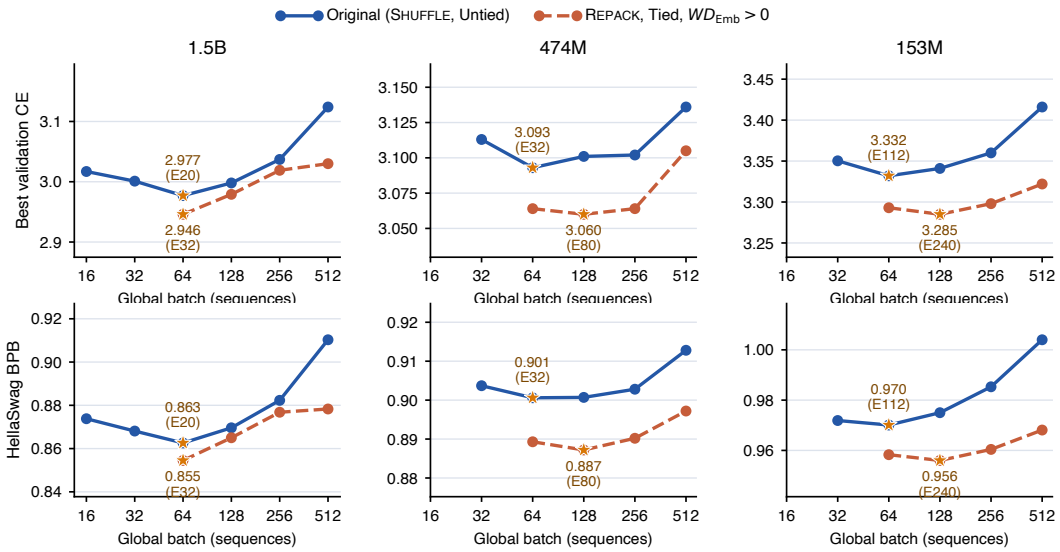

Figure 4: **Comparison between the original and repetition-tuned recipes (REPACK, tied embeddings, positive embedding decay).** The repetition-tuned recipe improves both validation CE (top) and HellaSwag BPB (bottom) across model sizes and batch sizes.

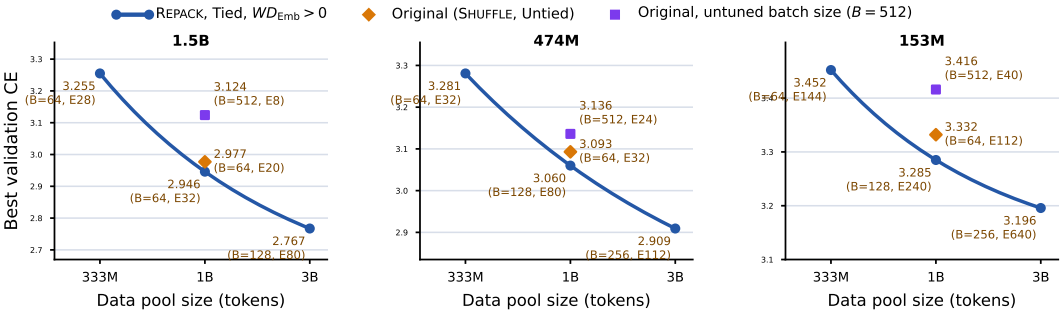

Figure 5: **The repetition-tuned recipe improves as the unique-data pool grows.** We report the best observed validation CE on 0.333B-, 1B-, and 3B-token pools. Orange diamonds show the tuned original recipe on the 1B-token pool, and purple squares show its untuned $B = 512$ baseline. Labels give the selected batch size and epoch.

aware (REPACK, tied embeddings, and positive embedding decay) recipes across model sizes and batch sizes. The repetition-aware recipe improves validation loss and HellaSwag BPB across all three model sizes and extends the validation-selected training horizon. The selected epoch moves from 20 to 32 for the 1.5B model, from 32 to 80 for the 474M model, and from 112 to 240 for the 153M model. The recipe also shifts the optimal batch size from 64 to 128 for the 474M and 153M models. One possible explanation is that the additional regularization reduces the marginal regularization benefit of small-batch gradient noise.

Applying the finalized recipe across data-pool sizes. Having established the repetition-aware recipe, we fix it for the rest of the experiments and extend our analysis to three unique-data pools: 0.333B, 1B, and 3B tokens. We tune the learning rate on the 1B-token pool and transfer it to the 0.333B- and 3B-token pools, following prior evidence that the optimal learning rate is relatively stable across data-pool sizes (Kim et al., 2025; Xu et al., 2026); our preliminary experiments show the same pattern. We also tune weight decay with the assumption that larger data pools use weight decay values no higher than those used for smaller pools.

Figure 5 shows that more unique data improves the best observed validation loss and shifts the selected configuration toward larger batches and later epochs. This is likely because larger pools are less data-constrained: overfitting begins later and additional repetitions remain useful for longer.

5 SCALING ACROSS GPU AND TIME BUDGETS

We now relax the infinite-compute assumption and consider realistic resource constraints. In particular, smaller batches can improve repeated-data generalization but require more sequential training, motivating to treat GPU and training time as separate constraints.

5.1 MOTIVATION: WHY SEPARATE GPUS AND TIME?

As an illustrative extrapolation, consider a dense $N = 70$B model trained on $D_{\mathrm{u}} = 10$T unique tokens. We fit the loss curve and define $E^\star$ as the earliest epoch within 0.001 CE of the fitted minimum, then estimate the corresponding batch size. The fitting procedure and extrapolation are detailed in §D.1. Note that this example is an extreme extrapolation far beyond our measured model and data scales; we therefore treat the resulting numbers as illustrative rather than conclusive.

The fit gives $E^\star \approx 214.2$ and $B^\star \approx 1{,}536$. At sequence length 4,096, this is 340.1 million optimizer steps. Assuming $a = 1$, $m = 1$, $t = 4$, $p = 1$, and an optimistic step latency of $\tau(70) \approx 2.10$ seconds,[2] this requires

$$G^\star \approx 6{,}144 \text{ H100 GPUs} \qquad T^\star \approx 22.6 \text{ years}.$$

This motivates the four regimes in Figure 1(b).

1) Infinite GPUs, infinite time. If a lab has sufficient GPUs and time, it can execute the infinite-compute recipe directly.

2) Finite GPUs, infinite time. With fewer GPUs but sufficient time, gradient accumulation can be used to recover the same recipe with fewer GPUs and longer training.

3) Infinite GPUs, finite time. If GPU availability is effectively nonbinding but training faces a deadline—as may be the case for hyperscale frontier labs—then sequential time becomes the primary constraint. Publicly reported training scales suggest that this regime is already plausible: Llama 4 reports using 32,000 GPUs, and Grok 3 reportedly used approximately 100,000 H100s (Meta AI, 2025; Shilov, 2025). Thus, our extrapolated requirement of 6,144 H100s may be attainable for large AI labs; however, the projected 22.6-year training time would likely remain a constraint.

In this regime, the unconstrained recipe generally cannot be used as is. More GPUs can shorten training through larger batches or model parallelism, but larger batches can worsen validation loss and model parallelism adds overhead. We study this regime in §5.2.

4) Finite GPUs, finite time. When both resources bind, as is likely in most settings, the stopping epoch, batch size, and compute configuration must be selected jointly for the available budgets. We study this joint GPU-time-constrained regime in §5.3.

5.2 TIME-CONSTRAINED SCALING

When training is deadline-constrained, three coupled choices determine the attainable loss:

1. the number of data repetitions (equivalently, the stopping epoch E),
2. the global batch size B, and
3. the compute-parallelism configuration, including the microbatch size and the DP, TP, and PP degrees.

Stopping early while retaining the unconstrained batch $B^\star$ may expose the model to too little data. On the other hand, increasing the batch processes more data per optimizer step, but can eventually yield worse validation loss as repeated-data overfitting emerges. This motivates a deadline-aware recipe that jointly optimizes these choices.

Short deadlines favor larger batches. Figure 6 shows this tradeoff systematically across training recipes, epoch counts E, and batch sizes B, across all three model sizes. Under short budgets, larger batches tend to be preferred because they process more data per optimizer step. As the

[2] Systems assumptions are detailed in §F.

