SMELT: SCALING LAWS FOR COMPUTE-MATCHED MOE LOOPED TRANSFORMERS

Anonymous authors
Paper under double-blind review

ABSTRACT

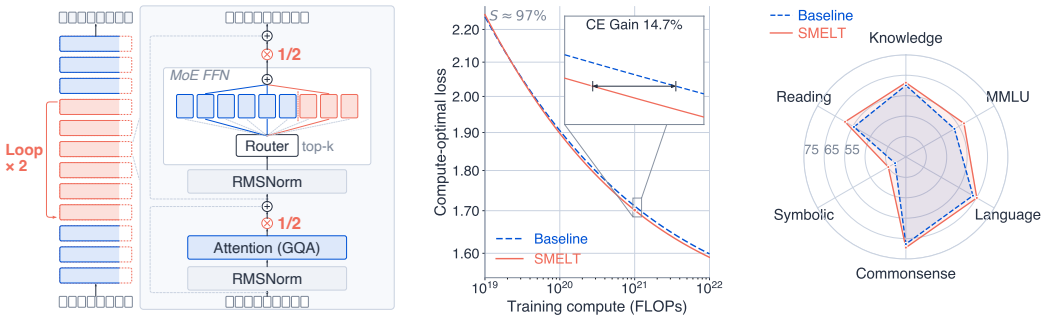
Looped Transformers iterate a shared block of layers to gain effective depth without adding parameters, and prior work reports that this extra depth improves language modeling, multi-hop reasoning, and math. However, most of these evaluations compare models of the same size, where the looped model also spends more FLOPs per token, so its gains mix architectural advantage with extra compute. Meanwhile, work that compares Looped Transformers with unlooped ones at the same FLOPs often finds that the looped models perform worse, but there the unlooped models store more parameters. Therefore, to investigate whether looping itself has an advantage, we study looping on Mixture-of-Experts Transformers and compare each looped model with an unlooped Baseline that closely matches it in per-token FLOPs, total non-embedding parameters, and KV cache. Through three ablations, we arrive at an effective recipe we call SMELT (**S**parse **M**o**E** Transformer, middle layers **L**oop **T**wice), which loops the middle half of layers twice and pays for the extra layer executions by narrowing the model. We scale SMELT and the Baseline up to 54B total non-embedding parameters and fit a separate Chinchilla-style scaling law to each architecture. SMELT's loss drops faster with compute, saving 14.7–18.0% of training compute on the compute-optimal frontier at 10^{21} FLOPs. Beyond validation loss, SMELT's advantage exceeds what its loss gap predicts on downstream benchmarks and grows with sample length and the number of in-context examples. Mechanistic analysis shows that, on their second pass, the looped layers put less attention on the attention sink and more on content-relevant tokens, an inductive bias that may underlie the observed performance gains. These results show that looping can improve Transformers even under matched budgets.

1 INTRODUCTION

Looped Transformers have drawn rapidly growing attention because they promise more capable language models without more parameters (Zhu et al., 2025; Prairie et al., 2026). A looped Transformer runs the same layers several times per forward pass, so the model executes more layers than it stores (Dehghani et al., 2019). Theoretically, a Transformer's depth bounds how many serial steps it can perform (Merrill & Sabharwal, 2023; Li et al., 2024b), and a looped model can run many such steps with the parameters of only a few layers (Saunshi et al., 2025). Empirically, looped models improve reasoning on arithmetic, multi-hop induction, and math (Geiping et al., 2025; Saunshi et al., 2025), and match larger unlooped models (Yang et al., 2024; Zhu et al., 2025).

Many of these comparisons, however, hold the parameter count fixed, so the looped model executes more layers and spends more compute per token than the unlooped model it is compared with (Geiping et al., 2025; Zhu et al., 2025; Prairie et al., 2026). A comparison at equal compute instead uses an *untied baseline* that executes as many layers as the looped model but gives each layer its own parameters. In language modeling, this *untied baseline* consistently reaches lower loss than the looped model and also leads on knowledge tasks (Saunshi et al., 2025; Kapl et al., 2026; Schwethelm et al., 2026). However, the untied baseline also stores more parameters. It is known that how much a model can memorize is bounded by its stored parameters (Allen-Zhu & Li, 2025; Morris et al., 2025), and irrelevant to loop count (Zhu et al., 2025). We therefore suspect that the untied baseline wins because it stores more parameters, not because looping uses compute inefficiently. A fair test of looping must therefore hold three budgets fixed at once: *1) per-token FLOPs*, the cost of training

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.



