MOST SMALL-SCALE SOFTWARE PROGRESS IN PRETRAINING IS DATA PROGRESS

Anonymous authors
Paper under double-blind review

ABSTRACT

How much of the rapid progress in language models observed over the last few years has come from model versus data improvements? Current observational estimates from model releases have only provided measurements of joint improvements from both components. We conduct a novel investigation into the improvements in language model compute efficiency for pretraining specifically from 2019 to 2025, by experimentally decomposing these improvements into algorithmic progress in the training procedure and data progress in the pretraining corpora. We do so by retraining year-representative model training recipes (GPT-2 to OLMo 2) and corpora (OpenWebText to Ultra-FineWeb) at 10^{17}–10^{19} FLOPs. We evaluate these models on a common downstream OLMES10 benchmark suite and compare the observed compute multipliers. From GPT-2 trained on OpenWebText at 10^{19} FLOPs, upgrading the corpus to Ultra-FineWeb yields a multiplier of $\times$12.0, upgrading the model to OLMo 2 yields a multiplier of $\times$5.0, and upgrading both $\times$21.1. A Shapley decomposition attributes 64% of the joint gain to data progress. We conclude that from 2019 to 2025, at the scales we study, most software progress in pretraining is accounted for by data progress.

1 INTRODUCTION

Language models have improved rapidly in the past few years, and have also required far less compute to train to a given quality. From more than two hundred published models, Ho et al. (2024) estimate that the compute needed to reach a fixed perplexity halved roughly every eight months between 2012 and 2023, faster than hardware improved. This *software progress*, the reduction in training compute needed to reach a given capability as distinct from growth in the compute itself (Ho, 2026), has two sources in pretraining. *Algorithmic* or *model* progress improves the training procedure that turns compute into a model, namely its architecture, optimizer, initialization, learning-rate schedule and hyperparameters, which together we call a recipe. *Data progress* improves the corpus the model learns from, through better sources, quality filtering and deduplication, among other data engineering methods.

How much of software progress is algorithmic, and how much is data? The answer matters for forecasts of AI capability and for what will bottleneck progress next, because algorithmic progress can plausibly compound with research effort, whereas data progress may draw on a finite supply of data and on how well it can be curated.

Limitations of prior work. Three lines of work each measure part of the answer, but none measures both components on a common scale. Observational estimates measure only the joint effect. Ho et al. (2024) infer progress from published models, each of which changes its recipe, corpus, tokenizer, context length and scale at once. They note that their estimates "pertain to both algorithmic improvements and data quality improvements, the relative contributions of which could be a subject of future research," and Ho (2026) has since suggested that "most software progress might actually be due to data quality improvements." Controlled model ablations measure algorithmic progress only. Gundlach et al. (2025) ablate architecture and optimizer components on a single fixed corpus and measure quality as loss on that corpus, which cannot rank models trained on different data. Controlled data ablations measure data progress only. DataDecide (Magnusson et al., 2025) trains 25 corpora across model sizes and seeds under one fixed recipe, to predict which corpus is best at

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.

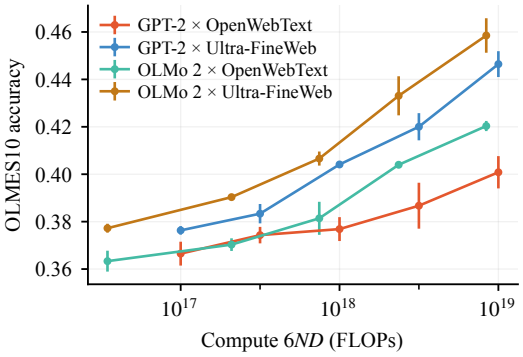


Figure 1: OLMES10 accuracy against training compute from 2019 (GPT-2 on OpenWebText) to 2025 (OLMo 2 on Ultra-FineWeb). Points are means over three training seeds; error bars are ± 1 seed standard deviation.

Table 1: Compute multipliers implied by Figure 1. Brackets are 68% paired-seed intervals over the 27 resamples of the three training seeds.

Upgrade	Multiplier at 10^{19}
Corpus upgrade, recipe fixed at GPT-2	
OpenWebText → Ultra-FineWeb	×12.0 [10.4, 13.7]
Recipe upgrade, corpus fixed at OpenWebText	
GPT-2 → OLMo 2	×5.0 [4.1, 5.9]
Both upgrades	×21.1 [16.4, 26.7]

larger scale rather than to weigh data against algorithms. The relative size of algorithmic and data progress over the last few years therefore remains an open question.

Our contribution. We measure both components on a common scale with controlled experiments that recombine historical recipes and corpora. For each year from 2019 to 2025 we take a representative published recipe and a representative released corpus, from GPT-2 and OpenWebText to OLMo 2 and Ultra-FineWeb.

We train each configuration with a shared tokenizer, context length and batch size, at up to five compute budgets from 10^{17} to 10^{19} FLOPs[1] and at each recipe's compute-optimal model size. We measure quality downstream on the ten-task OLMES10 suite (Gu et al., 2025), and express each change as a *compute multiplier*, the factor by which it reduces the training compute needed to match a reference model's accuracy. Upgrading the recipe with the corpus held fixed thus measures algorithmic progress, and upgrading the corpus with the recipe held fixed measures data progress.

We run three experiments. (1) A 2×2 design crosses the 2019 and 2025 recipes with the 2019 and 2025 corpora, so that each upgrade is measured alone and in combination. (2) We also look at two sweeps that trace progress year by year. The recipe sweep trains every year's recipe on FineWeb-Edu (2024), and the corpus sweep trains every year's corpus under OLMo 2 (2025). (3) Finally we consider a full grid of all 49 recipe–corpus pairs at one budget that tests whether the two components interact.

We summarize our key observations below, with brackets giving 68% paired-seed intervals over training seeds.

1. Most software progress is data progress. Based on our first experiment, from GPT-2 on OpenWebText at 10^{19} FLOPs, upgrading the corpus to Ultra-FineWeb yields a multiplier of ×12.0 [10.4, 13.7], upgrading the recipe to OLMo 2 yields ×5.0 [4.1, 5.9], and upgrading both yields ×21.1 [16.4, 26.7] (Table 1). The data multiplier is 2.4 [2.3, 2.5] times the recipe one, and a Shapley decomposition attributes 64% [63, 67] of the joint gain in log-compute to data.

2. Data progress is also more important in the year-by-year sweeps. Our second experiment trains every year's recipe, from GPT-2 (2019) to OLMo 2 (2025), on FineWeb-Edu, and every year's corpus, from OpenWebText (2019) to Ultra-FineWeb (2025), under OLMo 2. Upgrading the corpus from OpenWebText to Ultra-FineWeb under OLMo 2 yields ×6.2 [5.2, 6.9], against ×2.3 [2.0, 2.7] for upgrading the recipe from GPT-2 to OLMo 2 on FineWeb-Edu (Tables 4 and 5).

[1]Throughout the paper, compute is training compute, $C = 6ND$ FLOPs, with N the number of non-embedding parameters, the attention and MLP weight matrices (Kaplan et al., 2020), counted the same way for every recipe (Appendix D.4), and D the number of training tokens.

To our knowledge, this is the first controlled decomposition of software progress in pretraining into its algorithmic and data components. It supports the conjecture of Ho (2026). At the scales we study, most software progress in pretraining can be explained by data progress.

2 Related work

Measuring software progress. Beyond the three lines of work discussed in Section 1, compute-equivalent measurement began in vision. Hernandez & Brown (2020) found that the compute needed to reach AlexNet-level ImageNet accuracy fell $44\times$ over seven years, and Erdil & Besiroglu (2022) fitted augmented scaling laws to published models.

Algorithmic improvements in language models. Since the Transformer (Vaswani et al., 2017), dense language-model recipes have accumulated many changes, such as pre-layer normalization (Xiong et al., 2020), rotary position embeddings (Su et al., 2024), RMSNorm (Zhang & Sennrich, 2019), gated MLPs such as SwiGLU (Shazeer, 2020), grouped-query attention (Ainslie et al., 2023) and query–key normalization (Henry et al., 2020; Wortsman et al., 2024), with AdamW (Loshchilov & Hutter, 2019) and cosine learning-rate decay (Loshchilov & Hutter, 2017) as the standard optimizer and schedule. Newer optimizers such as Muon are reported to be about twice as compute-efficient as AdamW (Liu et al., 2025). However, controlled re-evaluations of the main 2012–2023 innovations find modest gains. In small-scale ablations these innovations account for less than $\times 10$ in compute, and most of the remaining gain comes from innovations whose benefit grows with scale (Gundlach et al., 2025).

The value of pretraining data. A sequence of open corpora has shown that curation can substantially improve models trained at fixed compute, through heuristic filtering and deduplication (Raffel et al., 2020; Penedo et al., 2023), source diversity (Gao et al., 2020), and learned quality classifiers (Penedo et al., 2024; Li et al., 2024; Wang et al., 2025). Other work has studied how quality, domain and age shape pretraining (Longpre et al., 2024), how data pruning and quality interventions change scaling behavior (Sorscher et al., 2022; Ardalani et al., 2026), and how repetition limits the value of small corpora (Muennighoff et al., 2023). Mayilvahanan et al. (2025) also vary the architecture and find that the data, far more than the architecture, tokenizer or optimization hyperparameters, determines how pretraining loss translates into loss on downstream tasks.

3 Experimental design

Recipes and corpora. For each year from 2019 to 2025 we select one documented, publicly implementable dense-transformer recipe and one released English pretraining corpus (Tables 2 and 3). The recipes accumulate the changes that define modern practice, from rotary positions, RMSNorm and SwiGLU to QK-norm and revised initialization. The corpora trace the progression from link-based scraping through heuristic filtering and deduplication to learned quality classifiers. Each recipe is a controlled reimplementation of its published architecture, optimizer and schedule within our shared framework (Appendix A). Experiments with further recipes and corpora are reported in Section 5 and Appendix E.

Controlled training. We pretrain the recipes from scratch on the corpora at compute budgets ranging from 10^{17} to 10^{19} FLOPs, with three independent seeds for every multi-budget curve. Every run is placed at its actual compute. Every run uses the same GPT-2 BPE tokenizer (50,257 tokens), 2,048-token context and 262,144-token batches, so that the measured gains are those of the recipe and the corpus alone. At each budget we train an isoFLOP ladder of model sizes for each recipe (Hoffmann et al., 2022) and use its compute-optimal size, the one with the lowest loss on held-out text from the training corpus (its native validation loss). We observe that this optimum is relatively constant across corpora under a fixed recipe, so most corpora reuse the size selected on FineWeb-Edu (Appendix D.3).

[2]OLM-CC's release reports only 15.6M documents; the 14.9M of them in our largest training bin contain 6.9B GPT-2 tokens, so the full corpus has about 7.2B.

Table 2: **Model recipes.** Each year's recipe, as we implement it (Appendix A), and its main changes from the previous one. All seven are dense decoder-only Transformers trained with AdamW and cosine learning-rate decay.

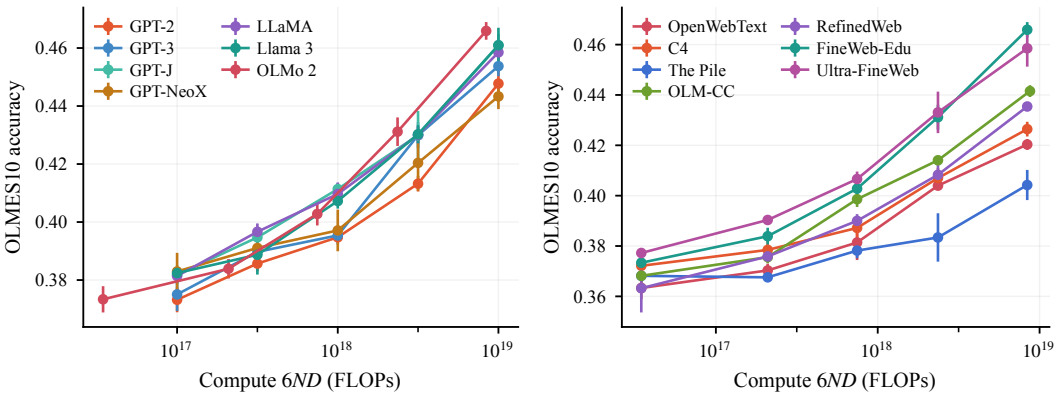
Year	Recipe	Main changes from the previous recipe
2019	GPT-2 (Radford et al., 2019)	pre-LayerNorm, GELU, learned positions, dropout, tied input and output embeddings
2020	GPT-3 (Brown et al., 2020)	no dropout, weight decay 0.1, Adam β_2=0.95
2021	GPT-J (Wang & Komatsuzaki, 2021)	rotary position embeddings
2022	GPT-NeoX (Black et al., 2022)	parallel attention and MLP, rotary on 25% of each head, untied head, weight decay 0.01
2023	LLaMA (Touvron et al., 2023)	RMSNorm, SwiGLU MLP, no bias terms
2024	Llama 3 (Grattafiori et al., 2024)	grouped-query attention, rotary base 5×10^5
2025	OLMo 2 (OLMo Team et al., 2025)	QK-norm, normalization of sublayer outputs, truncated-normal initialization

Table 3: **Data corpora.** Each year's corpus, how it was curated, and its approximate size in tokens.

Year	Corpus	Curation	Size
2019	OpenWebText (Gokaslan & Cohen, 2019)	web pages linked from Reddit, deduplicated; open WebText reproduction	9B
2020	C4 (Raffel et al., 2020)	Common Crawl; language, bad-word and boilerplate filters; deduplicated	156B
2021	The Pile (Gao et al., 2020)	curated mixture of 22 sources, including papers, code and books	300B
2022	OLM-CC (Thrush et al., 2022)	2022 Common Crawl snapshot; BigScience/OSCAR filters; deduplicated	7.2B[2]
2023	RefinedWeb (Penedo et al., 2023)	Common Crawl; strict heuristic filters; exact and fuzzy deduplication	600B
2024	FineWeb-Edu (Penedo et al., 2024)	FineWeb filtered by an educational-quality classifier	1.3T
2025	Ultra-FineWeb (Wang et al., 2025)	FineWeb re-filtered by a classifier trained on verified seed data	1T

Hyperparameters. The end capabilities of our training runs are highly dependent on hyperparameters. Obviously, we are limited in our ability to extensively sweep over all possible choices of hyperparameters, but we try to control for this as much as possible, and we consider peak learning rate as the main hyperparameter of significance. For the learning rate, we start from a prior for each recipe, based on its published setting or on earlier small-model sweeps, and recalibrate it to our batch size (Porian et al., 2024; Bi et al., 2024) and context length. We then sweep around it in small models and fit a power law for the optimal rate, which we extrapolate to the larger models and budgets, with the exception of OLMo 2, which instead adapts the published DataDecide settings (Magnusson et al., 2025). We then sweep again around these extrapolated rates at selected model sizes and budgets and adopt a better rate wherever three seeds confirm it. All further main results use these selected rates (Appendix C).