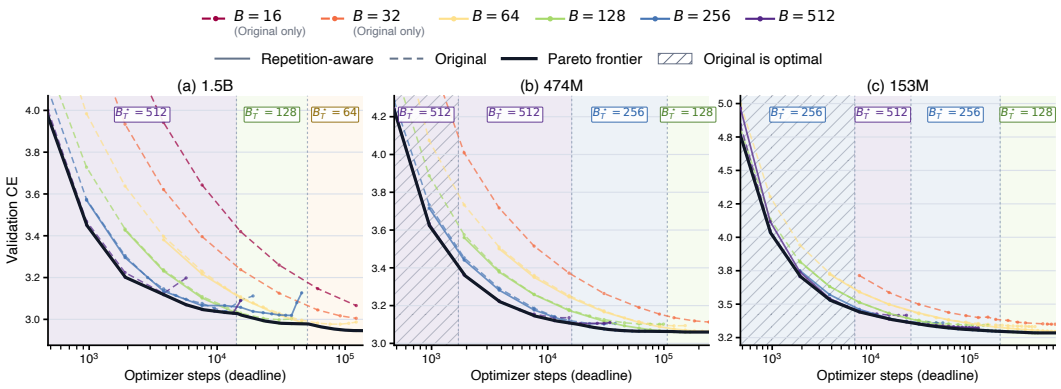

Figure 6: **The time-constrained frontier jointly selects the training recipe, batch size, and the number of repetitions on the 1B-token data pool.** The black curve shows the best validation CE attainable at each optimizer-step deadline; $B_T^\star$ indicates the batch that supplies the frontier.

budget increases, progressively smaller batches become preferable, until the frontier reaches the best observed unconstrained batch $B^\star$.

Importantly, a fixed time budget does not imply that larger batches are always preferable. Figure 1(c) illustrates this for the 1.5B model. Under a short deadline (equivalent to 8K steps), a larger batch using 64 GPUs achieves the best loss. As the deadline increases (equivalent to 32K steps), however, a small batch using only 16 GPUs achieves a lower loss, despite using strictly fewer GPUs. This indicates that maximizing available GPUs by increasing the global batch size can be suboptimal.

Deadlines also change the preferred training recipe. For small models (474M and 153M), ORIGINAL may be preferred under very short deadlines, likely because repeated-data overfitting has not yet become relevant. The repetition-aware recipe subsequently becomes optimal and remains so at longer deadlines, including across the entire observed horizon for the 1.5B model. The difference is consistent but minor, likely because it primarily helps by delaying overfitting and increasing the number of beneficial repetitions.

Parallelism can instead be spent on latency. Batch size is not the only way to trade compute for training time: additional GPUs can reduce latency without changing the global batch through smaller microbatches and more aggressive model parallelism. In a simulated, communication-aware H100 model (§D.2), reducing the microbatch from 8 to 1 yields a projected $2.7\times$ speedup for the 1.5B model; at the extrapolated 70B scale, $8\times$ more model parallelism yields a $3.75\times$ speedup. Because these are simulated rather than observed latency improvements, we exclude them from the figures and §5.3, and refer readers to §D.2.

5.3 JOINT GPU-TIME-CONSTRAINED SCALING

Finally, when both resources are constrained, the number of repetitions (E), batch size (B), and compute-parallelism configuration must be selected jointly; gradient accumulation provides an additional mechanism for trading training time against GPU availability.

Figure 7 reports the best evaluated configuration for each GPU and time budget on the 1B-token pool, with time measured in optimizer-step equivalents. We show the resulting frontiers for ORIGINAL (left) and repetition-aware training (right). For both recipes, More GPUs move the selected configuration toward batches that expose greater concurrency, whereas more time favors smaller batches as they generalize better with more repetitions. As more time becomes available, additional GPUs provide only marginal gains, and eventually, configurations using more GPUs perform worse than those using fewer GPUs, consistent with the finding in §5.2.

Comparing the left (ORIGINAL) and right (repetition-aware) panels of Figure 7, we find that repetition-aware training uniformly improves the joint GPU-time frontier. Its gains therefore persist even after each recipe is independently optimized for the available resources.

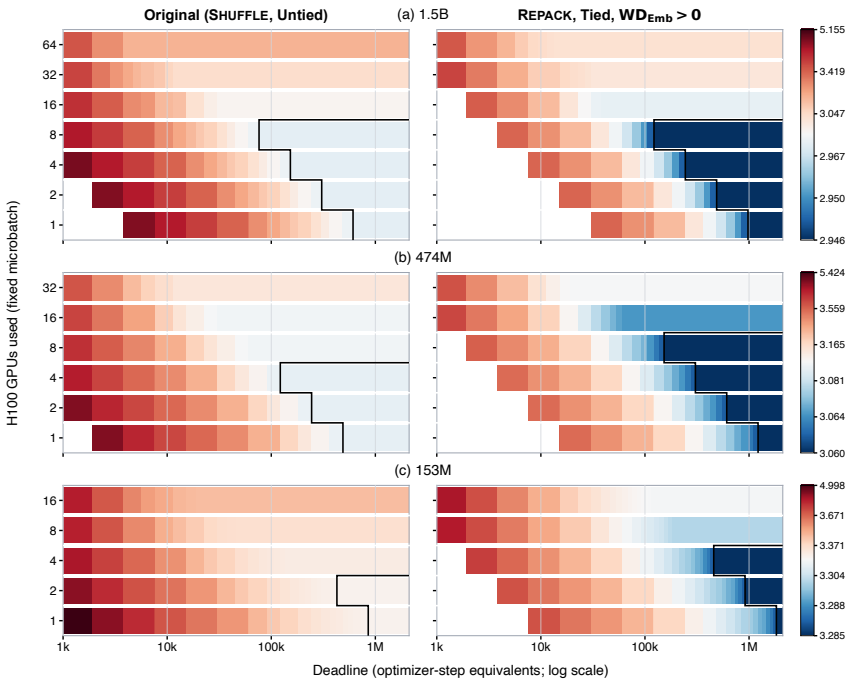

Figure 7: **Joint GPU-time-constrained frontier for the original and repetition-aware recipes on the 1B-token data pool.** Color indicates the validation loss of the best evaluated configuration at each resource budget, normalized separately for each model size on a log scale. The black boundary indicates where the best observed unconstrained loss is reached; beyond it, neither additional time nor additional GPUs improve validation loss.

Finally, the solid black boundary indicates where the best observed unconstrained loss is reached. Beyond it, neither more GPUs nor more time improves the frontier.

6 Conclusion and Limitations

This work studies pretraining when compute is abundant but unique training data are limited, requiring models to train repeatedly on the same data until validation performance approaches its attainable minimum. We begin with the infinite-compute setting studied in prior work (Kim et al., 2025; Xu et al., 2026), but show that batch size, previously underexplored, materially affects the repeated-data frontier. We also show that this frontier can be further improved through repetition-aware training: dynamically repacking sequences each epoch, tying input/output embeddings, and positive weight decay on the shared embeddings.

However, the infinite-compute optimum can require prohibitively many sequential optimizer steps, making it impractical under tighter training deadlines. This motivates decomposing compute into two distinct resources—GPUs and time—and studying regimes in which either or both are binding. Combining observed results across 153M–1.5B-parameter models and 0.333B–3B-token data pools, we map the attainable frontier across GPU and time budgets. Our results show that optimal pretraining requires jointly selecting the number of repetitions, batch size, and training recipe.

Limitations. There are many ways to improve this work; here, we highlight the two most important limitations. First, our observations are limited to 153M–1.5B-parameter models and 0.333B–3B-token data pools. In particular, the illustrative 70B/10T example requires significant extrapolation and should be viewed as illustrative projections rather than estimates that have been rigorously validated at the target scale. Second, our findings may depend on the specific architecture, optimizer and learning-rate schedule (AdamW with WSD), context length, and training data (the relatively heavily filtered DCLM-Baseline corpus). Extending this analysis to mixture-of-experts (MoE) models, for instance, is likely to affect our conclusions and remains a key direction for future work.

AI USE STATEMENT

Generative AI tools were used for coding and debugging, assisting with some experiments, and drafting or refining selected parts of the paper. All AI-assisted outputs were reviewed and verified by the authors. The authors take full responsibility for the methodology, results, and findings.

REPRODUCIBILITY STATEMENT

All code, including exact command lines, will be open-sourced upon acceptance. We report the hyperparameters (learning rate, weight decay, batch size) needed to reproduce every figure and table in §E.

REFERENCES

Akshita Bhagia, Jiacheng Liu, Alexander Wettig, David Heineman, Oyvind Tafjord, Ananya Harsh Jha, Luca Soldaini, Noah A. Smith, Dirk Groeneveld, Pang Wei Koh, Jesse Dodge, and Hannaneh Hajishirzi. Establishing task scaling laws via compute-efficient model ladders. In *Conference on Language Modeling*, 2025. URL https://arxiv.org/abs/2412.04403.