(a) The SMELT recipe. (b) Compute-optimal frontier. (c) Downstream accuracy.

Figure 1: **SMELT overview and main results.** (a) The SMELT recipe applied to an unlooped MoE Baseline: loop the middle 50% of layers twice, narrow the hidden dimension, raise the expert count to restore total parameters, and scale the looped residuals by $1/2$, alongside a smaller attention head size at a higher GQA ratio that holds KV cache nearly unchanged across the extra layer executions. Together these changes closely match the Baseline's per-token FLOPs, total non-embedding parameters, and KV cache (Section 2). (b) Compute-optimal loss at compute-equivalent sparsity $S \approx 97\%$ (defined in Section 2.1), from a Chinchilla-style scaling law fitted separately to each architecture. SMELT's frontier drops faster; at $C = 10^{21}$ FLOPs it reaches the same loss with 14.7% less compute (Section 3). (c) Downstream accuracy at the 1.6B scale (54B non-embedding parameters, $S \approx 97\%$), over the five DCLM Core categories (Appendix F.3), with MMLU on its own axis. Each axis is the unweighted mean accuracy over its tasks. SMELT leads the Baseline on all six axes.

and inference; *2) total parameters*, the capacity for knowledge; and *3) KV cache*, which relates to long-context performance and limits servable context length. (Table 1).

We use Mixture-of-Experts (MoE) Transformers to make this budget matching feasible. We start from an unlooped MoE model, which we call the Baseline, and loop some of its layers, so the looped model must trade hidden dimension or physical depth for its extra layer executions. Because MoE decouples total parameters from per-token FLOPs (Fedus et al., 2022; Shazeer et al., 2017), the looped model can then restore the total parameter count by raising the expert count. KV cache parity is restored separately by adjusting the head size and GQA ratio. We call the Baseline and the looped model that matches it on all three budgets a *matched pair*. Under this matching, we search the loop design space through three ablations and find three rules: *1) loop the middle half* of layers rather than the full stack, *2) give the looped model a larger effective depth-to-width ratio* than the Baseline, and *3) loop twice* rather than three or four times. Together these lock in a recipe we call **SMELT**, a **S**parse **M**o**E** Transformer whose middle layers **L**oop **T**wice (Figure 1; Section 2).

We scale the recipe up to 54B non-embedding parameters across four model sizes and four sparsity levels, comparing SMELT against the Baseline at each grid point. SMELT reaches lower loss than the Baseline at every scale and sparsity level. To predict how the advantage evolves at scale, we fit separate Chinchilla-style scaling laws (Hoffmann et al., 2022) for the two architectures. SMELT's loss drops faster with both compute and data, and the CE Gain increases across the fitted compute range and is largest on Code among validation domains (Section 3).

Beyond aggregate validation loss, we examine downstream benchmarks, sample length, and in-context learning. On DCLM (Li et al., 2024a) and MMLU (Hendrycks et al., 2021), SMELT outperforms the Baseline in nearly all matched pairs, with an improvement that exceeds what the validation loss gap alone would predict. The advantage is further amplified on longer samples and when more in-context demonstrations are available, suggesting that looping becomes more valuable as the input provides more context to build on (Section 4).

To build intuition for these advantages, we examine what the looped layers compute differently on their second visit. We find that the experts selected and the tokens attended to overlap substantially across the two visits, while the residual updates grow larger and stay aligned, indicating that the second visit amplifies the first rather than overwriting it. Within the attention mechanism, values change more than queries and keys, and the attention sink is broadly reduced, redirecting mass toward content-relevant tokens, which may relate to the ICL advantage (Appendix B).

Table 1: Comparison of looped Transformer studies in terms of architecture, loop span, matched budgets, scaling analysis, and key finding. ✓ = matched; ✗ = unmatched or grows with loop count; (✓) = partially matched (see notes). Span denotes the fraction of the layer stack that is repeated, and r denotes the number of times that span executes per token (both are defined in Section 2.1).