Downstream evaluation. Each selected checkpoint is scored on our primary endpoint, OLMES10, the macro-average of the ten OLMES tasks ARC-Easy, ARC-Challenge, BoolQ, CommonsenseQA, HellaSwag, MMLU, OpenBookQA, PIQA, SocialIQA and WinoGrande (Gu et al., 2025), on the full evaluation split of every task. Unlike loss on each corpus's own held-out text, a downstream benchmark scores every model on the same questions, so it can compare models trained on different corpora. We use OLMES10 because it is a documented standard that fixes prompt formats, in-context examples and answer normalization, and scores small models in a cloze formulation they can answer (Gu et al., 2025). Our choice is also supported by DataDecide, the most extensive open suite of controlled pretraining-data experiments, which uses the same ten tasks because they suit models of our scale (Magnusson et al., 2025). It is important to note that downstream accuracy is noisier than loss, so model size and learning rate are always selected on native validation loss, which is measured much more precisely and stably.



(a) Every year's recipe, trained on FineWeb-Edu

(b) OLMo 2, trained on every year's corpus

Figure 2: **OLMES10 accuracy against compute, for every year's recipe on FineWeb-Edu (a) and for OLMo 2 on every year's corpus (b).** Each recipe–corpus pair is evaluated at its compute-optimal model size. Every point is the mean of three independently seeded training runs; error bars are ± 1 seed SD.

Compute multipliers. Let $q_k(C)$ denote the downstream accuracy that configuration k, a recipe trained on a corpus, reaches with training compute C at its selected model size. For a reference configuration r evaluated at budget C, the compute multiplier of a candidate k is

$$M_k(C) \;=\; \frac{C}{C_k}, \qquad C_k \;=\; \min\{\, C' : q_k(C') \geq q_r(C) \,\}, \tag{1}$$

the factor by which the candidate reduces the compute needed to match the reference's accuracy. We evaluate q_k by piecewise-linear interpolation in $\log C$ between the measured budgets, extending the nearest segment for crossings outside the measured range (marked ‡).

4 Results

4.1 Most software progress is data progress

We first train all four combinations of the 2019 and 2025 recipes (GPT-2 and OLMo 2) and corpora (OpenWebText and Ultra-FineWeb) at all five compute budgets, with three seeds each. Starting from the 2019 stack, GPT-2 trained on OpenWebText, we consider three ablations, which upgrade only the recipe to OLMo 2, only the corpus to Ultra-FineWeb, or both (Figure 1).

At 10^{19} FLOPs, relative to the accuracy of the 2019 stack (Table 1), we find the following.

- The corpus upgrade to Ultra-FineWeb attains $\times$12.0 [10.4, 13.7].
- The recipe upgrade to OLMo 2 attains $\times$5.0 [4.1, 5.9].
- Both upgrades together attain $\times$21.1 [16.4, 26.7].
- The corpus upgrade's multiplier is 2.4 [2.3, 2.5] times the recipe upgrade's.
- A Shapley decomposition of the joint gain in log-compute (Appendix B.4) attributes 64% [63, 67] of it to data progress and the remaining 36% to algorithmic progress.
- At 3.16×10^{18} FLOPs, the corpus and recipe upgrades attain $\times$8.3 [6.0, 16.6] and $\times$3.2 [2.8, 3.5].

4.2 Recipe and corpus sweeps

We then train all seven recipes of Table 2 on FineWeb-Edu (2024), and also train all seven corpora of Table 3 under OLMo 2 (2025), each at all five budgets with three seeds. Multipliers are relative to GPT-2 on FineWeb-Edu and to OpenWebText under OLMo 2 (Figure 2; Tables 4 and 5).

Compute efficiency generally increases over time on both axes, but far faster and with greater magnitude for corpora. At 10^{19} FLOPs the corpus upgrade from OpenWebText to Ultra-FineWeb attains $\times$6.2 [5.2, 6.9], against $\times$2.3 [2.0, 2.7] for the recipe upgrade from GPT-2 to OLMo 2.

Table 4: **Compute multipliers of the model recipes.** Each year's recipe trained on FineWeb-Edu, relative to GPT-2.

Year	Recipe	@3.16×10^{18}	@10^{19}
2019	GPT-2	1.00 (ref)	1.00 (ref)
2020	GPT-3	1.74 [1.72, 1.77]	1.34 [1.27, 1.40]
2021	GPT-J	2.82 [2.63, 2.98]	1.63 [1.40, 1.98]
2022	GPT-NeoX	1.43 [1.03, 1.70]	0.80‡ [0.58, 0.95]
2023	LLaMA	2.59 [2.36, 2.81]	1.55 [1.35, 1.77]
2024	Llama 3	2.34 [2.09, 2.62]	1.64 [1.57, 1.73]
2025	OLMo 2	2.78 [2.56, 3.01]	2.31 [2.02, 2.71]

Table 5: **Compute multipliers of the data corpora.** Each year's corpus trained with OLMo 2, relative to OpenWebText.

Year	Corpus	@3.16×10^{18}	@10^{19}
2019	OpenWebText	1.00 (ref)	1.00 (ref)
2020	C4	1.19 [1.07, 1.31]	1.49 [1.35, 1.65]
2021	The Pile	0.29 [0.21, 0.32]	0.37‡ [0.14, 0.56]
2022	OLM-CC	2.12 [1.89, 2.41]	2.64 [2.50, 2.77]
2023	RefinedWeb	1.31 [1.15, 1.42]	2.03 [1.90, 2.14]
2024	FineWeb-Edu	3.01 [2.87, 3.15]	5.52 [5.27, 5.78]
2025	Ultra-FineWeb	3.88 [3.35, 4.38]	6.20 [5.21, 6.91]

Brackets are 68% paired-seed intervals; ‡ marks a crossing outside the measured compute range.

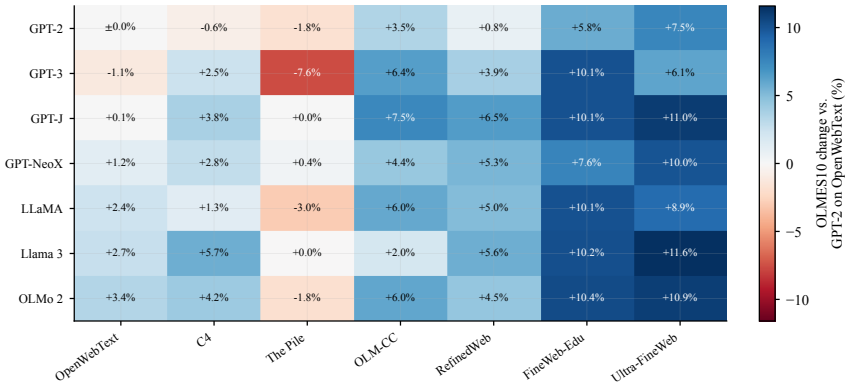


Figure 3: **Improvement of each recipe–corpus pair over the 2019 stack, at 3.16×10^{18} FLOPs.** Percentage change in OLMES10 accuracy relative to GPT-2 trained on OpenWebText, for all 49 combinations of the seven recipes (rows) and seven corpora (columns).

We observe two main anomalies. Firstly, GPT-NeoX performs worse than GPT-2 at 10^{19} FLOPs on the downstream evaluation, although it is ahead of GPT-2 from 10^{17} to 3.16×10^{18} FLOPs and also performs better on native validation loss at 10^{19} FLOPs ($\times 1.2$). We thereby ascribe this anomalous result to the evaluation noise between independent seeds.

The Pile also performs worse than OpenWebText, which makes sense because its main advance was diversity rather than filtering. The Pile features a curated mixture of 22 sources including PubMed and arXiv papers, GitHub code, legal opinions and patents, much of which may transfer little to the English multiple-choice questions of OLMES10. Because the Pile is far larger, we believe it will eventually overtake the small OpenWebText at larger scale.

4.3 Recipe and corpus gains are independent

We also train all 49 combinations of the seven recipes and seven corpora of Tables 2 and 3 at 3.16×10^{18} FLOPs, with one model size per recipe and mostly single-seed runs, to test whether recipe and corpus effects on accuracy interact. Figure 3 shows how much better each pair does than the 2019 stack (precise OLMES10 accuracies in Appendix Table 18).

The gains from recipes and corpora are largely independent. Realizing the gain of a newer recipe does not require a particular corpus, or the reverse. An additive model with one effect per recipe and one per corpus, $q_{ij} = \mu + \alpha_i + \beta_j + \epsilon_{ij}$ for recipe i and corpus j, explains 88.8% of the variance in accuracy across the 49 cells. Corpus effects account for 80.5% and recipe effects for 8.3%, leaving 11.2% for interactions and noise. The observed accuracies are all close to the additive model's predictions (Figure 4); the two largest deviations, 1.6 points each, are GPT-3 on the Pile, which was unstable in training, and Llama 3 on OLM-CC.

Despite this additivity in accuracy, the multipliers of the recipe and corpus upgrades do not multiply, because OLMES10 accuracy does not rise linearly in log-compute with a common slope.

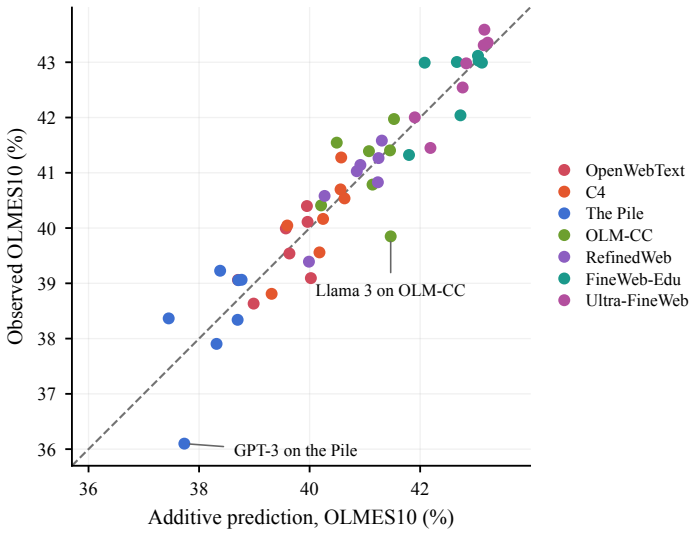


Figure 4: **Recipe and corpus effects are nearly additive.** Observed OLMES10 accuracy of each of the 49 recipe–corpus pairs at 3.16×10^{18} FLOPs against the prediction of the additive model, colored by corpus as in Figure 2(b).

4.4 Controlled software progress is much slower than observed progress

Annualizing the rate of improvement in data and algorithmic progress from 2019 to 2025, spread evenly over the six year-steps of the 2×2 design, we observe that data progress amounts to $\times 1.51$ [1.48, 1.55] per year and algorithmic progress to $\times 1.31$ [1.27, 1.34]. These are compute-halving times of 20 [19, 21] and 31 [28, 35] months (16 [15, 18] jointly). All are far slower than the roughly eight-month halving time that Ho et al. (2024) estimate from published models, whose window also spans the pre-2019 transition from LSTMs to Transformers.

This gap also indicates some limitations of what our experiments can measure. All our recipes are dense decoder-only Transformers, so we do not measure sparse mixture-of-experts architectures (Fedus et al., 2022; Jiang et al., 2024), which let a model use far more parameters per unit of training compute, or state-space models (Gu & Dao, 2024). The tokenizer, context length and batch size are held fixed, although each has also improved over the period. Our compute budgets also end at 10^{19} FLOPs, so algorithmic gains that emerge only at larger scale (Gundlach et al., 2025) lie outside our measurements. On the data side, our corpora are filtered web text, without synthetic pretraining data (Gunasekar et al., 2023). Our estimates therefore describe the software progress visible in small dense models trained on web text corpora, while not necessarily accounting for model progress that depends on scale, or data progress that depends on synthetic data generation methods. Systems work such as FlashAttention (Dao et al., 2022), which raises hardware throughput rather than reducing the FLOPs a model needs, is also neglected by our framing of LLM progress as compute-efficiency gains.

5 Extensions to other recipes and corpora

Beyond the year representatives, we trained a newer recipe more reflective of 2026 LLM pretraining methods (Delphi, 2026), a second fixed recipe for the corpus comparison (OLMo 3; Team Olmo, 2025) and more selective corpora (DCLM-Baseline, Li et al., 2024, and DCLM-QC7/FW3, Magnusson et al., 2025; Appendix E).

Latest recipes and more selective corpora. DCLM-QC7/FW3 (Magnusson et al., 2025), which keeps the top 7% of DCLM by quality classifier and requires a FineWeb-Edu score of at least 3, is more selective than any year representative. We train OLMo 3 on every corpus with five-budget, three-seed curves. Under it, DCLM-QC7/FW3 attains $\times 16.6$ over OpenWebText, against $\times 9.5$ for FineWeb-Edu and $\times 5.9$ for DCLM-Baseline (Appendix Figure 13).

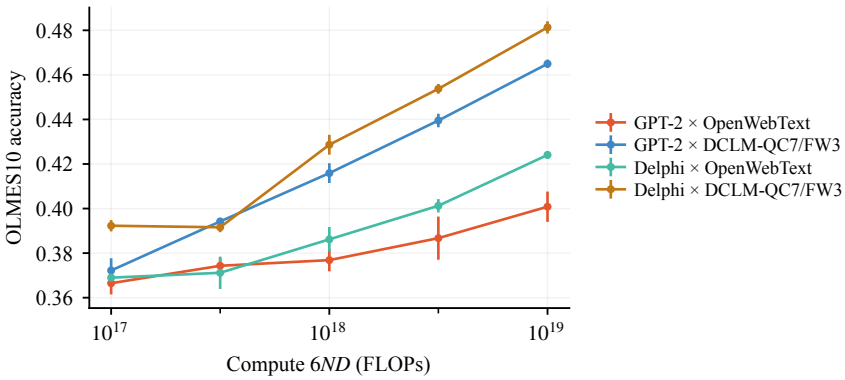


Figure 5: OLMES10 accuracy against training compute in the extended 2×2 design, which trains GPT-2 and Delphi on OpenWebText and on DCLM-QC7/FW3. Points are means over three training seeds; error bars are ± 1 seed SD.

Table 6: Compute multipliers implied by Figure 5, from GPT-2 on OpenWebText at 10^{19} FLOPs. Brackets are 68% paired-seed intervals over the 27 resamples of the three training seeds.

Upgrade	Multiplier at 10^{19}
Corpus upgrade, recipe fixed at GPT-2	
OpenWebText $\rightarrow$ DCLM-QC7/FW3	$\times$22.3 [20.1, 25.3]
Recipe upgrade, corpus fixed at OpenWebText	
GPT-2 $\rightarrow$ Delphi	$\times$3.3 [3.0, 4.0]
Both upgrades	$\times$23.7 [21.7, 25.9]

Delphi, our local adaptation of a 2026 configuration that pairs a QK-norm and SwiGLU backbone with the AdamH optimizer (Marin Community, 2026), is on par with OLMo 2 on FineWeb-Edu, attaining $\times$2.3 [2.2, 2.5] over GPT-2 at 10^{19} FLOPs against OLMo 2's $\times$2.3.