Xiao Bi, Deli Chen, Guanting Chen, et al. DeepSeek LLM: Scaling open-source language models with longtermism. *arXiv preprint arXiv:2401.02954*, 2024. URL https://arxiv.org/abs/2401.02954.

Stella Biderman, Hailey Schoelkopf, Quentin Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, USVSN Sai Prashanth, Edward Raff, Aviya Skowron, Lintang Sutawika, and Oskar van der Wal. Pythia: A suite for analyzing large language models across training and scaling. *arXiv preprint arXiv:2304.01373*, 2023. URL https://arxiv.org/abs/2304.01373.

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. PIQA: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 7432–7439, 2020. doi: 10.1609/aaai.v34i05.6239.

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. *arXiv preprint arXiv:1803.05457*, 2018.

Nolan Dey, Gurpreet Gosal, Zhiming Chen, Hemant Khachane, William Marshall, Ribhu Pathria, Marvin Tom, and Joel Hestness. Cerebras-GPT: Open compute-optimal language models trained on the cerebras wafer-scale cluster. *arXiv preprint arXiv:2304.03208*, 2023. URL https://arxiv.org/abs/2304.03208.

Alex Fang, Hadi Pouransari, Matt Jordan, Alexander Toshev, Vaishaal Shankar, Ludwig Schmidt, and Tom Gunter. Datasets, documents, and repetitions: The practicalities of unequal data quality. In *Advances in Neural Information Processing Systems*, volume 38, 2025. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/2aba6ec20299931d46ddeefd5ddcb442-Abstract-Conference.html.

Samir Yitzhak Gadre, Georgios Smyrnis, Vaishaal Shankar, Suchin Gururangan, Mitchell Wortsman, Rulin Shao, Jean Mercat, Alex Fang, Jeffrey Li, Sedrick Keh, Rui Xin, Marianna Nezhurina, Igor Vasiljevic, Jenia Jitsev, Luca Soldaini, Alexandros G. Dimakis, Gabriel Ilharco, Pang Wei Koh, Shuran Song, Thomas Kollar, Yair Carmon, Achal Dave, Reinhard Heckel, Niklas Muennighoff, and Ludwig Schmidt. Language models scale reliably with over-training and on downstream tasks. *arXiv preprint arXiv:2403.08540*, 2024. URL https://arxiv.org/abs/2403.08540.

Dirk Groeneveld, Iz Beltagy, Pete Walsh, et al. OLMo: Accelerating the science of language models. *arXiv preprint arXiv:2402.00838*, 2024. URL https://arxiv.org/abs/2402.00838.

David Heineman, Valentin Hofmann, Ian Magnusson, Yuling Gu, Noah A. Smith, Hannaneh Hajishirzi, Kyle Lo, and Jesse Dodge. Signal and noise: A framework for reducing uncertainty in language model evaluation. *arXiv preprint arXiv:2508.13144*, 2025. URL https://arxiv.org/abs/2508.13144.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Jack W. Rae, Oriol Vinyals, and Laurent Sifre. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.

Shengding Hu, Yuge Tu, Xu Han, et al. MiniCPM: Unveiling the potential of small language models with scalable training strategies. *arXiv preprint arXiv:2404.06395*, 2024. URL https://arxiv.org/abs/2404.06395.

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. *arXiv preprint arXiv:2001.08361*, 2020.

Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. In *International Conference on Learning Representations*, 2017. URL https://arxiv.org/abs/1609.04836.

Konwoo Kim, Suhas Kotha, Percy Liang, and Tatsunori Hashimoto. Pre-training under infinite compute. *arXiv preprint arXiv:2509.14786*, 2025. URL https://arxiv.org/abs/2509.14786.

Jeffrey Li, Alex Fang, Georgios Smyrnis, et al. DataComp-LM: In search of the next generation of training sets for language models. *arXiv preprint arXiv:2406.11794*, 2024. URL https://arxiv.org/abs/2406.11794.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=Bkg6RiCqY7.

Sam McCandlish, Jared Kaplan, Dario Amodei, and OpenAI Dota Team. An empirical model of large-batch training. *arXiv preprint arXiv:1812.06162*, 2018.

William Merrill, Shane Arora, Dirk Groeneveld, and Hannaneh Hajishirzi. Critical batch size revisited: A simple empirical approach to large-batch language model training. *arXiv preprint://arXiv:2505.23971*, 2025.

Meta AI. The llama 4 herd: The beginning of a new era of natively multimodal ai innovation. Meta AI, 2025. April 5, 2025.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. Can a suit of armor conduct electricity? a new dataset for open book question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 2381–2391, 2018. doi: 10.18653/v1/D18-1260.

Niklas Muennighoff, Alexander M. Rush, Boaz Barak, Teven Le Scao, Aleksandra Piktus, Nouamane Tazi, Sampo Pyysalo, Thomas Wolf, and Colin Raffel. Scaling data-constrained language models. *arXiv preprint arXiv:2305.16264*, 2023.

NVIDIA. Megatron bridge activation-recomputation performance measurements. https://docs.nvidia.com/nemo/megatron-bridge/nightly/skills/nemo-mbridge-perf-activation-recompute/SKILL.html, 2026a. Accessed 2026-08-17.

NVIDIA. Megatron bridge performance summary and archive. https://docs.nvidia.com/nemo/megatron-bridge/latest/performance-summary-archive.html, 2026b. Accessed 2026-08-17.

Ofir Press and Lior Wolf. Using the output embedding to improve language models. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pp. 157–163. Association for Computational Linguistics, 2017. URL https://aclanthology.org/E17-2025/.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. Technical report, OpenAI, 2019. URL https://cdn.openai.com/better-language-models/language-models.pdf.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. WinoGrande: An adversarial winograd schema challenge at scale. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 8732–8740, 2020. doi: 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 *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing*, pp. 4463–4473, 2019. doi: 10.18653/v1/D19-1454.

Anton Shilov. Elon musk confirms that grok 3 is coming soon—pretraining took 10x more compute power than grok 2 on 100,000 nvidia h100 gpus. Tom's Hardware, 2025. January 6, 2025.

Xian Shuai, Yiding Wang, Yimeng Wu, Xin Jiang, and Xiaozhe Ren. Scaling law for language models training considering batch size. *arXiv preprint arXiv:2412.01505*, 2024.

Samuel L. Smith, Erich Elsen, and Soham De. On the generalization benefit of noise in stochastic gradient descent. In *International Conference on Machine Learning*, 2020. URL https://arxiv.org/abs/2006.15081.

Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. CommonsenseQA: A question answering challenge targeting commonsense knowledge. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics*, pp. 4149–4158, 2019. doi: 10.18653/v1/N19-1421.

Pablo Villalobos, Anson Ho, Jaime Sevilla, Tamay Besiroglu, Lennart Heim, and Marius Hobbhahn. Will we run out of data? limits of llm scaling based on human-generated data. In *International Conference on Machine Learning*, 2024.

Ryan Wang, Akshita Bhagia, and Sewon Min. EMO: Pretraining mixture of experts for emergent modularity, 2026. URL https://arxiv.org/abs/2605.06663.

Kaiyue Wen, Zhiyuan Li, Jason Wang, David Hall, Percy Liang, and Tengyu Ma. Understanding warmup-stable-decay learning rates: A river valley loss landscape perspective. *arXiv preprint arXiv:2410.05192*, 2024. URL https://arxiv.org/abs/2410.05192.

Zhiwei Xu, Shihao Wu, Hanseul Cho, Wei Hu, and Yixin Wang. Data-constrained language model pretraining: Improved regularization and scaling laws. *arXiv preprint arXiv:2606.06888*, 2026.

Fuzhao Xue, Yao Fu, Wangchunshu Zhou, Zangwei Zheng, and Yang You. To repeat or not to repeat: Insights from scaling llm under token-crisis. In *Advances in Neural Information Processing Systems*, 2023.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. HellaSwag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 4791–4800, 2019. doi: 10.18653/v1/P19-1472.

Hanlin Zhang, Depen Morwani, Nikhil Vyas, Jingfeng Wu, Difan Zou, Udaya Ghai, Dean Foster, and Sham M. Kakade. How does critical batch size scale in pre-training? In *International Conference on Learning Representations*, 2025.