			Matched budgets				
Work	Arch.	Span	FLOPs/ token	Stored params	KV cache	Scaling analysis	Key finding
Huginn (Geiping et al., 2025)	Dense	Mid. 50%	✗	✓	✗	Fixed size (3.5B)	Prelude-recur-coda looping improves reasoning; extra FLOPs uncontrolled
Ouro (Zhu et al., 2025)	Dense	Full (100%)	✗	✓	✗	Scaling law (53M–1.36B)	Full-stack looping improves reasoning; extra FLOPs uncontrolled
Saunshi et al. (2025)	Dense	Mid. 33%	(✓)[a]	✗	✗	Fixed size (170M–1B)	Middle-block looping lowers perplexity; inductive bias toward reasoning
Schwethelm et al. (2026)	Dense	Mid.[c]	✓	✗	✓	Iso-depth law (25M–1.6B)	r recurrences $\approx r^{0.46}$ unique equivalents; looping loses
Prairie et al. (2026)	Dense	Mid. 33%	✗	✓	✗	Scaling law at fixed params (100M–1.3B)	Optimal r grows with compute; FLOPs and KV grow with r
Lee et al. (2026)	MoE	Full (100%)	✓	✗	✓	IsoFLOP law (16M–305M active)	Sparse routing mitigates the loop penalty; Looped-MoE beats dense Base but trails MoE in loss
LoopMoE (Chen et al., 2026a)	MoE	Mid. 33%	✓	✓	✗	Fixed size (3B, 9B)	Wins 8/9 benchmarks at 3B and 9/9 at 9B
Gao et al. (2026)	MoE	Each layer	✗[b]	✗	✗	Wall-clock-matched scaling ladder	Wins; per-loop gain largest at $r=2$; memory savings reinvested into width and depth
SMELT (ours)	MoE	Mid. 50%	✓	✓	✓	Scaling ladder across 0.1B–54B non-embedding parameters	Wins; 14.7–18.0% FLOPs saving on validation loss at 10^{21} FLOPs; benchmark gains exceed validation loss prediction

[a] Matched-FLOPs perplexity comparison for the prefix-loop-suffix layout; the accuracy-vs-loop-count results are at fixed model size. [b] Matches wall-clock training time; uses about $1.4\times$ the per-token FLOPs.
[c] Physical span varies with loop count (approx. 67%, 50%, 33% for r=2, 4, 8).

Our results show that looping can improve Transformers even under budget matching, with particularly strong gains on structured data and in-context learning tasks. Together with the growing body of work exploring looped language models from complementary angles (Geiping et al., 2025; Zhu et al., 2025; Saunshi et al., 2025; Schwethelm et al., 2026; Prairie et al., 2026), they suggest that looping is a robust and increasingly well-understood design axis.

2 Design Recipe under Matched Compute

2.1 Experimental setup

Models and setup. We write *Baseline* (M_{B}) for the standard unlooped MoE Transformer and *Looped Transformer* (M_{L}) for the variant that repeats a contiguous span of m layers r times. Both are decoder-only Transformers with grouped-query attention (Ainslie et al., 2023) and MoE feed-forward layers that route each token to its top-8 experts. We build each M_{L} from an M_{B} so that the two match in per-token FLOPs, total parameters, and KV cache, and we call the two models a *matched pair* (Section 2.2). This matching changes M_{L} alone, so the two members of a matched pair differ only in the loop configuration and in the width, expert, and head hyperparameters of M_{L}. Throughout this paper, parameter counts exclude embeddings. In the Looped Transformer, each sublayer's residual update within the looped span is scaled by $1/r$ (Wang et al., 2026) (Figure 1a); this prevents correlated weight-tied updates from inflating the residual stream across visits. All runs share the same optimizer, data, and schedule, and we compare models on held-out validation loss, DCLM Core, and DCLM Completion, reporting MMLU separately; Appendix F details the training setup, metrics, evaluation protocol, and per-model configurations.

Compute-equivalent sparsity. We compare the two architectures at several MoE sparsity levels, so both members of a matched pair need the same sparsity value. Parameter-ratio sparsity (Abnar et al., 2025), the share of stored parameters that a token leaves inactive, fails here: a looped layer runs r times per token, so the Looped Transformer spends more compute per active parameter and would be labeled sparser than its matched Baseline. We therefore count active parameters by compute, which also captures how attention cost varies with sequence length. At each scale, we measure each

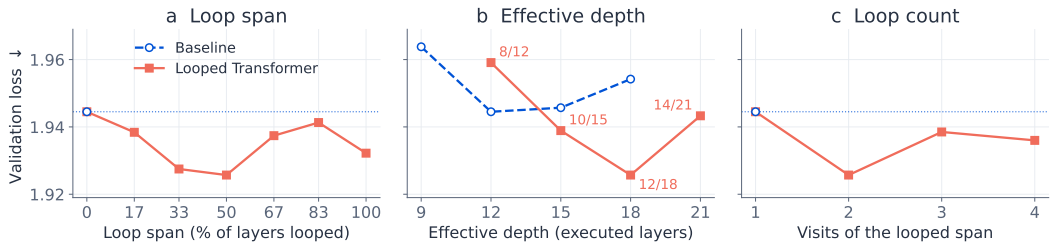


Figure 2: Validation loss of the three design ablations at 200M and $S \approx 85\%$. (a) Loop span with two passes: loss is lowest when the middle 50% of layers loop. (b) Effective depth: the Baseline is best at 12 layers, while the Looped Transformer is best at 12 physical and 18 executed layers. (c) Loop count: two visits give the lowest loss. Appendix F.5 reports the full results, including DCLM metrics and the $S \approx 95\%$ span sweep.