Extended 2×2 design. We extend the 2×2 design with the newest recipe and the most selective corpus we trained, Delphi and DCLM-QC7/FW3, with five-budget, three-seed curves in every cell (Figure 5 and Table 6).

From GPT-2 on OpenWebText at 10^{19} FLOPs, upgrading the corpus to DCLM-QC7/FW3 attains $\times$22.3 [20.1, 25.3], upgrading the recipe to Delphi $\times$3.3 [3.0, 4.0], and both $\times$23.7 [21.7, 25.9]; the Shapley decomposition gives data progress 80% [78, 81] of the joint gain, against 64% in the 2019–2025 design.

This further supports our conclusion that at the scales we investigate and for the particular downstream capability we are evaluating, data engineering work drives a much greater proportion of compute efficiency gains in pretraining.

6 Discussion

Algorithmic progress. A naive reading of our results is that pretraining progress since 2019 was mostly better data engineering and that algorithmic model research mattered much less. This is, however, an overly simplistic view of algorithmic improvements.

A compute multiplier credits a recipe only with efficiency at a fixed scale, whereas much algorithmic research has aimed at making larger amounts of compute usable at all. As parameters, context length, run duration and clusters grow, gradients explode or vanish and memory and bandwidth run out. Mixture-of-experts layers (Fedus et al., 2022), stability changes such as QK-norm and reordered normalization (Wortsman et al., 2024; OLMo Team et al., 2025) and systems work such as FlashAttention (Dao et al., 2022) fall into this category.

The recipe step that introduced grouped-query attention (Ainslie et al., 2023) adds little here. At 10^{19} FLOPs the recipe multiplier over GPT-2 rises only from $\times 1.55$ for LLaMA to $\times 1.64$ for Llama 3 (Table 4), as Gundlach et al. (2025) also find for RMSNorm. OLMo 2's step, which adds QK-norm and reordered normalization together with a new initialization, raises it to $\times 2.31$.

Much of the value of algorithmic progress may thus lie in enlarging the compute that can be used productively, which a fixed-scale multiplier does not credit, while data progress has supplied most of the measured gain in what each unit of that compute buys.

Why data progress may matter less at scale. The data improvements we investigated here might matter less for larger models. Small models (like the ones we trained) see significant gains from data quality improvements, because they don't have that much capacity relative to the amount of data that exists. So what matters most is the quality and information density of what they are trained on. Larger models may instead benefit from as much data as possible, even at a lower average quality, because aggressive filtering forces many epochs over a smaller corpus, which eventually costs performance (Muennighoff et al., 2023), especially for models trained on far more tokens than is compute-optimal to reduce inference cost. In the high-compute, data-scarce regime that is most relevant to the frontier models, Mohri et al. (2026) find a bitter lesson (Sutton, 2019) for data filtering, that with enough compute, the best filter is no filter.

Data progress is partly algorithmic. The most effective corpora select text with trained classifiers. FineWeb-Edu's classifier predicts educational value, and Ultra-FineWeb's predicts which data will help a model learn. Much of the data progress we measure is therefore algorithmic methods for data engineering, and its largest gain over the best earlier corpus ($\times 2.1$ at 10^{19} FLOPs, from OLM-CC to FineWeb-Edu) came when hand-written filters gave way to such learned selection (Table 5). Even though our study classifies these improvements as data improvements, this can also be considered as gains driven by algorithmic improvements (just not in model architecture but in data engineering).

Sensitivity to end evaluations. For this study, we are primarily investigating compute multipliers with respect to the OLMES10 evaluation (which combines 10 different relatively easy Q&A task types). The compute multipliers for each recipe and corpus might look different if we were looking at other benchmarks (say, coding-, math- or problem-solving-specific ones), which would probably reward very different methods of data engineering.

The data wall and synthetic data. Most of the corpora we study are curations of the same Common Crawl rather than new sources of data, so their gains draw on a finite stock. Synthetic data is widely used to train frontier models, and we have not investigated whether it can expand a data corpus without hurting model performance. If the gains are limited, the main driver of pretraining progress will stall, because we're not generating more internet, and a fixed set of data can only be curated so much. This is an important avenue for future investigations.

REPRODUCIBILITY STATEMENT

Section 3 and Appendices A–D describe what is needed to reproduce our training runs and analyses. They specify every recipe and corpus with its pinned upstream revision, the shared training inputs, the learning-rate calibration, the selection of model sizes, the parameter and compute accounting, and the construction of the compute multipliers and their uncertainty.

The supplementary package contains the training code together with the configuration of every recipe, corpus and model ladder, and a script that retrains a single configuration and compares it with our recorded run. It also contains a spreadsheet of all 1,495 training runs, including unsuccessful and superseded ones, with the final evaluation and loss of each. These are the 1,397 runs of the study and 98 follow-up runs. For every checkpoint that enters the analyses, the package includes the per-item evaluation outcomes on both task suites. Finally, CPU-only analysis scripts regenerate every table, figure and number in the main text and the appendix from these records and check each one against the value printed in the paper.

AI USE STATEMENT

We used generative AI tools extensively, across most of the research process. Below we state, for each kind of task, whether AI was used and how.

Research question and hypotheses. The research question and hypothesis, the separation of recipe and corpus, the choice of downstream endpoint and the decision to hold the tokenizer, context length and batch size fixed were decided by the authors. AI assistants did not propose the hypothesis. They helped assess alternative explanations of the results.

Research methodology and experiments. The research methodology was decided by the authors. AI agents assisted in writing the code for the experiments.

Conceptual framework and mathematics. The conceptual framework and mathematics were determined by the authors. AI agents assisted in the calculations.

Implementation. Most of the software behind this work (the training and evaluation harness, the corpus packing tool, the orchestration of training runs, the analysis pipeline, the figure scripts and the analysis notebooks) was written by an AI coding assistant working under human direction and review. The supplementary code submission, which reproduces the main results from the released run records, was also organized by an AI coding assistant.

Data. We generated no synthetic data. All training corpora and benchmarks are existing public releases, and AI did not write or alter any training or evaluation example. AI-written code tokenized and packed the corpora and ran the evaluations. AI was not used for translation or for qualitative or thematic analysis.

Analysis, interpretation and writing. All reported measurements and numerical estimates are computed by training, evaluation and analysis code. AI assistants helped interpret results, search the literature and check references against their sources. AI assistants also helped in composing the final results into the paper and edited the text. Most of the writing in the paper was human generated.

Verification. The authors reviewed all AI-assisted work.

We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. GQA: Training generalized multi-query transformer models from multi-head checkpoints. In *Proceedings of EMNLP*, 2023. URL https://arxiv.org/abs/2305.13245.

Allen Institute for AI. jeopardy_mc. Hugging Face dataset allenai/jeopardy_mc, version 1.0.0; accessed September 18, 2026, 2025. URL https://huggingface.co/datasets/allenai/jeopardy_mc.

Newsha Ardalani, Feiyang Kang, Michael Kuchnik, Mostafa Elhoushi, Shubho Sengupta, Shang-Wen Li, and Carole-Jean Wu. How text quality interventions reshape neural scaling laws for LLMs: Empirical study. In *International Conference on Learning Representations (ICLR)*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/6544c0b53cc1ea51d0c7086b536830a8-Abstract-Conference.html.

Xiao Bi, Deli Chen, Guanting Chen, Shanhuang Chen, Damai Dai, Chengqi Deng, Honghui Ding, Kai Dong, Qiushi Du, Zhe Fu, et al. DeepSeek LLM: Scaling open-source language models with longtermism. *arXiv preprint arXiv:2401.02954*, 2024. URL https://arxiv.org/abs/2401.02954.

Sid Black, Stella Biderman, Eric Hallahan, Quentin Anthony, Leo Gao, Laurence Golding, Horace He, Connor Leahy, Kyle McDonell, Jason Phang, Michael Pieler, USVSN Sai Prashanth, Shivanshu Purohit, Laria Reynolds, Jonathan Tow, Ben Wang, and Samuel Weinbach. GPT-NeoX-20B: An open-source autoregressive language model. In *Proceedings of BigScience Episode #5 – Workshop on Challenges & Perspectives in Creating Large Language Models*, pp. 95–136. Association for Computational Linguistics, 2022. doi: 10.18653/v1/2022.bigscience-1.9. URL https://aclanthology.org/2022.bigscience-1.9/.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020. URL https://arxiv.org/abs/2005.14165.

Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. FlashAttention: Fast and memory-efficient exact attention with IO-awareness. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022. URL https://arxiv.org/abs/2205.14135.

Dheeru Dua, Yizhong Wang, Pradeep Dasigi, Gabriel Stanovsky, Sameer Singh, and Matt Gardner. DROP: A reading comprehension benchmark requiring discrete reasoning over paragraphs. In *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. 2368–2378, 2019. doi: 10.18653/v1/N19-1246. URL https://aclanthology.org/N19-1246/.

EleutherAI. GPT-J-6B official model configuration, 2021. URL https://huggingface.co/EleutherAI/gpt-j-6b/blob/47e169305d2e8376be1d31e765533382721b2cc1/config.json. Verified revision 47e169305d2e8376be1d31e765533382721b2cc1.

Ege Erdil and Tamay Besiroglu. Algorithmic progress in computer vision. *arXiv preprint arXiv:2212.05153*, 2022. URL https://arxiv.org/abs/2212.05153.

William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(120):1–39, 2022.

Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, Shawn Presser, and Connor Leahy. The Pile: An 800GB dataset of diverse text for language modeling. *arXiv preprint arXiv:2101.00027*, 2020. URL https://arxiv.org/abs/2101.00027.

Aaron Gokaslan and Vanya Cohen. OpenWebText corpus. http://Skylion007.github.io/OpenWebTextCorpus, 2019.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024. URL https://arxiv.org/abs/2407.21783v3.

Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. In *First Conference on Language Modeling (COLM)*, 2024. URL https://openreview.net/forum?id=tEYskw1VY2.

Yuling Gu, Oyvind Tafjord, Bailey Kuehl, Dany Haddad, Jesse Dodge, and Hannaneh Hajishirzi. OLMES: A standard for language model evaluations. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pp. 5020–5048. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.findings-naacl.282. URL https://aclanthology.org/2025.findings-naacl.282/.

Suriya Gunasekar, Yi Zhang, Jyoti Aneja, Caio César Teodoro Mendes, et al. Textbooks are all you need. *arXiv preprint arXiv:2306.11644*, 2023. URL https://arxiv.org/abs/2306.11644.

Hans Gundlach, Alex Fogelson, Jayson Lynch, Ana Trisovic, Jonathan Rosenfeld, Anmol Sandhu, and Neil Thompson. On the origin of algorithmic progress in AI. *arXiv preprint arXiv:2511.21622*, 2025. URL https://arxiv.org/abs/2511.21622.

Alex Henry, Prudhvi Raj Dachapally, Shubham Pawar, and Yuxuan Chen. Query-key normalization for Transformers. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, 2020. URL https://arxiv.org/abs/2010.04245.

Danny Hernandez and Tom B. Brown. Measuring the algorithmic efficiency of neural networks. *arXiv preprint arXiv:2005.04305*, 2020. URL https://arxiv.org/abs/2005.04305.

Anson Ho. The least understood driver of AI progress. Epoch AI, Gradient Updates, 2026. URL https://epochai.substack.com/p/the-least-understood-driver-of-ai. Published 26 February 2026.

Anson Ho, Tamay Besiroglu, Ege Erdil, David Owen, Robi Rahman, Zifan Carl Guo, David Atkinson, Neil Thompson, and Jaime Sevilla. Algorithmic progress in language models. *arXiv preprint arXiv:2403.05812v1*, 2024. URL https://arxiv.org/abs/2403.05812v1.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022. URL https://arxiv.org/abs/2203.15556.

Albert Q. Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, et al. Mixtral of experts. *arXiv preprint arXiv:2401.04088*, 2024. URL https://arxiv.org/abs/2401.04088.

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. URL https://arxiv.org/abs/2001.08361.

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, Kristina Toutanova, Llion Jones, Matthew Kelcey, Ming-Wei Chang, Andrew M. Dai, Jakob Uszkoreit, Quoc Le, and Slav Petrov. Natural questions: A benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:452–466, 2019. doi: 10.1162/tacl_a_00276. URL https://aclanthology.org/Q19-1026/.

Jeffrey Li, Alex Fang, Georgios Smyrnis, Maor Ivgi, Matt Jordan, Samir Gadre, Hritik Bansal, Etash Guha, Sedrick Keh, Kushal Arora, et al. DataComp-LM: In search of the next generation of training sets for language models. In *Advances in Neural Information Processing Systems (Datasets and Benchmarks Track)*, volume 37, pp. 14200–14282, 2024. URL https://arxiv.org/abs/2406.11794.

Jingyuan Liu, Jianlin Su, Xingcheng Yao, Zhejun Jiang, Guokun Lai, Yulun Du, Yidao Qin, Weixin Xu, Enzhe Lu, Junjie Yan, et al. Muon is scalable for LLM training. *arXiv preprint arXiv:2502.16982*, 2025. URL https://arxiv.org/abs/2502.16982.

Shayne Longpre, Gregory Yauney, Emily Reif, Katherine Lee, Adam Roberts, Barret Zoph, Denny Zhou, Jason Wei, Kevin Robinson, David Mimno, and Daphne Ippolito. A pretrainer's guide to training data: Measuring the effects of data age, domain coverage, quality, & toxicity. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, 2024. URL https://arxiv.org/abs/2305.13169.

Ilya Loshchilov and Frank Hutter. SGDR: Stochastic gradient descent with warm restarts. In *International Conference on Learning Representations (ICLR)*, 2017. URL https://arxiv.org/abs/1608.03983.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations (ICLR)*, 2019. URL https://arxiv.org/abs/1711.05101.

Ian Magnusson, Nguyen Tai, Ben Bogin, David Heineman, Jena D. Hwang, Luca Soldaini, Akshita Bhagia, Jiacheng Liu, Dirk Groeneveld, Oyvind Tafjord, Noah A. Smith, Pang Wei Koh, and Jesse Dodge. DataDecide: How to predict best pretraining data with small experiments. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 42487–42502. PMLR, 2025. URL https://proceedings.mlr.press/v267/magnusson25a.html.

Marin Community. Completed AdamH v6 Delphi scaling suite. Campaign source snapshot, 2026. URL https://github.com/marin-community/marin/blob/4c4e759ecf688d1fa4120d6ac45e52f7b0844435/experiments/exp1337_delphi_suite.py. Pinned commit 4c4e759ecf688d1fa4120d6ac45e52f7b0844435; year denotes campaign label.

Prasanna Mayilvahanan, Thaddäus Wiedemer, Sayak Mallick, Matthias Bethge, and Wieland Brendel. LLMs on the line: Data determines loss-to-loss scaling laws. In *International Conference on Machine Learning (ICML)*, 2025. URL https://arxiv.org/abs/2502.12120.

Meta AI. LLaMA reference implementation. `llama_v1` branch, `llama/model.py`, 2023. URL https://github.com/facebookresearch/llama/blob/llama_v1/llama/model.py.