A Training, Model, and Evaluation Details

A.1 Model Architectures

We train dense decoder-only Transformers with 153M, 474M, and 1.5B parameters using the EMO codebase (Wang et al., 2026), built on olmo-core (Groeneveld et al., 2024). All models use a 4,096-token context, a 100,278-token padded vocabulary, rotary position embeddings with base 500,000, RMSNorm, and SwiGLU feed-forward blocks. We do not use QK normalization. The attention head dimension is 64 for the 153M and 474M models and 128 for the 1.5B model.

Table 3 reports the exact parameter counts and other architectural hyperparameters. After weight tying, which we find beneficial in §4, the parameter counts decrease to approximately 102M, 372M, and 1.3B. For consistency and to avoid confusion, we continue to refer to these models as 153M, 474M, and 1.5B throughout the paper. This convention does not unfairly favor our conclusions: it effectively compares a larger untied baseline against our smaller tied model, which nevertheless performs better. Both our experiments and prior work consistently show that larger models perform better even in the overfitted, repeated-data regime.

Table 3: Model architectures. Untied counts define the nominal model labels; tied counts are the trainable parameter counts used as N in our repetition-aware scaling laws. d_{model} is the residual width and d_{ff} is the SwiGLU feed-forward width.

Model	N_{untied}	N_{tied}	Layers	d_{model}	Heads	d_{ff}
153M	153,104,896	101,724,672	12	512	8	2,048
474M	473,990,144	371,229,696	16	1,024	16	4,096
1.5B	1,484,851,200	1,279,330,304	16	2,048	16	8,192

A.2 Data and Training Details

All training text comes from DCLM-Baseline (Li et al., 2024). We split documents before packing to form the training and validation sets. The validation set contains 100M tokens.

In §3, we reshuffle sequence indices every epoch (defined as SHUFFLE in §4.1), by default. Later in §4.1 we ablate this choice and converge on repacking every epoch (defined as REPACK), and the rest of the paper from there uses REPACK by default. We use seed for the dataloader (12,536 as an offset and adds epoch number) for reproducibility.

We optimize with AdamW (Loshchilov & Hutter, 2019), using $(\beta_1, \beta_2) = (0.9, 0.95)$, gradient clipping at 1.0, and a 10^{-5} z-loss.

We use a warmup–stable–decay (WSD) learning-rate schedule (Hu et al., 2024; Wen et al., 2024), with an approximately 100M-token warmup and a target-specific terminal decay over the final 10% of each run.

We sweep batch sizes of $b \in \{16, 32, 64, 128, 256, 512\}$ sequences for the 1.5B model and $b \in \{32, 64, 128, 256, 512\}$ for the 153M and 474M models. We search over peak learning rates in $\{5 \times 10^{-4}, 10^{-3}, 2 \times 10^{-3}, 4 \times 10^{-3}\}$ and weight decays in $\{0.011, 0.033, 0.1, 0.3, 1.0, 3.0\}$; values outside these ranges were never selected. Because AdamW applies decay once per optimizer step, its cumulative effect depends on B, so we tune learning rate and weight decay jointly with batch size. We use adaptive coordinate search: at each new epoch, the search starts from the previous best configuration and evaluates neighboring learning rates and higher weight decays. We tune the learning rate on the 1B-token pool and transfer it to the 0.333B- and 3B-token pools, following prior evidence that the optimal learning rate is relatively stable across data-pool sizes (Kim et al., 2025; Xu et al., 2026); our preliminary experiments show the same pattern.

A.3 Evaluation Details

Our primary metric is next-token cross-entropy (CE) on a fixed, document-disjoint DCLM-Baseline validation split. §B.3 later discusses efforts on evaluating on downstream tasks instead.

B Additional Results for Batch-Tuned Infinite-Compute Scaling (§3)

B.1 Robustness to Random Seeds

Batch size	LR	WD	Eval. epoch	Pool A	Pool B	Pool C	Across-pool SD
32	5×10^{-4}	0.3	20	3.00052	3.00076	3.00093	0.00021
64	1×10^{-3}	0.3	20	**2.97713**	**2.97733**	**2.97594**	0.00075
128	1×10^{-3}	0.3	16	2.99839	2.99936	2.99628	0.00157
256	2×10^{-3}	0.333	12	3.03689	3.04292	3.04167	0.00318

Table 4: Impact of different random seeds in the 1B-token unique data pools. *SD* indicates standard deviation.

We repeat experiments in §3 for a 1.5B model trained on a 1B-token unique-data pool, using three indepently sampled pools, denoted Pool A, Pool B, and Pool C in Table 4. Across all batches, the standard deviation of validation CE across pools ranges from only 0.00021 to 0.00318. This variation is substantially smaller than the differences between batch configurations, and the optimal batch size is consistent across all three pools (batch size of 64).

B.2 Comparison with Unique-Data Training

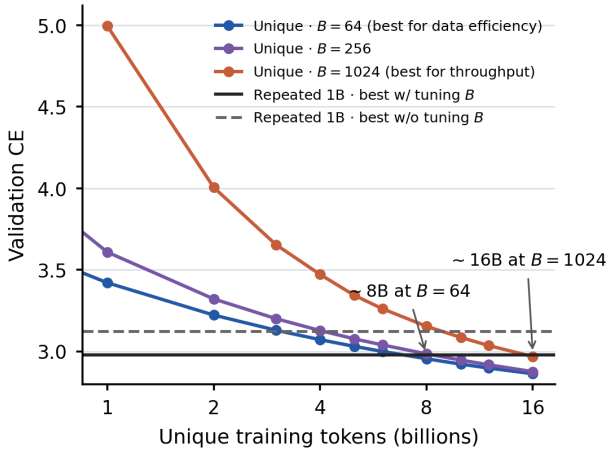

Figure 8: **Unique-data equivalent of repeated-data training.** Curves show validation CE for unique-data training at three batch sizes. Horizontal lines mark the best repeated 1B-token results with and without tuning B. With tuning, repeated-data training matches $6\times$–$8\times$ unique data at $B = 64$ and $\sim 16\times$ at $B = 1024$; without tuning, it matches $\sim 8\times$ at $B = 1024$.

Repeated-data training substitutes for substantially more unique data. Figure 8 compares the best repeated-data endpoints with batch-tuned unique-data curves. At matched $B = 64$, the batch-tuned repeated-data recipe matches training on $\sim 8\times$ more unique data. Against the throughput-oriented $B = 1024$ unique-data curve, it matches $\sim 16\times$ more unique data. Without tuning B, the prior repeated-data recipe matches only $\sim 8\times$ more unique data. Tuning B therefore provides a gain comparable to doubling the unique-data budget.

Prior findings tied to a particular number of epochs are also batch-dependent. For example, the observation that unique and repeated data remain similar for roughly four epochs (Muennighoff et al., 2023) changes with B: larger batches keep the two settings close for more epochs but fewer optimizer steps, while smaller batches keep them close for more optimizer steps but fewer epochs.