architecture against its own *reference model* at $S = 0\%$, which for the Baseline has eight experts per layer and activates all of them for every token. Its per-token training FLOPs and total non-embedding parameters are denoted by F_0 and N_0, respectively. For any configuration of that architecture and scale with per-token FLOPs F and N total non-embedding parameters, we define

$$N_{\text{act}}^{\text{eq}} = \frac{F}{F_0} N_0, \qquad S = 1 - \frac{N_{\text{act}}^{\text{eq}}}{N}, \tag{1}$$

so each reference model corresponds to $S = 0\%$ by construction. For an unlooped Baseline, S reduces to the usual parameter-ratio sparsity. We train matched pairs at four nominal sparsity levels, $S = 0\%$ and $S \approx 85\%$, 95%, and 97%. Both members of a pair belong to the same level, although their measured S values differ slightly (Appendix F.4).

2.2 Budget matching for looped models

To attribute performance differences to architecture, each M_{L} must match its M_{B} in per-token FLOPs, total parameters, and KV cache. We build M_{L} by first looping a span of M_{B}'s layers and then adjusting the looped model's width, expert count, and attention heads until all three budgets match those of M_{B}. Looping affects these three budgets differently. Looping a span of m layers r times increases the effective depth, the number of executed layers, from L to $L_{\text{eff}} = L + (r-1)\,m$ without adding parameters. Per-token FLOPs scale with L_{eff}, approximately as $L_{\text{eff}} \cdot H^2$ for hidden dimension H. KV cache also scales with L_{eff}, because each executed layer stores its own keys and values. Before any adjustment, the looped model therefore exceeds M_{B} in FLOPs and KV cache while matching it in total parameters. This intermediate model belongs to no matched pair, and the following three adjustments turn it into M_{L}.

To offset the additional FLOPs, we narrow H or reduce the physical depth L; Section 2.4 determines which option performs better. Both options reduce total parameters: narrowing H shrinks every expert, and reducing L removes entire layers of experts. We restore the total parameter count by adding experts to every layer, which leaves per-token FLOPs unchanged because each token is still routed to only its top-8 experts. Finally, we reduce KV cache to the level of M_{B} with a smaller attention head size and a higher GQA ratio. The resulting M_{L} and the unchanged M_{B} form a matched pair. Table 2 summarizes, for each budget, the source of mismatch, the adjustment that corrects it, and the mismatch after matching. Appendix F.4 lists the matched configurations and budget ratios for every pair, with a worked example.

Table 2: Each budget's source of mismatch, its correction, and the typical mismatch after matching.

Budget	Source of mismatch	Adjustment	Mismatch
Per-token FLOPs	Scales with L_{eff}	Narrow H or reduce L	$< 4\%$
Total parameters	Reduced as H or L shrinks	Add experts	$< 1\%$
KV cache	Scales with L_{eff}	Smaller heads, higher GQA ratio	$< 4\%$

We apply this matching at each of the four compute-equivalent sparsity levels (Section 2.1). We name each model scale by the Baseline's active non-embedding parameter count. With matching in place, we search the Looped Transformer design space through three ablations at 200M (Figure 2): which layers to loop, how deep the model should be, and how many passes to run.

2.3 LOOPING THE MIDDLE HALF BEATS FULL LOOPING

We first ask which layers should loop. With two passes and physical depth $L = 12$, we vary the length of the looped middle segment from 0 layers (the Baseline) to all 12 (full looping), at $S \approx 85\%$ and $S \approx 95\%$. Validation loss is lowest at a 50% span for both sparsity levels (Figure 2(a); Table 9). The DCLM metrics are noisier across spans (at $S \approx 95\%$, DCLM Core peaks at 67%), so we select on validation loss and use DCLM only as a consistency check. We fix the span at 50% for all subsequent experiments. This optimum agrees with depth-dependent specialization in Transformer representations (Lad et al., 2024; Tenney et al., 2019) and with partial-looping studies (Saunshi et al., 2025; Kapl et al., 2026; Cai et al., 2026): the first and last layers serve specialized roles that benefit from independent parameters.