Christopher Mohri, John Duchi, and Tatsunori Hashimoto. A bitter lesson for data filtering. *arXiv preprint arXiv:2605.19407*, 2026. URL https://arxiv.org/abs/2605.19407.

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. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023. URL https://arxiv.org/abs/2305.16264.

OLMo Team, Pete Walsh, Luca Soldaini, Dirk Groeneveld, Kyle Lo, Shane Arora, Akshita Bhagia, Yuling Gu, Shengyi Huang, Matt Jordan, et al. 2 OLMo 2 furious. *arXiv preprint arXiv:2501.00656*, 2025. URL https://arxiv.org/abs/2501.00656.

Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Ngoc Quan Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. The LAMBADA dataset: Word prediction requiring a broad discourse context. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1525–1534, 2016. doi: 10.18653/v1/P16-1144. URL https://aclanthology.org/P16-1144/.

Guilherme Penedo, Quentin Malartic, Daniel Hesslow, Ruxandra Cojocaru, Hamza Alobeidli, Alessandro Cappelli, Baptiste Pannier, Ebtesam Almazrouei, and Julien Launay. The RefinedWeb dataset for Falcon LLM: Outperforming curated corpora with web data only. In *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*, volume 36, 2023. URL https://arxiv.org/abs/2306.01116.

Guilherme Penedo, Hynek Kydlíček, Loubna Ben Allal, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro Von Werra, and Thomas Wolf. The FineWeb datasets: Decanting the web for the finest text data at scale. In *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*, 2024. URL https://arxiv.org/abs/2406.17557.

Tomer Porian, Mitchell Wortsman, Jenia Jitsev, Ludwig Schmidt, and Yair Carmon. Resolving discrepancies in compute-optimal scaling of language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 37, 2024. URL https://arxiv.org/abs/2406.19146.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. OpenAI technical report, 2018. URL https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf.

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

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020. URL https://arxiv.org/abs/1910.10683.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pp. 2383–2392, 2016. doi: 10.18653/v1/D16-1264. URL https://aclanthology.org/D16-1264/.

Siva Reddy, Danqi Chen, and Christopher D. Manning. CoQA: A conversational question answering challenge. *Transactions of the Association for Computational Linguistics*, 7:249–266, 2019. doi: 10.1162/tacl_a_00266. URL https://aclanthology.org/Q19-1016/.

Melissa Roemmele, Cosmin Adrian Bejan, and Andrew S. Gordon. Choice of plausible alternatives: An evaluation of commonsense causal reasoning. In *AAAI Spring Symposium on Logical Formalizations of Commonsense Reasoning*, 2011. URL https://roemmele.github.io/publications/AAAI-SPRING11A.PDF.

Noam Shazeer. GLU variants improve transformer. *arXiv preprint arXiv:2002.05202*, 2020. URL https://arxiv.org/abs/2002.05202.

Ben Sorscher, Robert Geirhos, Shashank Shekhar, Surya Ganguli, and Ari S. Morcos. Beyond neural scaling laws: Beating power law scaling via data pruning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022. URL https://arxiv.org/abs/2206.14486.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. RoFormer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024. doi: 10.1016/j.neucom.2023.127063. arXiv:2104.09864.

Richard Sutton. The bitter lesson. http://www.incompleteideas.net/IncIdeas/BitterLesson.html, 2019.

Team Olmo. Olmo 3. *arXiv preprint arXiv:2512.13961*, 2025. URL https://arxiv.org/abs/2512.13961v1.

Tristan Thrush, Helen Ngo, Nathan Lambert, and Douwe Kiela. Online language modelling data pipeline. https://github.com/huggingface/olm-datasets, 2022. Dataset: https://huggingface.co/datasets/olm/olm-CC-MAIN-2022-40-sampling-ratio-0.15894621295.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. LLaMA: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023. URL https://arxiv.org/abs/2302.13971.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017. URL https://arxiv.org/abs/1706.03762.

Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. SuperGLUE: A stickier benchmark for general-purpose language understanding systems. In *Advances in Neural Information Processing Systems*, volume 32, 2019. URL https://arxiv.org/abs/1905.00537.

Ben Wang and Aran Komatsuzaki. GPT-J-6B: A 6 billion parameter autoregressive language model. https://github.com/kingoflolz/mesh-transformer-jax, 2021.

Yudong Wang, Zixuan Fu, Jie Cai, Peijun Tang, Hongya Lyu, Yewei Fang, Zhi Zheng, Jie Zhou, Guoyang Zeng, Chaojun Xiao, Xu Han, and Zhiyuan Liu. Ultra-FineWeb: Efficient data filtering and verification for high-quality LLM training data. *arXiv preprint arXiv:2505.05427*, 2025. URL https://arxiv.org/abs/2505.05427.

Johannes Welbl, Nelson F. Liu, and Matt Gardner. Crowdsourcing multiple choice science questions. In *Proceedings of the 3rd Workshop on Noisy User-generated Text*, pp. 94–106, 2017. doi: 10.18653/v1/W17-4413. URL https://aclanthology.org/W17-4413/.

Mitchell Wortsman, Peter J. Liu, Lechao Xiao, Katie Everett, Alex Alemi, Ben Adlam, John D. Co-Reyes, Izzeddin Gur, Abhishek Kumar, Roman Novak, Jeffrey Pennington, Jascha Sohl-Dickstein, Kelvin Xu, Jaehoon Lee, Justin Gilmer, and Simon Kornblith. Small-scale proxies for large-scale Transformer training instabilities. In *International Conference on Learning Representations (ICLR)*, 2024. URL https://arxiv.org/abs/2309.14322.

Ruibin Xiong, Yunchang Yang, Di He, Kai Zheng, Shuxin Zheng, Chen Xing, Huishuai Zhang, Yanyan Lan, Liwei Wang, and Tie-Yan Liu. On layer normalization in the Transformer architecture. In *International Conference on Machine Learning (ICML)*, 2020. URL https://arxiv.org/abs/2002.04745.

Biao Zhang and Rico Sennrich. Root mean square layer normalization. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019. URL https://arxiv.org/abs/1910.07467.

A RECIPES AND CORPORA

Choice of representatives. For each year from 2019 to 2025 we chose a documented, publicly implementable dense-transformer recipe and a released English pretraining corpus. The choice is constrained by architecture, public availability and our shared training inputs, so the representatives stand for documented practice in each year. They are not necessarily the most popular or the most compute-efficient choices of that year. The additional recipes and corpora (Appendix E) extend this set. Unless stated otherwise, brackets throughout the appendix are 68% paired-seed intervals (Appendix B).

Table 7: Recipe configurations as trained and their pinned sources. The architecture column lists the main changes from the previous recipe. All recipes use AdamW unless stated (β denotes Adam's (β_1, β_2) and wd the weight decay). Learning rates are in Appendix C.

Recipe (year)	Architecture and regularization	Optimizer and schedule	Pinned source
GPT-2 (2019)	Pre-LayerNorm, GELU, learned positions; tied input and output embeddings, linear biases; dropout 0.1	$\beta = (0.9, 0.999)$, wd 0.01; cosine to 10%	GPT-2 paper; GPT-1 for weight decay and dropout (Radford et al., 2018); Adam's default β; GPT-3's 10% floor
GPT-3 (2020)	Same dense backbone, with dense attention in every layer (upstream alternates dense and sparse); dropout removed	$\beta = (0.9, 0.95)$, wd 0.1; cosine to 10%	arXiv:2005.14165v4, App. B
GPT-J (2021)	Full-head RoPE, sequential blocks and tied head kept from our GPT-3, all unlike upstream GPT-J	Carried from GPT-3 ($\beta_2 = 0.95$, wd 0.1; cosine to 10%)	Official GPT-J config `@47e16930` (EleutherAI, 2021)
GPT-NeoX (2022)	Parallel attention/MLP block; rotary on 25% of head dimensions; untied head; NeoX role-split "small init"	$\beta_2 = 0.95$, wd 0.01; 1% warmup, cosine to 10%	`EleutherAI/gpt-neox@71df4d50` (v1.0); arXiv:2204.06745
LLaMA (2023)	RMSNorm, SwiGLU, RoPE; untied head, no linear biases	$\beta_2 = 0.95$, wd 0.1; cosine to 10%	arXiv:2302.13971v1 §2.2–2.3; reference code for the head and biases (Meta AI, 2023)
Llama 3 (2024)	Grouped-query attention (rounded 1:4 KV/query target); common SwiGLU width; scaled RoPE (Llama-3 frequency transform), base 5×10^5	Carried from LLaMA ($\beta_2 = 0.95$, wd 0.1; cosine to 10%)	arXiv:2407.21783v3 §3.2; Llama 3.1 reference code for the RoPE transform
OLMo 2 (2025; ladder)	Reordered norm placement; QK-norm on the concatenated projections; SwiGLU MLP of width $4d$; truncated-normal init (0.02, cutoff 3)	$\beta_2 = 0.95$, wd 0.05; cosine to 1%	arXiv:2501.00656v3; `allenai/OLMo@55d4871d`
OLMo 3 (2025; ladder)	OLMo 2 backbone; z-loss 10^{-5}; sliding-window pattern, inert at context 2,048	SkipStep AdamW, $\beta = (0.9, 0.99)$, wd 0.1; warmup-stable-decay to zero	Team Olmo (2025), v1; `allenai/OLMo-core@002e0d79` (v2.5.0)
Delphi (2026)	Pre-RMSNorm, SwiGLU, QK-norm, scaled (Llama-3) RoPE; z-loss 10^{-7}	Local AdamH for matrices, Adam for the remaining parameters; wd 0; gradient clip 0.1; 10% warmup, linear decay over the final 20%	Marin Community (2026); `marin-community/marin@4c4e759e`; adapted to PyTorch

Table 7 gives each recipe's configuration as trained, the change it introduces relative to the previous recipe and the upstream source it follows, and Table 8 gives each corpus's source and pinned revision. Where the upstream source does not specify a value, we use a documented default. GPT-J and both LLaMA recipes carry over the GPT-2 initialization, and Llama 3 carries over LLaMA's Adam β values and gradient clipping. The 2021–2023 recipes use the RoPE paper's default rotary

Table 8: Corpora and their pinned revisions. Sizes are approximate token counts. The Pile is read through the official Pythia document index. DCLM-QC7/FW3 is released as OLMo-tokenized data; we decoded it into EOS-delimited segments and retokenized them with the GPT-2 tokenizer after checking that decoding and re-encoding agree. Elsewhere we use the short names of Tables 2 and 3.

Corpus (year)	Size	Source and selection	Pinned revision
OpenWebText (2019)	9B	Reddit-linked pages; open WebText reproduction with deduplication	`Skylion007/openwebtext @79d93d78`
C4 English (2020)	156B	Common Crawl; T5 language, bad-word and boilerplate filters; deduplication	`allenai/c4@1588ec45`, `en` split
The Pile (2021)	300B	Curated 22-source mixture; official Pythia document-index view	`EleutherAI/ pythia_pile_idxmaps @889f5859`
OLM-CC-2022 (2022)	15.6M docs; ≈7.2B GPT-2 tokens	CC-MAIN-2022-40 sample; OSCAR/BigScience filtering and deduplication	`olm/ olm-CC-MAIN-2022-40-... @0281194d`
Falcon RefinedWeb (2023)	600B	Common Crawl; heuristic filtering and exact and fuzzy deduplication	`tiiuae/falcon-refinedweb @c7358405`
FineWeb-Edu (2024)	1.3T; 350B sample	FineWeb filtered by educational-quality classifier	`HuggingFaceFW/fineweb-edu @fc9850df`, `sample-350BT`
Ultra-FineWeb-EN (2025)	1T	FineWeb re-filtered by fastText; seed pools selected by training-based verification	`openbmb/Ultra-FineWeb @7ddd4170`, `ultrafineweb_en` split
DCLM-Baseline (2024)	4T	Common Crawl; heuristic cleaning, Bloom-filter deduplication and fastText quality filtering	`mlfoundations/ dclm-baseline-1.0 @a3b142c1`
DCLM-QC7/FW3 (2025)	Released token stream	DataDecide: top-7% DCLM quality filter plus FineWeb-Edu score ≥ 3; decoded and re-tokenized	`allenai/ DataDecide-data-recipes @3baf34ba`, `v0_rep32_ ft7percentile_fw3` stream
Dolma 3 Mix (2025)	157B	Released mixture; raw-document view	`allenai/ dolma3_mix-150B-1025 @afa92bfb`

base frequency of 10,000 (Su et al., 2024). GPT-2's tied embeddings and linear biases follow its reference implementation.

Departures from the original systems. Each recipe is named after the model it follows, but it is our own reimplementation. Beyond the shared tokenizer, context length and batch size, the recipes depart from the original models as follows.

- **GPT-2, GPT-3, GPT-J and GPT-NeoX** use the exact GELU, and GPT-3 uses dense attention in every layer.
- **GPT-J** is a controlled variant. The original GPT-J uses parallel attention/MLP blocks, an untied head and partial rotary embeddings, whereas ours keeps GPT-3's sequential blocks and tied head and applies rotary embeddings to the full head.
- **GPT-NeoX** adopts the parallel blocks, the untied head and partial rotary embeddings.
- **Llama 3** keeps the shared SwiGLU width (the multiple of 32 nearest $8d/3$) and an ordinary causal mask over the packed stream, whereas the released models use wider feed-forward layers and a document-boundary mask. Its target key/value-to-query head ratio of $1/4$ is rounded to the nearest valid head divisor, giving realized query/KV ratios of 3, 4, 5 and 7.
- **LLaMA, Llama 3 and Delphi** use PyTorch 2.8's default RMSNorm epsilon (about 1.19×10^{-7} in the FP32/BF16 path), while OLMo 2 and OLMo 3 set it to 10^{-6}.

- **OLMo 2 and OLMo 3** follow the pinned DataDecide and OLMo-core ladder configurations, respectively. OLMo 2 uses weight decay 0.05 and no z-loss, and OLMo 3 uses the ladder's small-batch Adam $\beta_2 = 0.99$ and warmup-stable-decay schedule. Our SkipStep implementation for OLMo 3 computes its rolling statistics from the previous 127 observations rather than upstream's 128.
- **Delphi** is a local PyTorch adaptation of the recipe in the pinned Marin Completed AdamH v6 suite (Marin Community, 2026), trained from scratch under the shared tokenizer, context, batch and corpora. It applies AdamH to a fused QKV matrix and clips all parameter gradients jointly, whereas upstream uses separate Q, K and V matrices and clips the AdamH and Adam parameter groups separately.

B Training and evaluation protocol

B.1 Model-size selection and seeds

Each recipe trains a ladder of model sizes at every budget, and the size with the lowest mean native validation loss is selected as its compute-optimal size.

The six 2019–2024 recipes and Delphi share one ladder of shapes with head dimension 64, labeled s1, s2, … in order of size (Table 9). At a given shape, each recipe's MLP and attention design (SwiGLU width, grouped-query attention) sets its parameter count. OLMo 2 uses the published DataDecide ladder (Table 10), and OLMo 3 its own published ladder, of which we trained the sizes from 30M to 760M. The OLMo sizes are labeled by nominal size (e.g. 150M), which also indexes their learning-rate settings. Throughout, N is the non-embedding parameter count of Kaplan et al. (2020), the weight matrices of the attention and MLP blocks, which defines $C = 6ND$ and is counted the same way for every recipe (Appendix D.4).