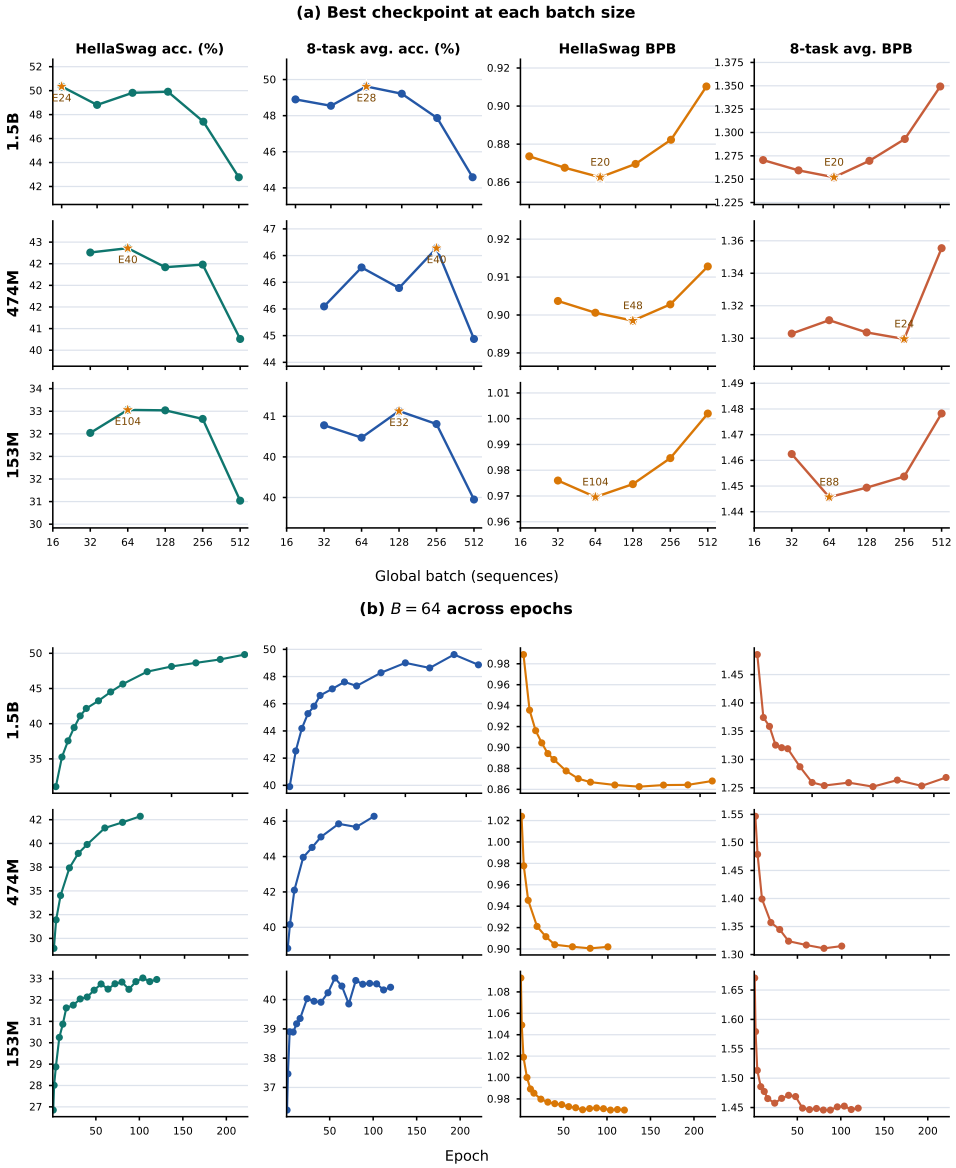

Figure 9: **HellaSwag and aggregate downstream performance.** (a) Performance versus batch size. Each best available epoch was selected based on validation CE. (b) Performance across epochs at $B = 64$.

B.3 Downstream Task Generalization

Making modeling conclusions from downstream performance is substantially more difficult than doing so with validation loss: individual-task accuracies vary considerably across checkpoints, particularly for small models. Recent work therefore predicts task performance indirectly through language-modeling loss, aggregates results across tasks, or uses soft metrics like perplexity or bits per byte (BPB) (Gadre et al., 2024; Bhagia et al., 2025; Heineman et al., 2025). These studies find that task aggregates are more predictable than many individual accuracies and that perplexity and BPB are often more reliable.

We evaluate eight widely used pre-training evaluation benchmarks: HellaSwag (Zellers et al., 2019); ARC-Easy and ARC-Challenge (Clark et al., 2018); CommonsenseQA (Talmor et al., 2019); OpenBookQA (Mihaylov et al., 2018); PIQA (Bisk et al., 2020); SocialIQA (Sap et al., 2019); and WinoGrande (Sakaguchi et al., 2020).

Figure 9 reports performance of three models, 1.5B, 474M, and 153M, on four downstream task metrics: Hellaswag accuracy, aggregated eight-task averaged accuracy, Hellaswag BPB, and aggregated eight-task BPB. The top figure (a) reports the best metric for each batch size, and the bottom figure (b) reports the best metric for each epoch number when $B = 64$.

The main conclusions from the earlier sections continue to hold. Across all model sizes, overly large batches perform poorly; very small batches are not uniformly optimal either; the strongest results tend to occur at intermediate batch sizes.

At the same time, we confirm previous findings that downstream evaluation exhibits substantially greater variance. Its curves are not always convex, as would be expected from a reliable scaling signal. BPB generally produces smoother trends than accuracy, consistent with prior work (Heineman et al., 2025), e.g., convex trends emerge in both BPB panels for the 1.5B and 153M models and in the HellaSwag BPB panel for the 474M model. Whenever the curve is convex, the downstream results closely agree with the validation-loss results: the optimal batch size lies consistently near $B = 64$ across model sizes.

C Additional Results for Repetition-Aware Training (§4)

Our survey of nine representative open-source training stacks (Table 5) shows that their default configurations vary substantially and often differ from our best-performing repetition-aware training. We recommend that researchers and LLM developers treat these choices as explicit design decisions rather than uncritically inheriting a stack's defaults.

Stack	Repeated-data order	Tying	Embed. WD	Head WD
OpenLM	FIXED	No	> 0	> 0
Marin / Levanter	FIXED	No	0	> 0
olmo-core	SHUFFLE	No	0	> 0
LLM Foundry MPT	SHUFFLE	Yes	shared 0	
Hugging Face Trainer	SHUFFLE	Model-specific	0	0
Meta Lingua	SHUFFLE	No	> 0	> 0
Pythia / GPT-NeoX	REPACK	No	> 0	> 0
Megatron-LM	REPACK	Yes	shared > 0	
NeMo Megatron Bridge	REPACK	Model-specific	> 0	> 0 or shared
Ours: data-constrained	**REPACK**	**Yes**	**shared** > 0	

Table 5: Representative open-source defaults and our data-constrained recommendation.

D Additional Results for Unified Scaling (§5)

D.1 Illustrative 70B/10T Repetition and Batch Extrapolation

We use our measured grid to estimate the repetition count and batch size—and hence the time and GPU resources—for a hypothetical 70B model trained on a 10T-token unique-data pool. This is an extreme, illustrative extrapolation rather than a precise target-scale prediction.

Epoch count. We first fit the validation-loss curves only to the tuned repetition-aware trajectories—Repack, tied embeddings, and positive embedding weight decay—for the 3×3 grid of model sizes and unique-data pools in §4. The fit uses the exact tied-embedding trainable counts $N \in \{101{,}724{,}672;\ 371{,}229{,}696;\ 1{,}279{,}330{,}304\}$ from Table 3 and the highest-precision stored validation CE from all 100 eligible checkpoints.

We model validation loss after E passes over the unique pool as

$$\widehat{L}(E, N, D_{\mathrm{u}}) = L_0 + AE^{-\alpha} + B_{\mathrm{rep}}E^{\beta}, \qquad B_{\mathrm{rep}} = \frac{\alpha A}{\beta E_{\min}^{\alpha+\beta}}, \tag{1}$$

where the first term captures useful learning and the second captures degradation from repetition; this parameterization makes $E_{\min}$ the continuous minimizer.

The loss is the mean of the weighted squared errors across the nine grid cells, in order to give each cell equal weight; within each cell, checkpoints are weighted by their local spacing in $\log E$ so that densely sampled early epochs do not dominate. Because the loss curves become very flat near saturation, estimates of $E_{\min}$ can be brittle. We therefore add soft hinge penalties that capture the observed curve shape without introducing synthetic loss values. When the first non-improving checkpoint brackets the turn between evaluated epochs E_a and E_b, the penalty is zero for $E_a < E_{\min} < E_b$ and grows quadratically only when $E_{\min}$ falls outside this interval. For $N = 153$M and $D_u = 3$B, we stopped training at $E = 640$ due to compute constraints, even though the validation loss was still improving; we therefore use a one-sided penalty that increases only when $E_{\min}$ lies below that endpoint. The hinge weight is 2.0.

Let $x = \log(N/371{,}229{,}696)$ and $y = \log(D_{\mathrm{u}}/10^9)$. A joint 15-parameter fit makes the five latent curve parameters affine in (x, y):

$$\begin{bmatrix} L_0 \\ \log A \\ \log \alpha \\ \log E_{\min} \\ \log \beta \end{bmatrix} = \begin{bmatrix} -0.10524 & -0.59081 & -1.51710 \\ 0.60213 & 0.18444 & 0.39532 \\ -1.62858 & -0.04994 & -0.64827 \\ 4.55678 & -0.74715 & 0.54148 \\ -2.80029 & 0.25300 & -0.07690 \end{bmatrix} \begin{bmatrix} 1 \\ x \\ y \end{bmatrix}. \tag{2}$$

The resulting mean within-grid, log-spacing-weighted R^2 for the checkpoint-loss predictions is 0.980.

Next, we use the fitted curves to estimate $E^\star(N, D_{\mathrm{u}})$, the practical saturation epoch for model size N and unique-data pool D_{u}. Because the exact minimizer can be unstable when the loss curve is flat, we define the stopping target as the first integer epoch within 0.001 validation loss of the fitted minimum:

$$E^\star(N, D_{\mathrm{u}}) = \min\left\{E \geq 1 :\ \widehat{L}(E, N, D_{\mathrm{u}}) \leq \widehat{L}(E_{\min}, N, D_{\mathrm{u}}) + 0.001\right\}. \tag{3}$$

This corresponds to a perplexity ratio of at most $\exp(0.001) \approx 1.001$.