2.4 THE LOOPED TRANSFORMER PREFERS A LARGER EFFECTIVE DEPTH-TO-WIDTH RATIO

Section 2.2 left open whether to offset the additional FLOPs of looping by narrowing H or by reducing L. To decide, we sweep the depth-to-width ratio of both architectures at 200M and $S \approx 85\%$, with a 50% span and two passes, re-matching the other hyperparameters at each depth (Figure 2(b); Table 10). The Baseline is best at 12 layers, while the Looped Transformer is best at 12 physical and 18 executed layers, so the Looped Transformer prefers a larger effective depth-to-width ratio. We therefore keep the Looped Transformer's physical depth equal to the Baseline's and offset the additional FLOPs by narrowing H. We hypothesize that the shared layers receive gradients from several visits, including later visits with shorter paths to the output, which may make the additional executed depth easier to optimize than an equally deep stack of independent layers.

2.5 TWO LOOPS BEAT THREE OR FOUR

The remaining question is how many times the middle half should loop. Each extra visit costs FLOPs, so matching makes the model narrower; Figure 2(c) shows the 200M, $S \approx 85\%$ sweep from one to four visits. Two visits give the best validation loss, DCLM Core, and DCLM Completion (Table 11), while three and four visits both regress to similar levels, which we attribute to the narrower models that matching requires. We therefore fix the loop count at 2 for all scale-up experiments. The three ablations together lock the recipe we call **SMELT**, a **S**parse **M**o**E** Transformer whose middle layers **L**oop **T**wice, through three rules: *1) loop the middle half* of layers rather than the full stack (Section 2.3), *2) give the looped model a larger effective depth-to-width ratio* than the Baseline (Section 2.4), and *3) loop twice* rather than three or four times.

3 SCALING LAWS AND COMPUTE SAVINGS

The three ablations in Section 2 locked the SMELT recipe, but they all ran at the 200M scale (up to 3.9B non-embedding parameters). Does the advantage survive when models grow to tens of billions of parameters? This section scales the recipe across four sizes up to 54B non-embedding parameters, fits a separate scaling surface for each architecture, and quantifies how much compute SMELT saves.

3.1 THE SMELT RECIPE SCALES ACROSS THE GRID

We apply the budget matching protocol and the locked recipe from Sections 2.1–2.5 to the full 4×4 grid: four scales (100M / 200M / 600M / 1.6B) crossed with four sparsity levels, S=0% and $S \approx \{85\%, 95\%, 97\%\}$. Each cell trains a matched Baseline / SMELT pair under the shared warmup-stable-decay (WSD) schedule (Appendix F.1). The six cosine-decay branches forked from each stable run yield six token horizons, which provide the variation along the data axis needed to fit the data term of the scaling law (Section 3.2). Forking decay branches from one stable run is cheaper than training a separate run per horizon, and annealed checkpoints are the ones that reflect converged loss (Hu et al., 2024; Tissue et al., 2025). That gives 32 runs, $16 \times 6 = 96$ matched Baseline / SMELT pairs, and 192 evaluation endpoints. Appendix G shows the training curves for all 16 cells, and SMELT ends below the Baseline in every one.

Figure 3 plots validation loss against cumulative training FLOPs. Each panel is one sparsity level; colors mark the four scales. Open circles mark Baseline endpoints; filled squares mark SMELT.

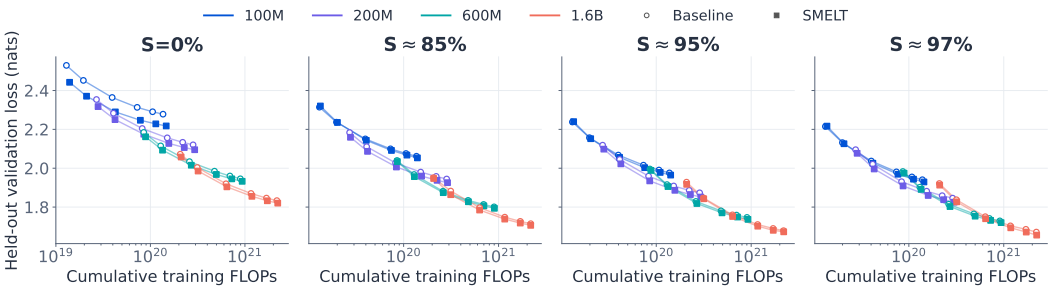


Figure 3: Validation loss vs. cumulative training FLOPs, faceted by compute-equivalent sparsity S and colored by scale. Open circles = Baseline; filled squares = SMELT. At comparable measured compute, SMELT reaches lower loss in every cell.