Table 9: **The shared model ladder.** Layers L, width d and heads (head dimension 64) of each size, and its non-embedding parameter count N (Appendix D.4) for the GELU (GPT-2, GPT-3, GPT-J and GPT-NeoX), LLaMA (SwiGLU), Llama 3 (SwiGLU with grouped-query attention) and Delphi (SwiGLU with MLP width $4d$) architecture families. Llama 3's key/value heads, in parentheses, are the divisor of its head count nearest to a quarter of it.

				Non-embedding N			
Size	L	d	Heads	GELU	LLaMA	Llama 3 (KV)	Delphi
s1	5	320	5	6.1M	6.2M	5.4M (1)	8.2M
s2	6	384	6	10.6M	10.6M	9.4M (2)	14.2M
s3	7	448	7	16.9M	16.8M	14.4M (1)	22.5M
s4	8	512	8	25.2M	25.3M	22.2M (2)	33.6M
s5	10	640	10	49.2M	48.9M	42.4M (2)	65.5M
s6	12	768	12	84.9M	84.9M	74.3M (3)	113.2M
s7	14	896	14	134.9M	135.3M	116.0M (2)	179.8M
s8	16	1024	16	201.3M	200.8M	175.6M (4)	268.4M
s9	20	1280	20	393.2M	394.0M	344.9M (5)	524.3M
s10	24	1536	24	679.5M	679.5M	594.5M (6)	906.0M

Table 10: **The OLMo 2 ladder (DataDecide).** The shape of each size and its non-embedding parameter count N, counted as for every recipe (Appendix D.4).

Size	L	d	Heads	MLP width	N
10M	8	144	8	576	2.7M
14M	8	192	8	768	4.7M
16M	8	208	8	832	5.5M
20M	16	192	8	768	9.4M
60M	16	384	12	1536	37.7M
90M	16	528	12	2112	71.4M
150M	12	768	12	3072	113.2M
300M	16	1024	16	4096	268.4M
530M	16	1344	16	5376	462.4M
750M	16	1536	16	6144	604.0M

B.2 Downstream score

OLMES10 is the unweighted mean of ten task accuracies, each computed on the task's full evaluation split, so every checkpoint answers the same items. For each task we take the better of its cloze (ranked-classification) and multiple-choice formulations, as in OLMES (Gu et al., 2025). For MMLU, each formulation is first averaged over the 57 subjects.

B.3 Compute multipliers and paired-seed intervals

Computing a multiplier. Each configuration, a recipe trained on a corpus, gives an accuracy curve, which plots its OLMES10 accuracy at each of the five budgets against training compute on a logarithmic axis. To compare a candidate configuration with a reference, we take the accuracy that the reference reaches at a given budget and ask how much compute the candidate needs to reach the same accuracy. We connect the candidate's measured points with straight lines on the log-compute axis and read off where its curve reaches the reference accuracy. The multiplier is the reference's compute divided by this candidate compute (Section 3). When the crossing falls below the candidate's smallest budget or above its largest, we extend the nearest line segment and mark the result with ‡.

Measuring compute. We measure compute as $C = 6ND$, where D is the number of training tokens and N is the non-embedding parameter count of Appendix B.1 (Appendix D.4). Every point is placed at its actual compute, which for most recipes equals the nominal budget. The OLMo runs set their token counts with the OLMo ladders' own parameter-counting convention, so their points lie below the nominal budgets (the selected OLMo 2 points by 13–65%). When we report a multiplier at a given budget, the reference point is the reference's run at that nominal budget, placed at its actual compute. Appendix D.4 also repeats the key comparisons with a fuller compute count.

Seeds. Each curve point averages three training runs that differ only in their random seed (6198, 6199 or 6200), which sets both the model initialization and the order of the training data. A few seeds were trained more than once in the same configuration. Their runs are averaged first, so that each seed counts once.

Paired-seed intervals. With only three seeds, we estimate uncertainty by resampling the seeds rather than by assuming a distribution. A resample draws three seed labels with replacement, for example (6198, 6198, 6200). There are $3^3 = 27$ such ordered draws, and we use all of them. For each draw, we recompute every curve point as the mean of the drawn seeds and then recompute the multiplier. The same draw is used for the reference and the candidate and at every budget, so the two curves are always averaged over the same seed labels. If the candidate fails to reach the reference accuracy in any of the 27 draws, we report no interval. Ratios of multipliers, the Shapley share and the annual rates are computed within each draw in the same way. Because they rest on three seeds, these intervals are coarse, and they treat the selected model sizes, tasks and learning rates as fixed.

B.4 SHAPLEY DECOMPOSITION OF THE JOINT GAIN

In the 2×2 design, let $v(S)$ be the log compute multiplier of the set S of upgrades applied to the 2019 stack, so that $v(\emptyset) = 0$, $v(\{d\}) = \ln M_{\text{data}}$, $v(\{r\}) = \ln M_{\text{recipe}}$ and $v(\{d, r\}) = \ln M_{\text{joint}}$. The Shapley value of the corpus upgrade averages its marginal contribution over the two orders in which the upgrades can be made,

$$\phi_d = \tfrac{1}{2}\, v(\{d\}) + \tfrac{1}{2}\, \big(v(\{d, r\}) - v(\{r\})\big),$$

and the recipe upgrade's value ϕ_r is defined symmetrically, so that $\phi_d + \phi_r = v(\{d, r\})$. We report $\phi_d / v(\{d, r\})$ as data progress's share of the joint gain. Because the two individual log-gains sum to more than the joint one (Section 4.3), neither individual multiplier can be read directly as a share, whereas the Shapley value splits the joint gain exactly.

C LEARNING RATES

Each run's peak learning rate is set in two stages. First, a rule fitted to small-model sweeps gives every recipe a *calibrated* rate for any model size and budget (Appendices C.1 and C.3). Second, we test that rate directly at the model sizes actually used and replace it where three seeds confirm a better rate. The result is the *selected* rate, which all main results use (Appendix C.4). Rates are always judged on native validation loss, the loss on held-out text from the model's own training corpus, because it is far less noisy than downstream accuracy. We call one recipe at one budget, trained at a given model size, a cell, and write alternative rates as multiples of the calibrated rate, so that $2\times$ is twice the calibrated rate.

C.1 PROCEDURE

Every learning rate in the paper is set in four steps.

1. Starting point. Each recipe starts from a prior estimate for its peak learning rate as a function of model size N and training tokens per parameter D/N (Appendix C.2). Published rates were tuned for other batch sizes, context lengths and model sizes, and the best rate depends on model size and batch size (Porian et al., 2024; Bi et al., 2024). The prior then serves to center our subsequent search.

2. Calibration. For the six 2019–2024 recipes, we train small models at a set of anchor settings that span model sizes from about 6×10^6 to 10^8 parameters and 5 to 56 training tokens per parameter. At each anchor we try three rates a factor of 2 apart with three seeds each, centered on the prior times a constant factor per recipe (from 0.67 for LLaMA to 7.1 for GPT-2), and at some anchors further rates, mostly where the best one lies at the edge. Ignoring diverged runs, we fit a parabola in log learning rate through the best multiple and its two neighbors and take its minimum as the anchor's optimal rate. Fitting one rule to the 21 usable anchor optima gives exponents shared by all six recipes and a separate level for each (Appendix C.3). OLMo 2 and OLMo 3 keep their published rules, and Delphi, which uses a different optimizer, gets its own rule fitted in the same way (Appendix C.3).

3. Extrapolation. The rule gives the calibrated rate for any model size and budget. The larger models selected at the top budgets lie outside the range of the anchors, so their rates are extrapolated.

4. Verification and selection. We then train each recipe with learning rates near the calibrated rate, and rerun the most promising alternatives with more seeds. A tested rate replaces the calibrated one only where a complete three-seed comparison finds lower native loss (Appendix C.4). Model sizes are not re-selected after a change of learning rate.

C.2 PRIORS

The priors of the six 2019–2024 recipes are set as follows. Except for GPT-2's, the priors are power laws that scale as N^b in model size and as $(D/N)^{-0.15}$ in tokens per parameter; we give each prior's value at $N = 10^8$ and $D/N = 20$ and its size exponent b.

- **GPT-2** publishes no peak learning rate, so its prior transfers GPT-1's Adam rate of 2.5×10^{-4}, with no dependence on model size or training duration.

- **GPT-3, GPT-J and LLaMA** take their prior levels from the best rates of earlier small-model sweeps at the four smallest shared shapes (s1–s4), re-centered at $N = 10^8$ and $D/N = 20$, which gives 1.58×10^{-3} for GPT-3 and GPT-J and 2.80×10^{-3} for LLaMA, with $b = -0.25$.
- **GPT-NeoX** starts from its published rate, 5.20×10^{-4}, with a size exponent $b = -0.326$ fitted to the released GPT-NeoX v1.0 model configurations.
- **Llama 3** starts from its published rate, 2.00×10^{-3}, with $b = -0.25$, checked against a small-model sweep.

C.3 Calibrated learning-rate rule

For the six 2019–2024 recipes, the calibrated peak learning rate follows a fitted rule with exponents shared by all recipes and a level A_i specific to recipe i:

$$\mathrm{lr}^\star(N, D) \;=\; A_i \cdot \left(\frac{N}{10^8}\right)^{-0.267} \cdot \left(\frac{D/N}{20}\right)^{+0.202},$$

where N is the model size and D/N the number of training tokens per parameter. The rule is fitted to the three-seed sweeps at the anchors of Appendix C.1, centered on each recipe's prior (Appendix C.2). Table 11 gives the fitted levels. The anchors have at most about 10^8 parameters, so for larger models the rule is extrapolated, and Appendix C.4 tests it there.

The three remaining recipes follow their own rules.

- **OLMo 2** uses the per-size rates published for the DataDecide ladder instead of our calibration, transferred to our batch size and training length as $\mathrm{lr} = \mathrm{lr}_{\mathrm{DD}}\,(B/B_{\mathrm{DD}})\,\sqrt{100\,N_{\mathrm{rung}}/D}$. Here $\mathrm{lr}_{\mathrm{DD}}$ and B_{DD} are the peak rate and batch size (in tokens) that DataDecide publishes for the size, $B = 262{,}144$ is our batch size, N_{rung} is the size's nominal parameter count (e.g. 1.5×10^8 for 150M), D is the run's number of training tokens, and 100 is DataDecide's number of training tokens per parameter.
- **OLMo 3** uses its published rule instead of our calibration, $\mathrm{lr} = 0.00235\,(N_{\mathrm{rung}}/1.08 \times 10^8)^{-1/3}$, where N_{rung} is the size's parameter count in the published ladder. The rule does not depend on training length.
- **Delphi**, which uses a different optimizer, has its own rule fitted to five anchors in the same way, which sets $\mathrm{lr} = 0.00796\,(N/10^8)^{-0.39}\,((D/N)/20)^{-0.05}$ for the AdamH matrix parameters, with the pinned ratio setting the Adam rates of the remaining parameters.

Table 11: Learning-rate rules. The six 2019–2024 recipes share the calibrated rule $\mathrm{lr}^\star = A_i\,(N/10^8)^{-0.267}\,((D/N)/20)^{+0.202}$, so the level A_i is each recipe's calibrated peak rate at $N = 10^8$ and $D/N = 20$. Delphi's rule has the same form with its own fitted constants. OLMo 2 and OLMo 3 follow their published rules, with the symbols defined in the text.

Recipe	Rule	A_i	Size exponent	D/N exponent
GPT-2	shared calibrated rule	0.00132	-0.267	$+0.202$
GPT-3	shared calibrated rule	0.00158	-0.267	$+0.202$
GPT-J	shared calibrated rule	0.00183	-0.267	$+0.202$
GPT-NeoX	shared calibrated rule	0.00113	-0.267	$+0.202$
LLaMA	shared calibrated rule	0.00263	-0.267	$+0.202$
Llama 3	shared calibrated rule	0.00184	-0.267	$+0.202$
Delphi	own calibrated rule (AdamH matrix parameters)	0.00796	-0.39	-0.05
OLMo 2	$\mathrm{lr}_{\mathrm{DD}}\,(B/B_{\mathrm{DD}})\,\sqrt{100\,N_{\mathrm{rung}}/D}$	—	—	—
OLMo 3	$0.00235\,(N_{\mathrm{rung}}/1.08{\times}10^8)^{-1/3}$	—	$-1/3$	0

C.4 Checking and selecting the learning rates

We checked the calibrated rate for every recipe at 3.16×10^{18} FLOPs, the seven year-representative recipes at 10^{18} FLOPs, and GPT-2, LLaMA and OLMo 2 at 10^{19} FLOPs, with nearby rates from $0.25\times$ to $2\times$ the calibrated rate, mostly with a single seed (Table 12). The calibrated rate is usually

at or near the best. At 3.16×10^{18} FLOPs it is the best rate tried in 9 of the 14 cells, and where another rate does better, the gain is at most 0.03 nats, worth no more than about $\times 1.3$ in compute.

Rerunning the most promising alternatives with three seeds changed the rate in six cells (Table 13). All other cells keep the calibrated rate, including GPT-J and LLaMA at 3.16×10^{18} FLOPs, whose better single-seed rates were not rerun.

Table 12: **Learning-rate checks.** Each row trains one recipe, corpus and model size at one budget with several peak learning rates, given as multiples of the calibrated rate (Appendix C.3), and reports the native validation loss (nats per token on held-out text from the training corpus). The $1\times$ column is the calibrated rate's three-seed mean $\pm$ seed SD, and other entries are single runs unless marked [3] (three seeds). "—" marks a rate not tried and "div." a run that diverged.