To extrapolate beyond the measured grid, we calculate $E^\star$ for each of the nine fitted grid-cell curves and fit a power function to those nine model-derived values:

$$E^\star(N, D_{\mathrm{u}}) = 37.64 \left(\frac{N}{10^9}\right)^{-0.6986} \left(\frac{D_{\mathrm{u}}}{10^9}\right)^{0.5111}. \tag{4}$$

This summary fit has a multiplicative RMS error of $1.011\times$ and log-space $R^2 = 0.9998$ on the nine model-derived targets, and gives $E^\star(70\mathrm{B}, 10\mathrm{T}) = 214.2$, which we round to 214 epochs.

Batch size. For batch size, nine observations are too few to support even this simple fit reliably. We therefore add the three Pool-111M batch optima, producing twelve observations for the batch-size fit only. The Pool-111M runs were continued only far enough to identify a clear batch-size winner for each model; because we did not train every model to saturation on that pool, we exclude those trajectories from the loss and epoch fits above. Ordinary least squares in log space gives

$$B^\star(N, D_\mathrm{u}) = 101.751 \left(\frac{N}{10^9}\right)^{-0.13583} \left(\frac{D_\mathrm{u}}{10^9}\right)^{0.35740} \quad \text{sequences.} \tag{5}$$

Its multiplicative RMS error is $1.29\times$ and its log-space R^2 is 0.768. For a 70B model and 10T unique tokens, this fit extrapolates to approximately $1{,}536$ sequences.

Resulting resource estimate. Rounding the projections to $E^\star = 214$ and $B^\star = 1{,}536$ gives approximately 340.1 million optimizer steps. Under the conventional configuration (Table 6)—gradient accumulation $a = 1$, microbatch size $m = 1$, tensor parallelism $t = 4$, pipeline parallelism $p = 1$, and optimizer-step latency $\tau(70) \approx 2.10$ seconds—this corresponds to approximately 6,144 H100 GPUs and 22.6 years of training.

D.2 From Throughput-First to Latency-First Compute Parallelism

This section describes how explicitly replacing a conventional, throughput-optimal model-parallelism configuration with a latency-optimal configuration can reduce training time—at the cost of using more GPUs—when training is time-constrained.

Conventional throughput-first policy. Large-scale training distributes a global batch across a compute-parallelism configuration defined by the local microbatch size and the data-parallel (DP), tensor-parallel (TP), and pipeline-parallel (PP) degrees.[3] With gradient accumulation fixed to $a = 1$, so that $B_\mathrm{fixed} = dm$, the conventional policy selects the feasible configuration with the highest throughput per GPU:

$$(d^*_\mathrm{th}, t^*_\mathrm{th}, p^*_\mathrm{th}, m^*_\mathrm{th}) = \underset{\substack{d,t,p,m \\ dm=B_\mathrm{fixed} \\ dtp\le G}}{\arg\max} \frac{mS}{tpT_\mathrm{step}(d,t,p,m)}.$$

In practice, this favors the minimum model-parallel degree needed for memory feasibility and satisfactory utilization, a microbatch near the per-GPU throughput optimum, and DP for the remaining batch parallelism.

Latency-first policy. The latency-first policy instead selects the feasible microbatch and DP/TP/PP configuration with the shortest optimizer step while preserving the same global batch:

$$(d^*, t^*, p^*, m^*) = \underset{\substack{d,t,p,m \\ dm=B_\mathrm{fixed} \\ dtp\le G}}{\arg\min} T_\mathrm{step}(d,t,p,m),$$

subject to memory, divisibility, and topology constraints. Additional H100s are assigned to TP or PP only when the reduction in exposed DP communication exceeds the added model-parallel communication and scheduling overhead.

Implementing and benchmarking the full microbatch and DP/TP/PP design space is beyond our scope. Instead, we use the simulated, communication-aware H100 model in §F to estimate optimizer-step latency (T_step) and throughput (tokens/s/GPU) across feasible configurations, and then report the throughput-optimal (left) and latency-optimal (right) configurations in Table 6. These results are fitted projections rather than end-to-end measurements.

Results show that explicitly optimizing for latency yields $1.67\times$–$3.75\times$ shorter optimizer steps across model sizes, at the cost of using more GPUs, through smaller microbatches and more aggressive model parallelism. For example, the latency-optimal 70B configuration reduces step latency by $3.75\times$ by using $8\times$ more GPUs. It is also worth noting that further model parallelism eventually stops improving latency because of communication and other overheads. For example, the 70B latency improvement saturates at $3.75\times$ under our fitted model; beyond that point, additional model parallelism does not reduce latency.

[3] Since our context length is fixed at 4,096 tokens, we leave context parallelism outside the scope.

	Conventional throughput-first H100							Latency-first H100						
Model	m	DP	TP	PP	GPUs	T_{step}	Tok./s/GPU	m	DP	TP	PP	GPUs	T_{step}	Tok./s/GPU
1.5B	8	128	1	1	128	~ 0.60 s	54.6k	1	1024	1	1	1024	~ 0.22 s	18.6k
8B	2	512	1	1	512	~ 1.00 s	8.19k	1	1024	2	1	2048	~ 0.60 s	3.41k
32B	1	1024	2	1	2048	~ 1.95 s	1.05k	1	1024	4	2	8192	~ 0.70 s	0.73k
70B	1	1024	4	1	4096	~ 2.10 s	0.49k	1	1024	8	4	32768	~ 0.56 s	0.23k

Table 6: H100 projections for throughput-first and latency-first compute-parallelism configurations at the same global batch size, $B = 1{,}024$ sequences, with gradient accumulation $a = 1$. Throughput and latency are simulated using a model that accounts for communication overhead. See §F for details.

D.3 Supplementary Materials for Time–GPU-Constrained Scaling

Figure 10 shows the optimal batch size for each combination of GPU and time budgets, complementing Figure 7 in §5.3, which reports the attainable loss across the same resource plane. Larger batches are preferred when more GPUs are available but training time is limited because they expose greater parallelism and require fewer sequential steps. As the time budget increases, the selected batch size eventually converges to the globally optimal batch under unconstrained GPU and time budgets.

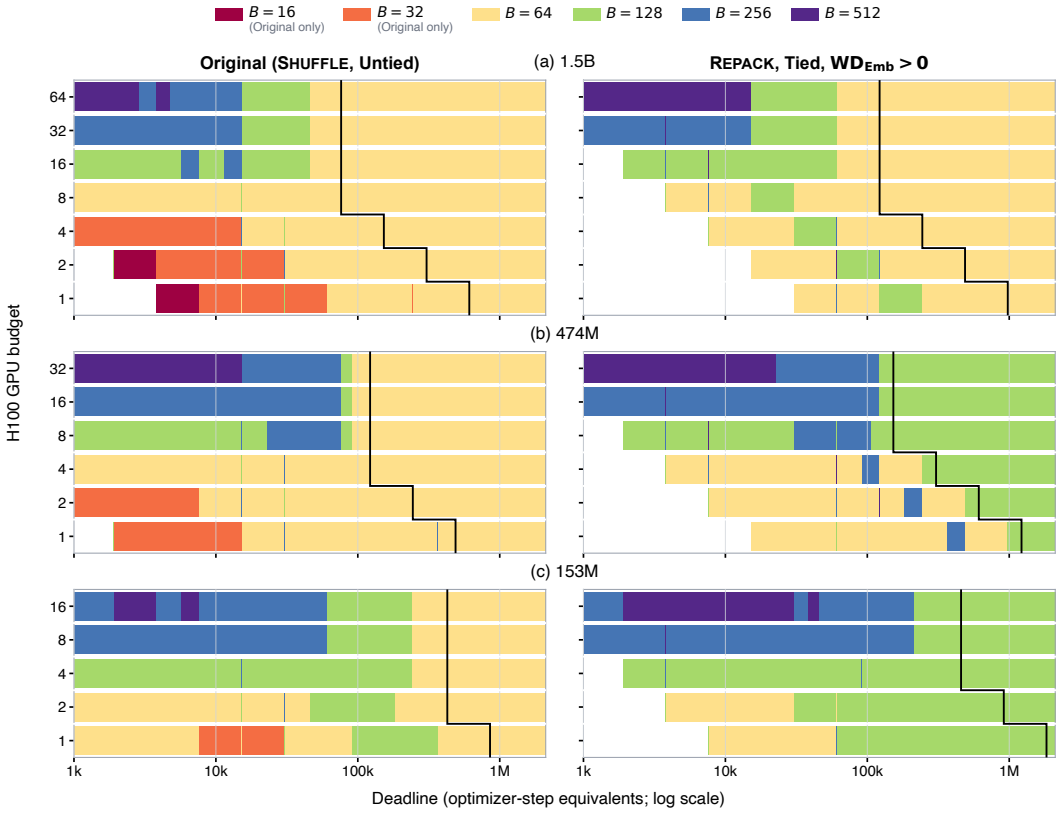

Figure 10: **The optimal batch depends jointly on the GPU budget and deadline on the 1B-token pool, both for the original (left) and the repetition-aware recipe (right).** At each H100 budget and optimizer-step-equivalent deadline, color identifies the batch of the lowest-validation-CE observed configuration that is feasible, allowing gradient accumulation when the global batch exceeds the per-step hardware capacity. The black boundary indicates the region where the globally optimal loss under unconstrained GPU and time budgets is reached; beyond it, moving right (more time) or up (more GPUs) does not improve the validation loss.