3.2 Scaling form and compute efficiency gain

Scaling form. Following Kaplan et al. (2020) and Hoffmann et al. (2022), we model loss as the additive form $\mathcal{L}(N, D) = E + AN^{-a} + KD^{-c}$, with an irreducible term, a capacity term, and a data term. Because total parameters N alone cannot place a dense and a sparse model on one axis, prior MoE scaling laws extend the capacity term with a routing variable such as expert count (Clark et al., 2022), expert granularity (Ludziejewski et al., 2024), or sparsity (Abnar et al., 2025). We follow that line, replacing N with per-token FLOPs F and scaling the capacity term by compute-equivalent sparsity S from Eq. 1:

$$\mathcal{L}(F, S, D) \;=\; E \;+\; \frac{A\,(1-S)^b}{F^a} \;+\; \frac{K}{D^c}. \tag{2}$$

F is the measured training FLOPs per token at the mean document length within the packed 4096-token contexts (since segment-level masks restrict attention to each document); it includes the attention cost that scales with that length and therefore exceeds the parameter-only $6N$ approximation. E is the irreducible loss. The middle term is capacity-limited: more per-token FLOPs F shrink it, and so does higher sparsity (lower $1-S$), because a sparser model stores more total parameters for the same per-token cost. The last term is data-limited: more training tokens D reduce it. We fit this form separately for the Baseline and for SMELT, giving each its own six coefficients (E, A, K, a, b, c). There is no shared loop indicator: all six coefficients may differ.

Compute-optimal frontier. Given a total training compute budget C and a sparsity level S, we want to split C between per-token FLOPs F and training tokens $D = C/F$ to minimize loss. Following Chinchilla Approach 3 (Hoffmann et al., 2022), we minimize $\mathcal{L}(F, S, D)$ subject to $C = F \cdot D$. The capacity term and the data term pull in opposite directions: a larger F shrinks the first but leaves fewer tokens $D = C/F$ for the second. Balancing the two gives compute-optimal per-token FLOPs F^* and tokens $D^* = C/F^*$, both power laws in C as in Chinchilla (Appendix C.3 gives F^* explicitly). Substituting back, the lowest achievable loss at budget C is

$$\mathcal{L}^*(C, S) \;=\; E \;+\; \underbrace{f(A, K, a, b, c, S)}_{\text{prefactor}}\; C^{-\gamma}, \qquad \gamma \;=\; \frac{a\,c}{a+c}. \tag{3}$$

The prefactor depends on (A, K, a, b, c, S), and γ sets how fast the reducible term falls with C.

Compute efficiency gain. When two architectures are fitted with separate surfaces, they trace different frontiers. Holding performance fixed and asking how much compute an improvement saves is the standard lens on algorithmic progress (Hernandez & Brown, 2020). We define the *compute efficiency gain* (CE Gain) of a target architecture over a reference architecture as the fraction of compute the target saves at a common loss. We evaluate the target's frontier loss at budget C_{tgt} and invert the reference's frontier to find the budget C_{ref} at which it reaches the same loss, which gives

$$\text{CE Gain} \;=\; 1 - \frac{C_{\text{tgt}}}{C_{\text{ref}}}. \tag{4}$$

A positive value means the target reaches the same loss with less compute than the reference. Work on algorithmic progress measures the same quantity as a multiplier $C_{\text{ref}}/C_{\text{tgt}}$ (Davidson et al., 2023; Ho et al., 2024). In our case the target is SMELT and the reference is the Baseline.

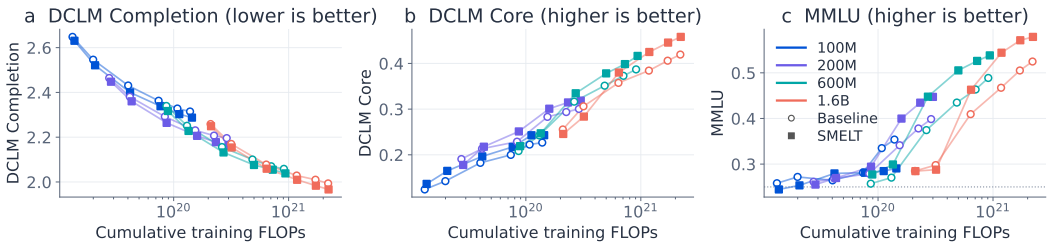


Figure 4: Downstream benchmarks vs. cumulative training FLOPs at $S \approx 97\%$. DCLM Completion is a loss; DCLM Core and MMLU are accuracies, and the dotted line marks MMLU chance (25%).