Recipe	Corpus	Size	$0.25\times$	$0.35\times$	$0.5\times$	$0.71\times$	$1\times$ (calib. $\pm$ SD)	$1.41\times$	$2\times$
10^{18} FLOPs									
GPT-2	FineWeb-Edu	s6		—	—	3.2785	3.2370	3.2117 $\pm$ 0.0041	3.2175	div.
GPT-3	FineWeb-Edu	s6		—	—	3.2176	3.1688	3.1560 $\pm$ 0.0150	3.1668	3.1661
GPT-J	FineWeb-Edu	s6		—	—	3.1330	3.1091	3.1148 $\pm$ 0.0016	3.1225	3.1750
GPT-NeoX	FineWeb-Edu	s6		—	—	3.2053	3.1657	3.1503 $\pm$ 0.0025	3.1368	3.1379
LLaMA	FineWeb-Edu	s6		—	—	3.1300	3.1137	3.1198 $\pm$ 0.0054	div.	div.
Llama 3	FineWeb-Edu	s6		—	—	3.1296	3.1176	3.1323 $\pm$ 0.0272	3.1148	3.2296
OLMo 2	FineWeb-Edu	150M		—	—	3.1735	3.1809	3.1764 $\pm$ 0.0029	3.1814	3.2053
3.16×10^{18} FLOPs									
GPT-2	FineWeb-Edu	s5		—	—	3.1218	—	3.1248 $\pm$ 0.0069	—	3.1302
GPT-2	FineWeb-Edu	s7		—	—	3.0899	—	3.0262 $\pm$ 0.0047	3.0373	div.
GPT-2	FineWeb-Edu	s9		—	—	3.2463	—	3.1627 $\pm$ 0.0118	—	div.
GPT-3	FineWeb-Edu	s7		—	—	3.0155	—	2.9940 $\pm$ 0.0205	—	3.1397
GPT-J	FineWeb-Edu	s7		—	—	2.9622[3]	2.9453	2.9611 $\pm$ 0.0041	2.9636	3.0089
GPT-NeoX	FineWeb-Edu	s7		—	—	3.0118	3.0021	2.9870 $\pm$ 0.0017	2.9887	2.9822[3]
LLaMA	FineWeb-Edu	s7	2.9819	2.9687	2.9486	2.9438	2.9738 $\pm$ 0.0105	—	div.
Llama 3	FineWeb-Edu	s7		—	—	2.9655	—	2.9498 $\pm$ 0.0064	—	3.0324
OLMo 2	FineWeb-Edu	150M		—	—	2.9803	—	2.9572 $\pm$ 0.0038	—	2.9629
OLMo 3	FineWeb-Edu	100M		—	—	3.0655	—	3.0674 $\pm$ 0.0041	—	3.0915
OLMo 3	DCLM-QC7/FW3	190M		—	—	2.8767	—	2.8666 $\pm$ 0.0047	—	2.8916
OLMo 3	FineWeb-Edu	190M		—	—	3.0138	—	3.0078 $\pm$ 0.0038	—	3.0283
OLMo 3	FineWeb-Edu	370M		—	—	3.0336	—	3.0074 $\pm$ 0.0023	—	3.0087
Delphi	FineWeb-Edu	s7		—	—	2.9764	—	2.9272 $\pm$ 0.0015	—	2.9640
10^{19} FLOPs									
GPT-2	FineWeb-Edu	s8		—	—	2.8934	—	2.8684 $\pm$ 0.0030	—	div.
LLaMA	FineWeb-Edu	s9		—	—	2.8085	2.8093[3]	2.8315 $\pm$ 0.0048	—	—
OLMo 2	FineWeb-Edu	300M		—	—	2.8146	—	2.7915 $\pm$ 0.0013	—	2.7721[3]

Recipe	FLOPs	Calibrated LR	Candidate LR	Native gain	Selected LR
GPT-J	10^{18}	0.0019681	0.0013916 (0.71×)	+0.00363	0.0013916
GPT-NeoX	10^{18}	0.0012153	0.0017186 (1.41×)	+0.01266	0.0017186
LLaMA	10^{18}	0.0028284	0.0020000 (0.71×)	-0.00444	0.0028284
Llama 3	10^{18}	0.0021643	0.0030608 (1.41×)	+0.02075	0.0030608
LLaMA	10^{19}	0.0016083	0.0011372 (0.71×)	+0.02228	0.0011372
OLMo 2	10^{19}	0.0031678	0.0063356 (2.00×)	+0.01938	0.0063356
GPT-NeoX	3.16×10^{18}	0.0011242	0.0022484 (2.00×)	+0.00476	0.0022484

Table 13: Selected learning rates. Each row is a cell whose best alternative rate was rerun with three seeds. The candidate is that alternative, with its multiple of the calibrated rate in parentheses, and the native gain is the calibrated-rate minus the candidate mean NLL at the originally selected model size.

D Compute-optimal model sizes and compute accounting

D.1 isoFLOP profiles

Figure 6 shows the loss-versus-size profiles from which the model sizes are selected, for all nine recipes.

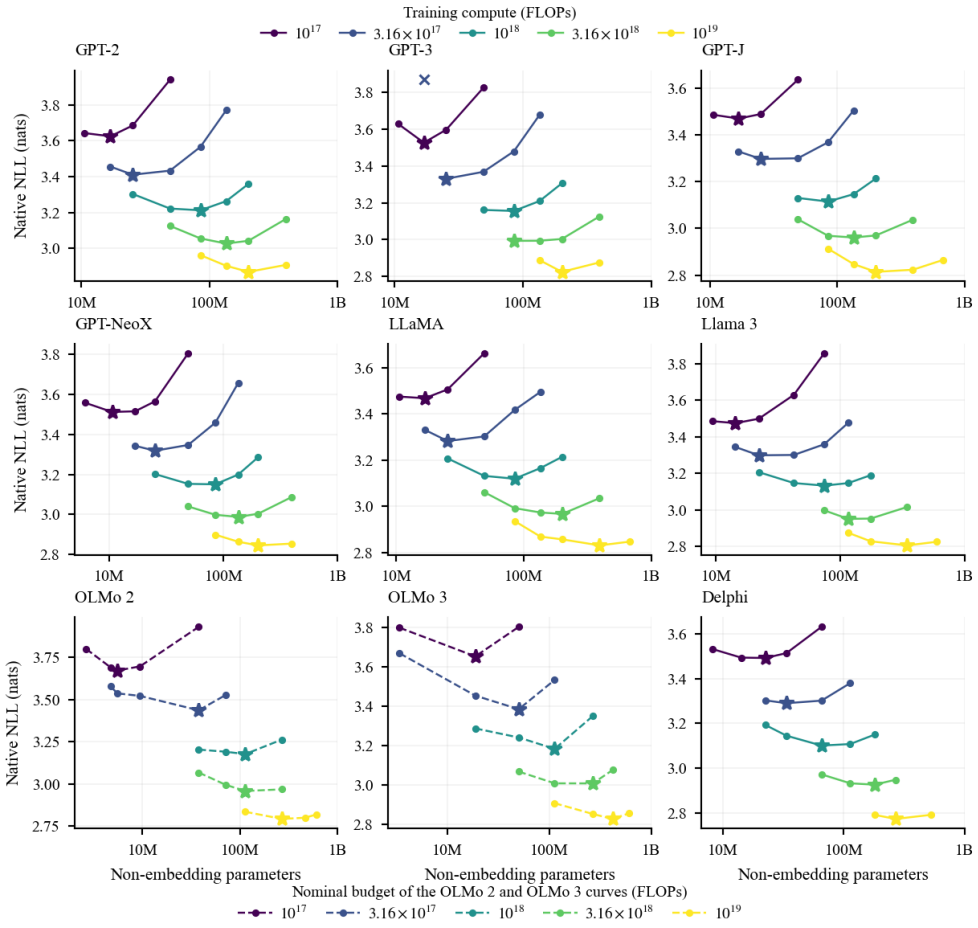


Figure 6: isoFLOP profiles of all nine recipes on FineWeb-Edu, showing native validation loss against non-embedding parameters (Appendix D.4) with stars at the minimizing sizes. × marks GPT-3's smallest size at 3.16×10^{17} FLOPs, where two of three seeds became unstable. The dashed OLMo curves are labeled by their nominal budget. The OLMo ladders set each run's token count from their own parameter count, which is larger than N, so each OLMo run uses only the fraction N/N_{ladder} of that budget. This fraction depends only on the model size and rises from 0.27 (OLMo 2, 10M) and 0.21 (OLMo 3, 30M) to 0.89 at the largest sizes.

D.2 Selected model sizes at 10^{19} FLOPs

Table 14 lists the selected recipe-sweep model sizes at 10^{19} FLOPs. Later recipes tend to select larger models, and hence fewer tokens per parameter, though not monotonically by year.

Table 14: Selected recipe-sweep model sizes at 10^{19} nominal FLOPs. N is the non-embedding parameter count (Appendix D.4), D/N the realized number of training tokens per parameter, and loss the native seed mean $\pm$ seed SD.

Recipe	Size	Selected N	D/N	Seeds	Native NLL
GPT-2	s8	201.3M	41.1	3	2.8684 ± 0.0030
GPT-3	s8	201.3M	41.1	3	2.8247 ± 0.0019
GPT-J	s8	201.3M	41.1	3	2.8141 ± 0.0067
GPT-NeoX	s8	201.3M	41.1	3	2.8446 ± 0.0038
LLaMA	s9	394.0M	10.7	3	2.8093 ± 0.0044
Llama 3	s9	344.9M	14.0	3	2.8043 ± 0.0090
OLMo 2	300M	268.4M	19.4	3	2.7721 ± 0.0026

D.3 The compute-optimal size across corpora

Under a fixed recipe, the compute-optimal size depends very little on the corpus (Figure 7). Most corpora therefore reuse the size selected for their recipe on FineWeb-Edu, while OpenWebText under GPT-2 has its own ladder. At 10^{19} FLOPs, OLM-CC uses the next larger size (530M, 3.1B tokens) instead of the 300M size selected on FineWeb-Edu.

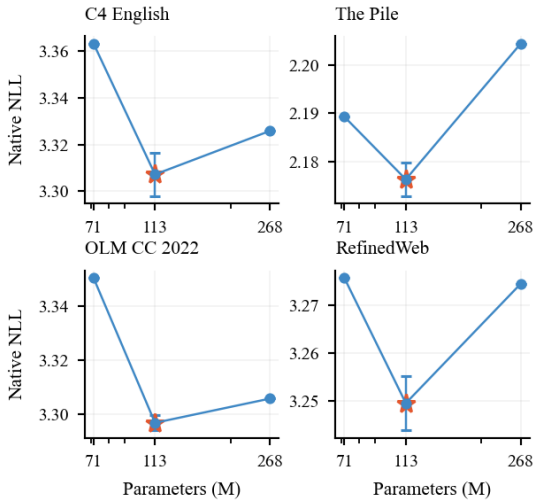


Figure 7: Native loss of the 90M, 150M and 300M OLMo 2 ladder sizes at 3.16×10^{18} FLOPs on four corpora, against non-embedding parameters (Appendix D.4). The 150M size, selected on FineWeb-Edu, is best on each corpus. Each panel has its own y-scale.

D.4 Parameter and compute accounting

Parameter count. N is the non-embedding parameter count of Kaplan et al. (2020, Eq. 2.1 and Table 1), which counts the weight matrices of the attention and MLP blocks. For a model with L layers, width d and MLP width d_{ff},

$$N = L\left(2d^2 + 2d\,d_{\text{kv}} + k\,d\,d_{\text{ff}}\right),$$

where $2d^2$ counts the query and output projections, $2d\,d_{\text{kv}}$ the key and value projections, and $k\,d\,d_{\text{ff}}$ the MLP, with $k = 2$ for GELU and $k = 3$ for SwiGLU. The key/value width d_{kv} equals d, except under Llama 3's grouped-query attention, where it is the head dimension times the number of key/value heads. For the GELU recipes, whose MLP width is $4d$, this gives Kaplan et al.'s $N = 12Ld^2$. N excludes the input embedding, the learned positional embeddings of GPT-2 and GPT-3, and the output head, whether it is tied to the input embedding (GPT-2, GPT-3 and GPT-J) or untied (the other recipes). Like Kaplan et al., it also omits biases and normalization gains. Tables 9 and 10 list N for every size.

Compute conventions. With D training tokens, context length $T = 2048$ and vocabulary size $V = 50{,}257$, this appendix uses three conventions.

- **Nominal.** $C = 6ND$ is the non-embedding training compute of Kaplan et al. (2020), with six FLOPs per parameter and token for the forward and backward passes, and is used throughout the paper. It omits the FLOPs of attention over the context and of the output head.
- **Chinchilla** (Hoffmann et al., 2022, Appendix F). The parameter count adds the embeddings to N (the input embedding, the learned positional embeddings and any untied output head), and $C = D\,(6N + 12LTd + 12dV)$ counts all training FLOPs, including attention over the context and the input and output embeddings as matrix products. We use it to compare our compute-optimal sizes with Chinchilla's.
- **Fuller.** $C = D\,(6N + 6dV + 12LTd)$ is the nominal compute plus the FLOPs of the output head and of attention over the context. The head is counted for every recipe, since a tied head does the same computation as an untied one. We use it to test the headline comparisons.

Scaling of the compute-optimal size. In the Chinchilla convention, our compute-optimal sizes agree with Chinchilla's. Across the seven recipes they scale as $N^\star \propto C^{0.544}$ (Figure 8; $C^{0.48}$ to $C^{0.64}$ for individual recipes), close to the $C^{0.49}$ of Chinchilla's isoFLOP profiles (Hoffmann et al., 2022, Table 2). At our largest budget, 1.4×10^{19} to 1.8×10^{19} FLOPs, the selected sizes of 250M to 520M parameters lie around the 400M that Chinchilla projects as compute-optimal for 1.92×10^{19} FLOPs (Hoffmann et al., 2022, Table 3).

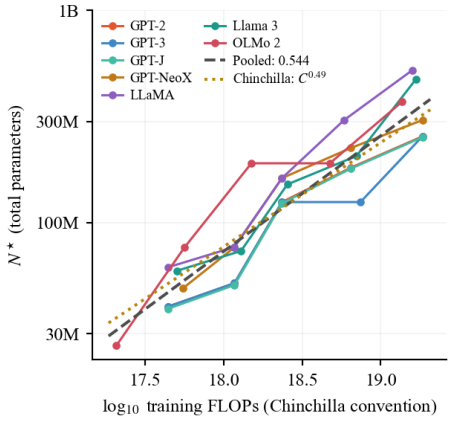


Figure 8: Compute-optimal size $N^\star$ (all parameters) of the 35 selected recipe-sweep sizes against training FLOPs, both in the Chinchilla convention, with the pooled slope $C^{0.544}$ and Chinchilla's isoFLOP exponent $C^{0.49}$ (Hoffmann et al., 2022).

Fuller compute accounting. The fuller count leaves the conclusion unchanged. It adds the most compute to the smallest models, moving their points to the right by up to $\times 4.1$ (Figure 9). The data-to-recipe ratio of the 2×2 design falls from 2.4 to 2.0, so the corpus upgrade still attains about twice the recipe upgrade's multiplier. The sweeps keep the same ordering (Table 15).

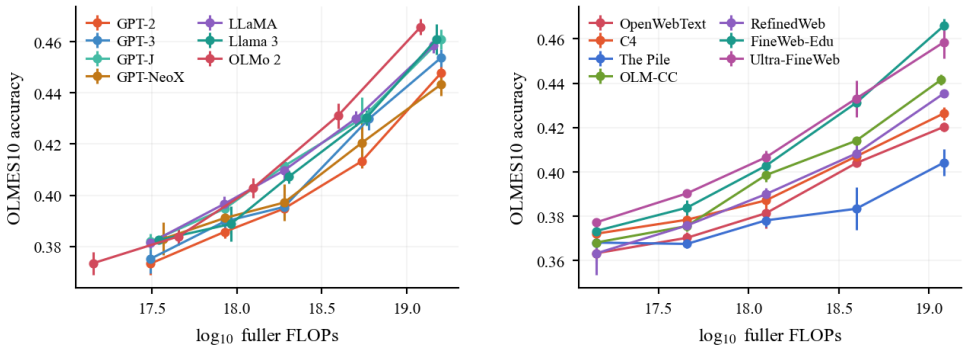


Figure 9: OLMES10 accuracy against compute in the fuller convention for the recipe sweep (left) and the corpus sweep (right).