E EXPERIMENT CONFIGURATIONS FOR SECTIONS 3 AND 4

We report the exact hyperparameters used in all experiments in the main paper. B is the global batch size in 4,096-token sequences, LR is the peak learning rate, WD is AdamW weight decay, and E is the number of training passes, including terminal WSD decay.

Table 7: Hyperparameters for the batch-size sweep in Figure 2 (§3), all on a 1B-token pool. The 1.5B, $B = 64$ row also supplies our observed result in the batch-size comparison in Table 1. All runs use the original recipe.

1.5B

B	LR	WD	E	CE
16	5×10^{-4}	0.1	20	3.017
32	5×10^{-4}	0.3	20	3.001
64	10^{-3}	0.3	20	2.977
128	10^{-3}	0.3	16	2.998
256	2×10^{-3}	0.3	12	3.037
512	2×10^{-3}	0.3	8	3.124

474M

B	LR	WD	E	CE
32	10^{-3}	0.1	32	3.113
64	2×10^{-3}	0.1	32	3.093
128	2×10^{-3}	0.3	40	3.101
256	2×10^{-3}	0.3	32	3.102
512	2×10^{-3}	0.3	24	3.136

153M

B	LR	WD	E	CE
32	10^{-3}	0.1	88	3.350
64	2×10^{-3}	0.1	112	3.332
128	2×10^{-3}	0.1	72	3.341
256	2×10^{-3}	0.1	48	3.360
512	2×10^{-3}	0.1	40	3.416

Table 8: Examples of selected hyperparameters across training horizons for three batch sizes per model size on the 1B-token pool, corresponding to Figure 2 (§3). All evaluated epochs through the first checkpoint after the validation loss saturates or reverses are shown. Bold marks the best checkpoint. All runs use the original recipe.

1.5B, $B = 512$

E	LR	WD	CE
1	10^{-3}	0.1	4.000
2	10^{-3}	0.3	3.469
4	10^{-3}	0.3	3.224
8	$\mathbf{2\times10^{-3}}$	**0.3**	**3.124**
12	10^{-3}	0.3	3.198

474M, $B = 512$

E	LR	WD	CE
1	2×10^{-3}	0.1	4.235
2	2×10^{-3}	0.1	3.625
4	2×10^{-3}	0.3	3.364
8	2×10^{-3}	0.3	3.226
12	2×10^{-3}	0.3	3.178
16	2×10^{-3}	0.3	3.155
24	$\mathbf{2\times10^{-3}}$	**0.3**	**3.136**
32	2×10^{-3}	0.3	3.136

153M, $B = 256$

E	LR	WD	CE
1	2×10^{-3}	0.033	4.038
2	2×10^{-3}	0.033	3.710
4	2×10^{-3}	0.033	3.531
8	2×10^{-3}	0.033	3.444
12	2×10^{-3}	0.1	3.413
16	2×10^{-3}	0.1	3.392
24	2×10^{-3}	0.1	3.372
32	2×10^{-3}	0.1	3.364
40	2×10^{-3}	0.1	3.361
48	$\mathbf{2\times10^{-3}}$	**0.1**	**3.360**
56	2×10^{-3}	0.1	3.360

Table 9: Hyperparameters for every plotted checkpoint in Figure 3 (§3), with the 1.5B model and a 1B-token pool. Each epoch uses its own selected LR/WD pair. All runs use the original recipe.

(a) $B = 16$

E	LR	WD	CE
1	5×10^{-4}	0.1	3.420
2	5×10^{-4}	0.1	3.260
4	5×10^{-4}	0.1	3.147
8	5×10^{-4}	0.1	3.066
12	5×10^{-4}	0.1	3.034
16	5×10^{-4}	0.1	3.021
20	5×10^{-4}	0.1	3.017

(b) $B = 64$

E	LR	WD	CE
1	10^{-3}	0.3	3.397
2	10^{-3}	0.3	3.230
4	5×10^{-4}	0.3	3.108
8	10^{-3}	0.3	3.029
12	10^{-3}	0.3	2.996
16	10^{-3}	0.3	2.982
20	10^{-3}	0.3	2.977

(c) $B = 512$

E	LR	WD	CE
1	10^{-3}	0.1	4.000
2	10^{-3}	0.3	3.469
4	10^{-3}	0.3	3.224
8	2×10^{-3}	0.3	3.124
12	10^{-3}	0.3	3.198

Table 10: Configurations for the data-loader comparison table in Figure **??**(a) (§4), with the 1.5B model and a 1B-token pool. All runs use untied embeddings and no embedding decay.

$B = 64$

Loader	LR	WD	E	CE
FIXED	10^{-3}	0.3	16	3.011
SHUFFLE	10^{-3}	0.3	20	2.977
REPACK	10^{-3}	0.3	24	2.971

$B = 512$

Loader	LR	WD	E	CE
FIXED	10^{-3}	0.3	8	3.205
SHUFFLE	2×10^{-3}	0.3	8	3.124
REPACK	10^{-3}	1	24	3.064

Table 11: Hyperparameters for every plotted checkpoint in Figure **??**(b) (§4), with the 1.5B model, a 1B-token pool, and $B = 512$. All runs use untied embeddings and no embedding decay.

FIXED

E	LR	WD	CE
1	10^{-3}	0.3	4.000
2	10^{-3}	0.3	3.467
4	10^{-3}	0.3	3.240
8	10^{-3}	0.3	3.205
12	10^{-3}	0.3	3.248

SHUFFLE

E	LR	WD	CE
1	10^{-3}	0.1	4.000
2	10^{-3}	0.3	3.469
4	10^{-3}	0.3	3.224
8	2×10^{-3}	0.3	3.124
12	10^{-3}	0.3	3.198

REPACK

E	LR	WD	CE
1	10^{-3}	0.1	4.000
2	10^{-3}	0.3	3.468
4	10^{-3}	0.3	3.221
8	10^{-3}	0.3	3.137
12	10^{-3}	1	3.091
16	10^{-3}	1	3.069
20	10^{-3}	1	3.065
24	10^{-3}	1	3.064
28	10^{-3}	1	3.067

Table 12: Configurations for the weight-tying/embedding-decay comparison table in Figure **??**(a) (§4), with the 1.5B model and a 1B-token pool.

$B = 64$

Configuration	$\mathrm{WD}_{\mathrm{Emb}}$	LR	WD	E	CE
SHUFFLE, untied	0	10^{-3}	0.3	20	2.977
REPACK, untied	0	10^{-3}	0.3	24	2.971
REPACK, tied	0	10^{-3}	0.3	24	2.971
REPACK, untied	0.3	10^{-3}	0.3	32	2.951
REPACK, tied	0.3	10^{-3}	0.3	32	2.946

$B = 512$

Configuration	$\mathrm{WD}_{\mathrm{Emb}}$	LR	WD	E	CE
SHUFFLE, untied	0	2×10^{-3}	0.3	8	3.124
REPACK, untied	0	10^{-3}	1	24	3.064
REPACK, tied	0	10^{-3}	1	20	3.061
REPACK, untied	1	10^{-3}	1	28	3.050
REPACK, tied	1	10^{-3}	1	28	3.030

Table 13: Hyperparameters for every plotted checkpoint in Figure **??**(b) (§4), with the 1.5B model, a 1B-token pool, $B = 64$, and REPACK.