3.3 SMELT SAVES TRAINING COMPUTE ON THE FRONTIER

Fitted coefficients. We fit Eq. 2 separately for the Baseline and for SMELT with an identical procedure on the 144 endpoints of the sparse grid $S \approx \{85\%, 95\%, 97\%\}$, which span 1.3×10^{19} to 2.2×10^{21} training FLOPs (Appendix C.4). Table 3 lists the six coefficients of each architecture.

Table 3: Separate-surface coefficients on $S \approx \{85\%, 95\%, 97\%\}$, rounded to four decimal places (A and K to four significant figures in scientific notation). RMSE is in nats on validation loss itself, not on log-loss; the fit objective is Huber loss on log-loss.

Arch	E	A	K	a	b	c	RMSE
Baseline	1.4439	1.366×10^3	1.975×10^6	0.3703	0.1530	0.6594	0.00554
SMELT	1.4493	1.963×10^3	5.264×10^6	0.3892	0.1460	0.7011	0.00952

The exponents a and c set how fast loss drops with model size and data. SMELT has the larger capacity exponent a (0.3892 vs. 0.3703) and the larger data exponent c (0.7011 vs. 0.6594). Through Eq. 3, they give SMELT a 5.5% larger frontier exponent ($\gamma_{\text{SMELT}} = 0.250$ vs. $\gamma_{\text{base}} = 0.237$), so its frontier loss falls faster with compute. Appendix C.2 shows the same trend in every cell.

CE Gain on the frontier. We turn the steeper frontier into compute savings by evaluating the CE Gain (Eq. 4) at three budgets in Table 4. SMELT saves 6.8–10.0% of training compute at $C = 10^{20}$, and the saving grows to 14.7–18.0% at $C = 10^{21}$. Broken down by validation domain, the saving is positive in all five categories and largest on Code (Appendix C.1).

Table 4: CE Gain on the compute-optimal frontier: the fraction of training compute SMELT saves at the same loss. Brackets are cell-bootstrap 95% intervals (2,000 draws; Appendix C.4).

SMELT budget C_{tgt}	$S \approx 85\%$	$S \approx 95\%$	$S \approx 97\%$
10^{20} FLOPs	10.0% [1, 22]	7.8% [3, 15]	6.8% [4, 14]
10^{21} FLOPs	18.0% [8, 28]	15.8% [10, 25]	14.7% [8, 25]
10^{22} FLOPs†	23.5% [8, 42]	20.9% [0, 48]	19.6% [0, 51]

† Extrapolated beyond the fitted compute window.

4 BEYOND AGGREGATE VALIDATION LOSS

Section 3 measures the advantage on validation loss, but lower validation loss does not guarantee better downstream performance. This section first tests whether the advantage transfers to independent benchmarks and then how it varies with sample length and the number of in-context examples.

4.1 DOWNSTREAM PERFORMANCE AMPLIFIES THE VALIDATION LOSS GAP

SMELT wins on DCLM Completion in all 96 matched pairs, on DCLM Core in 83 of 96, and on MMLU in 29 of the 30 pairs whose Baseline scores at least 10 percentage points above chance, where pairwise comparisons are stable. Figure 4 shows the three benchmarks at $S \approx 97\%$, and Appendix D.1 shows every sparsity level.

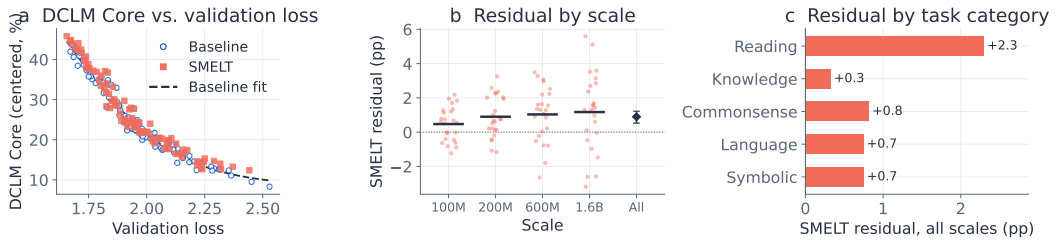


Figure 5: DCLM Core against validation loss, in centered accuracy (the DCLM-v2 reference baseline maps to zero). (a) Each point is one endpoint, and the dashed curve is Eq. 5 fitted to the 96 Baseline endpoints. (b) Residual of each SMELT endpoint against that curve (Eq. 6), grouped by scale. Bars mark per-scale means, and the "All" marker shows the pooled mean with its cell-bootstrap 95% interval. (c) Residual pooled over all scales for each DCLM task category, after fitting the same sigmoid separately to each of the 22 tasks.