Table 15: Multipliers at 10^{19} FLOPs in the nominal and fuller conventions, with the scored checkpoints, model sizes and reference accuracies held fixed. Brackets are 68% paired-seed intervals.

Quantity	Nominal	Fuller
2×2: corpus upgrade	12.00 [10.37, 13.67]	9.50 [8.58, 10.41]
2×2: recipe upgrade	4.98 [4.10, 5.91]	4.73 [3.91, 5.61]
2×2: both upgrades	21.07 [16.36, 26.70]	18.21 [14.90, 21.98]
2×2: data/recipe ratio	2.41 [2.30, 2.53]	2.01 [1.85, 2.20]
Corpus sweep (OLMo 2)	6.20 [5.21, 6.91]	5.34 [4.49, 5.95]
Recipe sweep (FineWeb-Edu)	2.31 [2.02, 2.71]	2.36 [2.09, 2.71]

E FURTHER RESULTS

E.1 MULTIPLIERS AT EVERY BUDGET

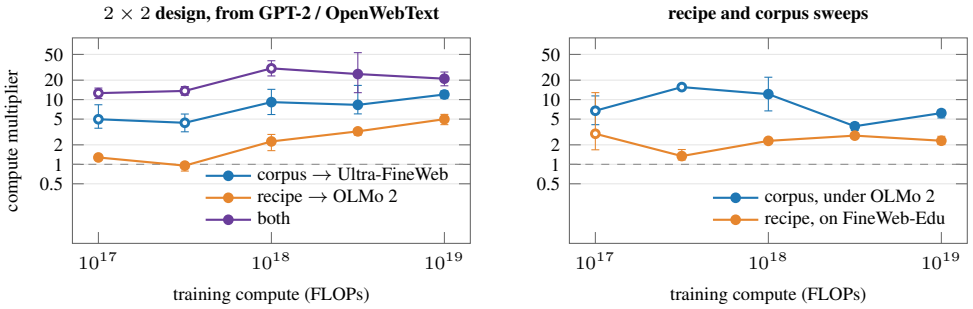


Figure 10: **Data progress attains greater compute multipliers than algorithmic progress at every budget.** Compute multipliers of the corpus upgrade (data progress) and the recipe upgrade (algorithmic progress) against the reference's training compute. **Left:** upgrades from GPT-2 trained on OpenWebText (the 2×2 design). **Right:** the same two upgrades in the recipe and corpus sweeps. Bars are 68% paired-seed intervals, omitted where a resampled curve does not reach the reference. Hollow markers are crossings outside the measured compute range, and the dashed line marks a multiplier of 1.

Figure 10 shows how the compute multipliers vary with the budget. Extending these curves to larger budgets would show how the multipliers depend on scale.

2×2 **design.** The left panel starts from GPT-2 trained on OpenWebText. The corpus upgrade to Ultra-FineWeb attains a larger multiplier than the recipe upgrade to OLMo 2 at every budget. The recipe multiplier rises with compute, from $\times 0.95$ at 3.16×10^{17} FLOPs, where OLMo 2 is no better than GPT-2, to $\times 5.0$ at 10^{19} FLOPs. The corpus multiplier lies between $\times 4.4$ and $\times 12.0$. As the recipe multiplier grows, the ratio of the two narrows.

Sweeps. The right panel shows the corpus upgrade from OpenWebText to Ultra-FineWeb under OLMo 2 and the recipe upgrade from GPT-2 to OLMo 2 on FineWeb-Edu. Here too the corpus upgrade is larger at every budget, by a factor of at least 1.4 (at 3.16×10^{18} FLOPs).

E.2 DELPHI AGAINST OLMO 2

At 10^{19} FLOPs the Delphi recipe attains $\times 2.3$ [2.2, 2.5] over GPT-2 on FineWeb-Edu, on par with OLMo 2 ($\times 2.3$). On native loss it attains $\times 2.03$ [2.01, 2.05], against $\times 2.31$ for OLMo 2. Its implementation is described in Appendix A and its learning rate in Appendix C.

Figure 11 compares Delphi with OLMo 2 and GPT-2 on FineWeb-Edu and on OpenWebText.

Table 16 reports Delphi on FineWeb-Edu at all five budgets.

Table 16: The Delphi recipe on FineWeb-Edu at every budget. Accuracy and native NLL are means $\pm$ sample SD over three seeds, with repeated runs averaged within seed. Multipliers are relative to GPT-2 on the same corpus, with 68% paired-seed intervals; ‡ marks an extrapolated crossing.

Nominal FLOPs	N (M)	OLMES10	Native NLL	OLMES10 multiplier	NLL multiplier
10^{17}	22.5	0.3819 ± 0.0027	3.4918 ± 0.0026	$2.17^{\ddagger}$ [2.05, 2.37]	$2.13^{\ddagger}$ [2.09, 2.17]
3.16×10^{17}	33.6	0.3948 ± 0.0017	3.2902 ± 0.0072	2.26 [1.93, 2.62]	1.98 [1.95, 2.02]
10^{18}	65.5	0.4081 ± 0.0033	3.1014 ± 0.0027	3.18 [2.81, 3.52]	1.96 [1.92, 2.01]
3.16×10^{18}	179.8	0.4395 ± 0.0039	2.9272 ± 0.0015	2.62 [2.44, 2.88]	1.92 [1.91, 1.94]
10^{19}	268.4	0.4700 ± 0.0020	2.7745 ± 0.0011	2.32 [2.16, 2.51]	2.03 [2.01, 2.05]

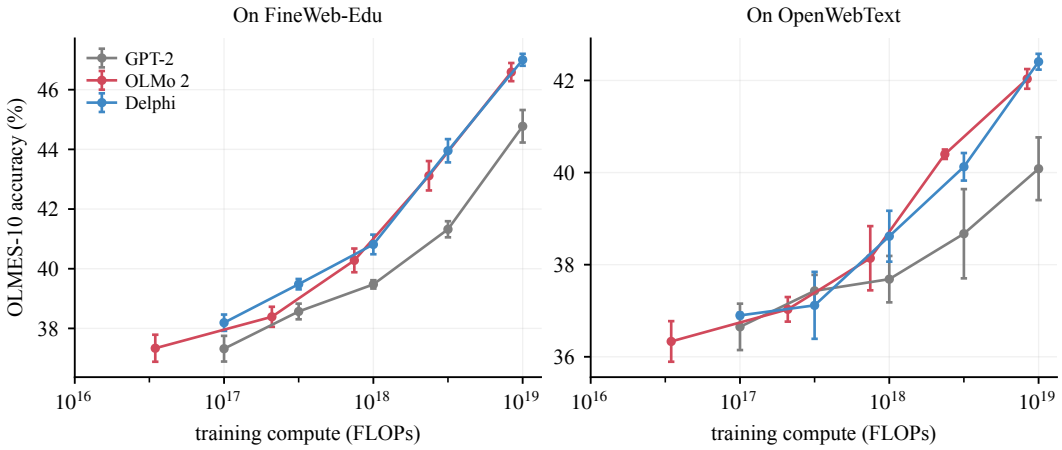


Figure 11: **Delphi and OLMo 2 reach the same accuracy at 10^{19} FLOPs on both corpora.** OLMES10 accuracy against training compute for GPT-2, OLMo 2 and Delphi on FineWeb-Edu (left) and on OpenWebText (right), with error bars of ± 1 seed SD.

E.3 From OLMo 2 to OLMo 3

OLMo 3 keeps the OLMo 2 architecture and changes the training recipe. It adds a z-loss, trains with SkipStep AdamW, $\beta_2 = 0.99$ and weight decay 0.1 instead of AdamW with $\beta_2 = 0.95$ and weight decay 0.05, and decays the learning rate with a warmup-stable-decay schedule instead of a cosine (Table 7). It also uses its own published model ladder and learning-rate rule. Figure 12 compares the two recipes on the three corpora on which both have five-budget, three-seed curves. At our scales the OLMo 3 recipe does not improve on OLMo 2.

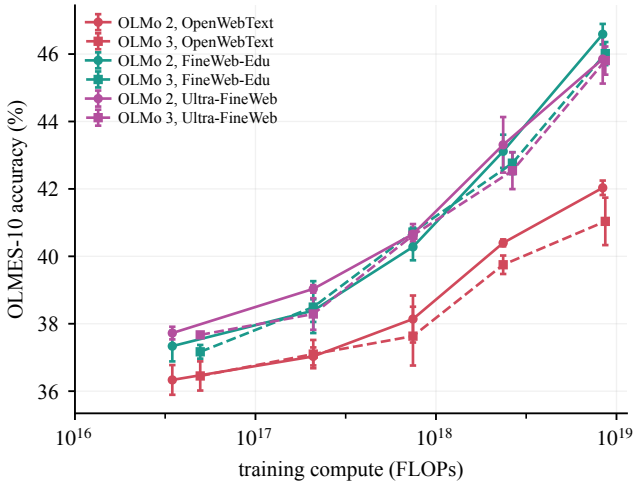


Figure 12: OLMES10 accuracy against training compute for OLMo 2 (solid) and OLMo 3 (dashed) on OpenWebText, FineWeb-Edu and Ultra-FineWeb, with error bars of ± 1 seed SD.

E.4 Additional corpora under OLMo 3

Figure 13 compares, under OLMo 3, the two DCLM corpora and Ultra-FineWeb with FineWeb-Edu.

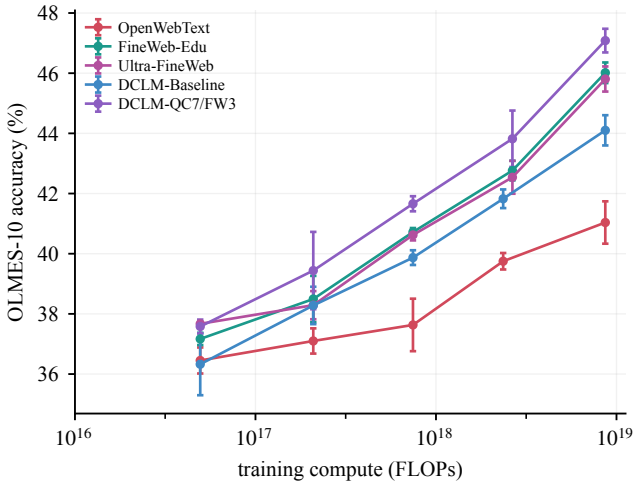


Figure 13: OLMES10 accuracy against training compute for five corpora trained with OLMo 3, with error bars of ± 1 seed SD.

E.5 Raw accuracies

Table 17 gives the OLMES10 accuracies at 10^{19} FLOPs behind the multipliers of Tables 4 and 5.

Table 17: OLMES10 accuracy at 10^{19} FLOPs (seed mean $\pm$ seed SD) at the selected model size of each cell. The 2×2 block gives the four cells behind Figure 1. MMLU is averaged over its 57 subjects.

Yr	Recipe (on FineWeb-Edu)	OLMES10	Yr	Corpus (under OLMo 2)	OLMES10
2019	GPT-2	0.4477 ± 0.0054	2019	OpenWebText	0.4203 ± 0.0021
2020	GPT-3	0.4538 ± 0.0035	2020	C4	0.4264 ± 0.0029
2021	GPT-J	0.4609 ± 0.0039	2021	The Pile	0.4042 ± 0.0060
2022	GPT-NeoX	0.4433 ± 0.0044	2022	OLM-CC	0.4415 ± 0.0023
2023	LLaMA	0.4585 ± 0.0028	2023	RefinedWeb	0.4354 ± 0.0017
2024	Llama 3	0.4610 ± 0.0060	2024	FineWeb-Edu	0.4659 ± 0.0030
2025	OLMo 2	0.4659 ± 0.0030	2025	Ultra-FineWeb	0.4585 ± 0.0073
2 × 2 design: three distinct seeds per cell; runs in parentheses					
GPT-2 / OpenWebText (3 runs)		0.4008 ± 0.0068	GPT-2 / Ultra-FineWeb (4 runs)		0.4464 ± 0.0055
OLMo 2 / OpenWebText (4 runs)		0.4203 ± 0.0021	OLMo 2 / Ultra-FineWeb (6 runs)		0.4585 ± 0.0073

E.6 The recipe–corpus grid

Table 18 lists the OLMES10 accuracy of every recipe–corpus pair we trained at 3.16×10^{18} FLOPs, each at its recipe's selected model size. It holds two grids, and the additive recipe-plus-corpus model fits both closely.

- **The 7×7 grid** of Section 4.3 crosses the year-representative recipes with the year-representative corpora (the first seven rows and columns). The additive model explains 88.8% of the variance in accuracy (Figure 4).
- **The 8×10 grid** covers all 80 cells, adding Delphi and the three additional corpora. The additive model explains 91.8% of the variance in accuracy. Corpus effects account for 84.4% and recipe effects for 7.3%, leaving 8.2% for interactions and noise.

Table 18: **OLMES10 accuracy (%) of the 8×10 grid at 3.16×10^{18} FLOPs.** Eight recipes (rows) trained on ten corpora (columns), each at its recipe's selected model size. The first seven rows and columns are the 7×7 grid behind Figures 3 and 4 (the recipes of Table 2 and corpora of Table 3). The Delphi row and the last three columns are the additional recipe and corpora. Cells with three seeds give the seed mean $\pm$ sample SD, and the others are single runs. GPT-3 on the Pile was unstable in training.

		Year-representative corpora							Additional corpora		
Year	Recipe	Open-WebText	C4	The Pile	OLM-CC	Refined-Web	FineWeb-Edu	Ultra-FineWeb	DCLM-Baseline	DCLM-QC7/FW3	Dolma 3 Mix
2019	GPT-2	39.1 ± 0.4	38.8	38.4	40.4	39.4	41.3 ± 0.3	42.0 ± 0.6	40.8	44.0 ± 0.3	39.7
2020	GPT-3	38.6	40.0	36.1	41.5	40.6	43.0 ± 0.4	41.4	41.6	44.2	40.5
2021	GPT-J	39.1	40.5	39.1	42.0	41.6	43.0 ± 0.8	43.4	42.7	45.0	40.1
2022	GPT-NeoX	39.5	40.2	39.2	40.8	41.1	42.0 ± 1.0	43.0	41.4	44.2	40.3
2023	LLaMA	40.0	39.6	37.9	41.4	41.0	43.0 ± 0.3	42.5	40.8	44.3	40.9
2024	Llama 3	40.1	41.3	39.1	39.8	41.3	43.0 ± 0.3	43.6	41.8	44.3	41.7
2025	OLMo 2	40.4 ± 0.1	40.7 ± 0.4	38.3 ± 1.0	41.4 ± 0.1	40.8 ± 0.5	43.1 ± 0.5	43.3 ± 0.8	42.3	45.0	40.9
2026	Delphi	40.1 ± 0.3	41.1	39.4	41.1	41.5	44.0 ± 0.4	43.2	41.4	45.4 ± 0.2	42.2

E.7 NATIVE VALIDATION LOSS OF THE RECIPE SWEEP

We also look at the native validation losses of every recipe trained on FineWeb-Edu, on a held-out subsection of the corpus.