Untied, no embedding decay

E	LR	WD	CE
1	10^{-3}	0.3	3.397
2	10^{-3}	0.3	3.229
4	10^{-3}	0.3	3.112
8	10^{-3}	0.3	3.027
12	10^{-3}	0.3	2.994
16	10^{-3}	0.3	2.978
20	10^{-3}	0.3	2.972
24	10^{-3}	0.3	2.971

Tied, no embedding decay

E	LR	WD	CE
1	10^{-3}	0.3	3.376
2	10^{-3}	0.3	3.211
4	10^{-3}	0.3	3.097
8	10^{-3}	0.3	3.021
12	10^{-3}	0.3	2.993
16	10^{-3}	0.3	2.979
20	10^{-3}	0.3	2.973
24	10^{-3}	0.3	2.971

Tied, positive embedding decay

E	LR	WD	CE
1	10^{-3}	0.3	3.381
2	10^{-3}	0.3	3.217
4	10^{-3}	0.3	3.102
8	10^{-3}	0.3	3.020
12	10^{-3}	0.3	2.986
16	10^{-3}	0.3	2.967
20	10^{-3}	0.3	2.956
24	10^{-3}	0.3	2.950
28	10^{-3}	0.3	2.947
32	10^{-3}	0.3	2.946
36	10^{-3}	0.3	2.946

Table 14: Hyperparameters for the repetition-tuned recipe across all model sizes and batch sizes, in Figure 4 (§4), on a 1B-token pool. The same checkpoints were used for the validation loss and the Hellaswag BPB. Original-recipe configurations are in Table 7.

1.5B

B	LR	WD	E	CE
64	10^{-3}	0.3	32	2.946
128	10^{-3}	0.3	20	2.979
256	10^{-3}	1	40	3.019
512	10^{-3}	1	28	3.030

474M

B	LR	WD	E	CE
64	2×10^{-3}	0.1	56	3.064
128	2×10^{-3}	0.3	80	3.060
256	2×10^{-3}	0.3	64	3.064
512	2×10^{-3}	0.3	40	3.105

153M

B	LR	WD	E	CE
64	2×10^{-3}	0.1	224	3.293
128	2×10^{-3}	0.1	240	3.285
256	2×10^{-3}	0.1	208	3.298
512	2×10^{-3}	0.3	256	3.322

Table 15: Hyperparameters for every repetition-tuned point in the data-pool-size comparison of Figure 5 (§4). D_{u} is the unique-data pool size in billions of tokens.

1.5B

D_{u}	B	LR	WD	E	CE
0.333	64	10^{-3}	1	28	3.255
1	64	10^{-3}	0.3	32	2.946
3	128	10^{-3}	0.3	80	2.767

474M

D_{u}	B	LR	WD	E	CE
0.333	64	2×10^{-3}	0.3	32	3.281
1	128	2×10^{-3}	0.3	80	3.060
3	256	2×10^{-3}	0.1	112	2.909

153M

D_{u}	B	LR	WD	E	CE
0.333	64	2×10^{-3}	0.3	144	3.452
1	128	2×10^{-3}	0.1	240	3.285
3	256	2×10^{-3}	0.1	640	3.196

F Parallelism and Step-Time Estimation

This appendix summarizes the H100 performance model used in §D.2. We compare (1) a conventional **thoughput-first** policy that prioritizes tokens per GPU-second with (2) a **latency-first** policy that minimizes optimizer-step time. The estimates are projections rather than measurements of every configuration.

Setup and policies. We model dense decoder-only Transformers at sequence length $S = 4{,}096$, with no context parallelism. Let d, t, p, m, and a denote the data-parallel, tensor-parallel, pipeline-parallel, microbatch, and gradient-accumulation degrees. Then

$$B = dma, \qquad M = tp, \qquad G = dtp. \tag{6}$$

We assume FP8 matrix multiplications, BF16 parameters and communication, FlashAttention, activation checkpointing, sequence parallelism for $t > 1$, and distributed AdamW with sharded optimizer states. Parameters remain replicated across DP groups; we do not substitute ZeRO-3 or FSDP for TP or PP. Both policies fix the global batch at $B = 1{,}024$ sequences and gradient accumulation at $a = 1$.

The throughput-first policy uses a microbatch near the per-GPU throughput optimum and the smallest model-parallel degree that fits. The latency-first policy instead fixes $m = 1$, hence $d = 1{,}024$, and searches $t \in \{1, 2, 4, 8\}$ and $p \in \{1, 2, 4, 8\}$ subject to memory, divisibility, and topology constraints. TP groups are kept within one eight-GPU NVLink domain.

Configuration-dependent compute and pipeline time. For parameter count N, the forward-and-backward work per sequence is $F_{\text{seq}} \approx 6NS$. Rather than assigning one effective FLOP rate to all latency-first configurations, let $P_{\text{eff}}(N, t, p, m)$ denote the effective per-GPU rate for the particular configuration. This rate includes TP collectives and the loss of kernel efficiency as local matrices and pipeline stages become smaller. For one microbatch, the pipeline critical path is

$$T_{\text{FB}}(t, p, m) = p\, T_{\text{stage}}(N, t, p, m) \approx \frac{6NSm}{tP_{\text{eff}}(N, t, p, m)}. \tag{7}$$

The second form is exact only when stages are balanced. Writing the first form explicitly matters: PP does not ideally shorten the single-microbatch compute path, and can lengthen it when smaller stages lower utilization.

Forward activations and backward activation gradients cross each of the $p - 1$ pipeline boundaries. For hidden dimension h, their approximate BF16 volume is

$$V_{\text{PP}} \approx \frac{4(p-1)mSh}{t}, \qquad T_{\text{PP}} = \frac{V_{\text{PP}}}{\beta_{\text{PP}}} + 2(p-1)\alpha_{\text{PP}}. \tag{8}$$

This term is normally much smaller than DP gradient communication, but it is monotone in p and therefore prevents cost-free pipeline splitting.

Data-parallel communication and overlap. With BF16 communication, a distributed-optimizer step performs a gradient reduce-scatter and a parameter all-gather. Their per-rank volumes are

$$V_{\text{RS}} \approx V_{\text{AG}} \approx \frac{2N}{tp}\frac{d-1}{d}. \tag{9}$$

Let T_{RS} and T_{AG} include both bandwidth and collective latency. Gradient reduction can overlap only with the remaining backward schedule after a bucket becomes ready; parameter gathering can overlap only with optimizer work and the appropriate portion of the next forward pass. For a compact fitted model, we combine the phases into $T_{\text{DP}} = T_{\text{RS}} + T_{\text{AG}}$ and expose only the portion that cannot overlap the forward–backward critical path:

$$T_{\text{DP}}^{\text{exp}} = T_{\text{DP}} - \eta \min(T_{\text{FB}}, T_{\text{DP}}), \tag{10}$$

where η is a fitted overlap fraction. This interpolates between no overlap ($\eta = 0$) and the optimistic maximum model ($\eta = 1$) without allowing communication to overlap more compute than is available.

Our steady-state step estimate is

$$\widehat{T}_{\text{step}}(d, t, p, m) = T_{\text{FB}}(t, p, m) + T_{\text{DP}}^{\text{exp}}(d, t, p) + T_{\text{PP}}(t, p, m) + T_{\text{opt}}(t, p) + T_0, \tag{11}$$

where T_{opt} and T_0 cover optimizer and fixed framework costs. All terms that vary with t or p are kept explicit or calibrated separately; they are not absorbed into a configuration-independent residual.

Calibration and selection. We anchor H100 compute rates and feasible configurations to NVIDIA Megatron measurements (NVIDIA, 2026b;a). For 70B we use $P_{\text{eff}} = 0.65$ PFLOP/s/GPU at TP8, $\eta = 0.5$, and fit the effective DP bandwidth to the stated TP4/PP1 latency of 2.10 s. This gives $\beta_{\text{DP}} = 40$ GB/s. We apply stage-efficiency penalties of 0%, 3%, and 8% at PP2, PP4, and PP8, respectively. The fitted combined values of $T_{\text{PP}} + T_{\text{opt}} + T_0$ are 0.05, 0.09, 0.11, and 0.18 s for PP1, PP2, PP4, and PP8. These coefficients are held fixed while evaluating the candidates; they are not refit to favor a particular configuration.

The resulting 70B candidate latencies are

(t, p)	(4,1)	(4,2)	(4,4)	(4,8)	(8,1)	(8,2)	(8,4)	(8,8)
$\widehat{T}_{\text{step}}$ (s)	2.10	1.28	1.01	1.01	1.08	0.69	**0.56**	0.59

Thus the fitted latency objective selects TP8/PP4. PP8 provides no ideal single-microbatch compute reduction after PP4 reaches the compute floor, and its lower stage efficiency and additional PP overhead make it slower.