Figure 14 plots this loss against training compute, and Table 19 gives the compute multipliers it implies over GPT-2, computed as for Table 4 with loss in place of OLMES10 accuracy.

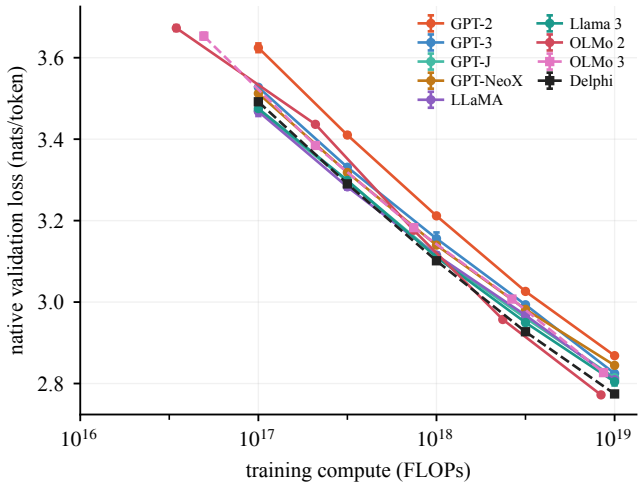


Figure 14: Native validation loss on held-out FineWeb-Edu against training compute for each year's recipe (solid) and for OLMo 3 and Delphi (dashed), each at its selected model size. Points are means over three seeds; error bars are ± 1 seed SD.

Table 19: **Compute multipliers on native validation loss.** Each recipe trained on FineWeb-Edu, relative to GPT-2 at each budget, with native validation loss (lower is better) in place of OLMES10 accuracy. The year representatives are above the rule, and OLMo 3 and Delphi below it. Brackets are 68% paired-seed intervals.

Year	Recipe	10^{18}	3.16×10^{18}	10^{19}
2019	GPT-2	1.00 (ref)	1.00 (ref)	1.00 (ref)
2020	GPT-3	1.44 [1.42, 1.46]	1.26 [1.23, 1.30]	1.35 [1.34, 1.36]
2021	GPT-J	1.85 [1.81, 1.88]	1.63 [1.59, 1.67]	1.53 [1.51, 1.55]
2022	GPT-NeoX	1.59 [1.57, 1.61]	1.38 [1.35, 1.41]	1.22 [1.22, 1.22]
2023	LLaMA	1.94 [1.89, 2.00]	1.58 [1.54, 1.62]	1.54 [1.49, 1.58]
2024	Llama 3	1.86 [1.82, 1.89]	1.73 [1.69, 1.77]	1.66 [1.60, 1.72]
2025	OLMo 2	1.59 [1.58, 1.61]	1.93 [1.92, 1.94]	2.31 [2.29, 2.33]
2025	OLMo 3	1.61 [1.59, 1.63]	1.37 [1.34, 1.39]	1.52 [1.50, 1.53]
2026	Delphi	1.96 [1.92, 2.01]	1.92 [1.91, 1.94]	2.03 [2.01, 2.05]

F A SECOND TASK SUITE

To test whether the advantage of data improvements over model improvements is specific to OLMES10, we scored checkpoints on eight format-diverse tasks.

Tasks. The suite comprises eight tasks. LAMBADA (Paperno et al., 2016) (narrative cloze, 5,153 items) is scored by whether greedy decoding reproduces the target. COPA (Roemmele et al., 2011) (100 items, the SuperGLUE validation split; Wang et al., 2019) and SciQ (Welbl et al., 2017) (1,000 items) are multiple choice, scored by raw choice accuracy. SQuAD (Rajpurkar et al., 2016), Natural Questions (Kwiatkowski et al., 2019), Jeopardy (Allen Institute for AI, 2025), CoQA (Reddy et al., 2019) and DROP (Dua et al., 2019) (1.6–2.1k items each) are converted to ranked multiple choice and scored by choice accuracy after per-character likelihood normalization. The evaluator scores continuations only and runs no open-ended generation. Diverse8 is the unweighted mean of the eight tasks. None of these tasks was used in selecting FineWeb-Edu's data, but DCLM's core evaluation includes COPA, LAMBADA, SQuAD, CoQA and Jeopardy (Li et al., 2024, Appendix G), so the suite is not independent of every corpus's development.

Table 20 compares the headline multipliers of the main text on the two suites.

Table 20: **Headline compute multipliers on OLMES10 and on Diverse8**, from the same checkpoints and with the same crossing construction, at 3.16×10^{18} and 10^{19} FLOPs. Brackets are 68% paired-seed intervals; ratios are taken within each seed resample. ‡ marks a crossing outside the measured compute range. The sweeps were scored on Diverse8 at these two budgets only.

	OLMES10		Diverse8	
Comparison	@3.16×10^{18}	@10^{19}	@3.16×10^{18}	@10^{19}
2 × 2 design, from GPT-2 on OpenWebText				
Corpus upgrade	8.30 [6.02, 16.59]	12.00 [10.37, 13.67]	4.54 [4.20, 4.88]	4.31 [3.97, 4.67]
Recipe upgrade	3.23 [2.83, 3.55]	4.98 [4.10, 5.91]	1.90 [1.74, 2.07]	2.07 [2.01, 2.12]
Both upgrades	24.84 [12.81, 53.19]	21.07 [16.36, 26.70]	3.76 [3.48, 4.12]	5.72 [5.34, 6.09]
Ratio $M_{\text{data}}/M_{\text{recipe}}$	2.57 [2.13, 4.68]	2.41 [2.30, 2.53]	2.39 [2.22, 2.59]	2.09 [1.92, 2.26]
Sweeps				
Corpus sweep: OpenWebText → Ultra-FineWeb	3.88 [3.35, 4.38]	6.20 [5.21, 6.91]	3.41‡ [2.76, 4.23]	3.12 [2.76, 3.55]
Recipe sweep: GPT-2 → OLMo 2	2.78 [2.56, 3.01]	2.31 [2.02, 2.71]	1.30 [1.25, 1.34]	1.66 [1.57, 1.74]
Ratio $M_{\text{data}}/M_{\text{recipe}}$	1.40 [1.11, 1.73]	2.68 [2.16, 3.37]	2.63 [2.10, 3.30]	1.88 [1.59, 2.21]

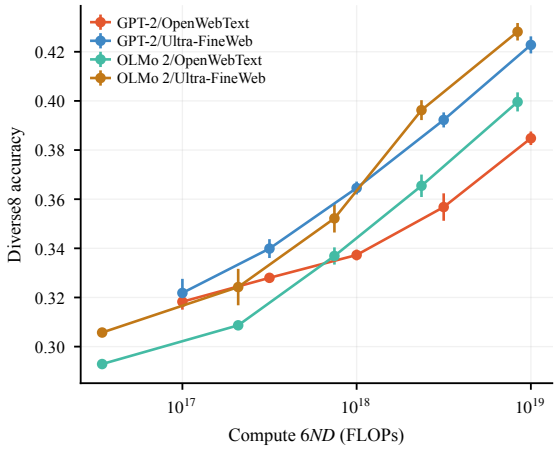


Figure 15: Diverse8 accuracy against compute for the 2 × 2 design at all five budgets. Points are means over three seeds; error bars are ±1 seed SD.

Results. The corpus upgrade again attains a larger multiplier than the recipe upgrade, on both suites and at both budgets (Table 20 and Figure 15). The Diverse8 multipliers are notably smaller than their OLMES10 counterparts.

Table 21: Second-suite accuracy per task at 10^{19} FLOPs (seed means at the selected model size), with OLMES10 alongside. Diverse8 and OLMES10 give the seed mean $\pm$ sample SD.

Configuration	n	LAMB	COPA	SciQ	SQuAD	NQ	Jeop	CoQA	DROP	Diverse8	OLMES10
Recipe sweep (FineWeb-Edu)											
GPT-2	3	0.306	0.667	0.597	0.496	0.238	0.387	0.492	0.275	0.432 $\pm$0.007	0.448 $\pm$0.005
GPT-3	3	0.303	0.643	0.611	0.507	0.246	0.410	0.529	0.259	0.439 $\pm$0.003	0.454 $\pm$0.003
GPT-J	3	0.310	0.657	0.606	0.508	0.242	0.416	0.507	0.274	0.440 $\pm$0.004	0.461 $\pm$0.004
GPT-NeoX	3	0.292	0.687	0.603	0.478	0.241	0.404	0.480	0.275	0.432 $\pm$0.007	0.443 $\pm$0.004
LLaMA	3	0.296	0.690	0.619	0.477	0.244	0.397	0.484	0.273	0.435 $\pm$0.001	0.459 $\pm$0.003
Llama 3	3	0.296	0.663	0.613	0.486	0.245	0.409	0.503	0.258	0.434 $\pm$0.006	0.461 $\pm$0.006
OLMo 2	3	0.308	0.677	0.617	0.498	0.250	0.437	0.509	0.262	0.445 $\pm$0.009	0.466 $\pm$0.003
Corpus sweep (OLMo 2)											
OpenWebText	3	0.333	0.685	0.461	0.464	0.209	0.316	0.464	0.264	0.400 $\pm$0.004	0.420 $\pm$0.002
C4	3	0.321	0.673	0.473	0.422	0.215	0.299	0.435	0.253	0.386 $\pm$0.004	0.426 $\pm$0.003
The Pile	3	0.332	0.643	0.480	0.440	0.210	0.297	0.437	0.259	0.387 $\pm$0.007	0.404 $\pm$0.006
OLM-CC	3	0.345	0.693	0.472	0.450	0.211	0.312	0.459	0.274	0.402 $\pm$0.004	0.442 $\pm$0.002
RefinedWeb	3	0.341	0.683	0.491	0.452	0.212	0.317	0.452	0.271	0.402 $\pm$0.004	0.435 $\pm$0.002
FineWeb-Edu	3	0.308	0.677	0.617	0.498	0.250	0.437	0.509	0.262	0.445 $\pm$0.009	0.466 $\pm$0.003
Ultra-FineWeb	3	0.276	0.653	0.637	0.478	0.235	0.379	0.498	0.269	0.428 $\pm$0.004	0.459 $\pm$0.007
2 × 2 design											
GPT-2 / OpenWebText	3	0.313	0.643	0.436	0.463	0.202	0.293	0.461	0.267	0.385 $\pm$0.003	0.401 $\pm$0.007
GPT-2 / Ultra-FineWeb	3	0.288	0.638	0.606	0.516	0.220	0.342	0.508	0.265	0.423 $\pm$0.003	0.446 $\pm$0.005
OLMo 2 / OpenWebText	3	0.333	0.685	0.461	0.464	0.209	0.316	0.464	0.264	0.400 $\pm$0.004	0.420 $\pm$0.002
OLMo 2 / Ultra-FineWeb	3	0.276	0.653	0.637	0.478	0.235	0.379	0.498	0.269	0.428 $\pm$0.004	0.459 $\pm$0.007

Table 22: Compute multipliers on Diverse8, with the same crossing construction as OLMES10. The 2×2 design was scored at all five budgets. The recipe and corpus sweeps were scored at 3.16×10^{18} and 10^{19} FLOPs only, so their curves have two points and extrapolated crossings (‡) are common. The corpus sweep is referenced to C4.

Configuration	@3.16×10^{18}	@10^{19}
Recipe sweep (reference GPT-2)		
GPT-3	1.23‡ [1.16, 1.32]	1.24 [1.17, 1.32]
GPT-J	1.20‡ [1.17, 1.23]	1.28 [1.15, 1.43]
GPT-NeoX	0.89 [0.82, 0.95]	1.01 [0.93, 1.08]
LLaMA	0.86 [0.81, 0.93]	1.08 [0.99, 1.18]
Llama 3	1.02‡ [0.90, 1.18]	1.07 [0.89, 1.23]
OLMo 2	1.30 [1.25, 1.34]	1.66 [1.57, 1.74]
Corpus sweep (reference C4)		
OpenWebText	1.11‡ [1.07, 1.14]	1.63 [1.51, 1.78]
The Pile	0.88 [0.81, 1.00]	1.04 [0.85, 1.24]
OLM-CC	1.31‡ [1.11, 1.62]	1.80 [1.69, 1.96]
RefinedWeb	1.12‡ [1.05, 1.18]	1.74 [1.59, 1.91]
FineWeb-Edu	2.54‡ [2.41, 2.69]	4.78‡ [4.44, 5.09]
Ultra-FineWeb	3.80‡ [3.23, 4.51]	5.26‡ [4.58, 6.02]
2 × 2 design (reference GPT-2 / OpenWebText)		
data upgrade	4.54 [4.20, 4.88]	4.31 [3.97, 4.67]
recipe upgrade	1.90 [1.74, 2.07]	2.07 [2.01, 2.12]
both	3.76 [3.48, 4.12]	5.72 [5.34, 6.09]

The two classifier-filtered corpora have the highest Diverse8 means in the corpus sweep, but their advantage depends on the task. FineWeb-Edu and Ultra-FineWeb outperform C4 on science and ranked reading-comprehension tasks, while the broad web corpora do better on LAMBADA narrative cloze (Table 21 and Figure 16).

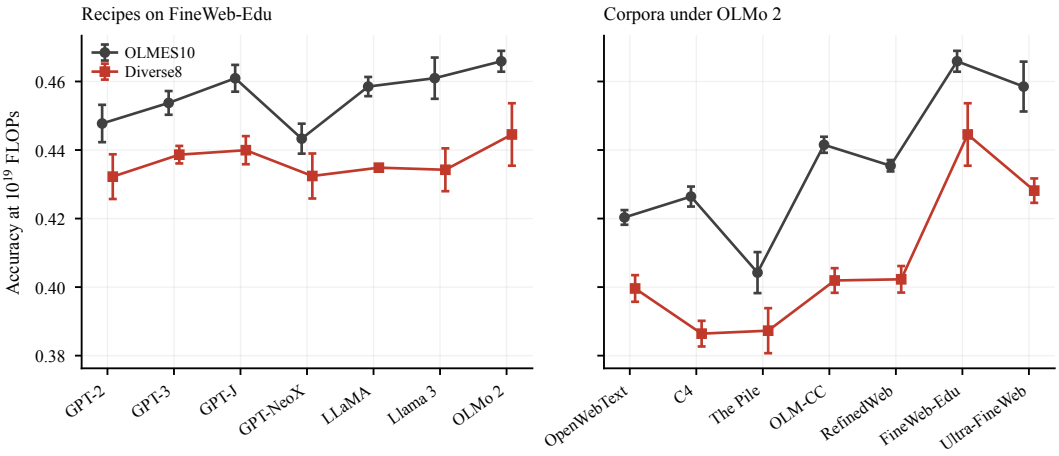


Figure 16: Diverse8 and OLMES10 accuracy at 10^{19} FLOPs for every year's recipe on FineWeb-Edu (left) and OLMo 2 on every year's corpus (right). Points are means over three seeds; error bars are ± 1 seed SD.