Residual analysis against the Baseline curve. A model with lower validation loss should score better on any benchmark that tracks validation loss, so these win rates restate Section 3 rather than add to it. The key question is whether the downstream improvement is proportional to the validation loss reduction or exceeds it. To test this, we fit a calibration from validation loss ℓ to each of the three aggregate scores y (DCLM Core, DCLM Completion, and MMLU), using all 96 Baseline endpoints as training data, and present DCLM Core here because it is the broadest, averaging 22 tasks. Following prior task-scaling work that models loss-to-accuracy mappings with a sigmoid (Bhagia et al., 2025), we use a four-parameter monotone sigmoid:

$$\hat{y}(\ell) \;=\; b + \frac{a}{1 + \exp[\alpha(\ell - \tau)]}, \qquad a, \alpha > 0. \tag{5}$$

The DCLM Core fit is tight ($R^2 = 0.974$). For each SMELT endpoint i with validation loss ℓ_i and measured score y_i, we define the residual

$$\delta_i \;=\; y_i \;-\; \hat{y}(\ell_i), \tag{6}$$

so a positive δ_i means SMELT outperforms a Baseline that reaches the same validation loss.

The residual is positive and grows with scale. In Figure 5(a), SMELT lies above the Baseline curve. Its mean residual is positive at every scale and grows from 0.5 pp at 100M to 1.2 pp at 1.6B (Figure 5(b)). The residual is also positive in every DCLM task category and largest on Reading Comprehension (Figure 5(c)), and Appendix D.3 splits it by scale. The calibrations for DCLM Completion and MMLU also give a positive residual at every scale (Appendix D.2). SMELT's downstream advantage therefore exceeds what its validation loss improvement accounts for, and this excess grows as models scale up.

4.2 Gain grows with sample length and in-context examples

We expect the second pass through the looped layers to matter most when the input offers more context to build on. We test this along sample length and the number of in-context examples.

SMELT's gain concentrates on long samples. We group validation samples into eight length buckets by each document's token count (upper bounds 32, 64, …, 4096 tokens) and measure SMELT's per-token loss improvement over its matched Baseline in each bucket. To compare the shape of this profile rather than its size, we normalize each pair's per-bucket improvement to mean 1. SMELT's gain is larger on long samples, where its mean over the four longest buckets (512–4096 tokens) is 1.52× its mean over the four shortest (32–256 tokens) (Figure 6(a)). To check whether any added capacity would do the same, we apply the same analysis to two Baseline-only controls, one that adds experts by moving up one sparsity level at fixed scale and one that adds active parameters by moving up one scale at fixed sparsity. Neither control shows the tilt, with long-to-short ratios of 0.98 for more parameters and 0.88 for more experts. The long-sample gain is therefore specific to looping rather than to added capacity in general.

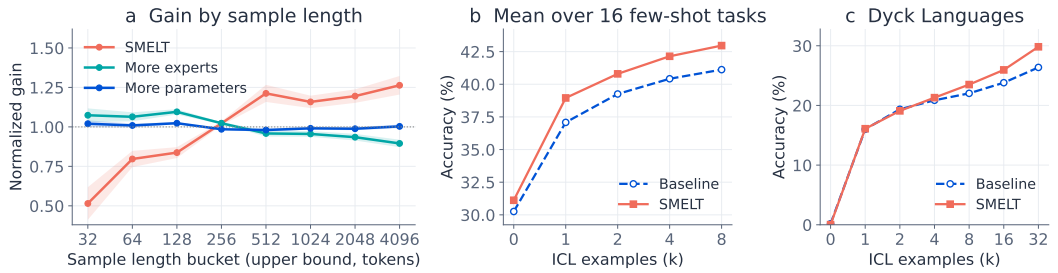


Figure 6: SMELT's extra gain concentrates on long samples and on in-context learning. (a) Loss improvement by document length, normalized to mean 1 within each contrast, for SMELT against its matched Baseline and for two Baseline-only controls that add experts or active parameters. (b) Mean accuracy over 16 few-shot tasks against the number of in-context examples k. (c) Accuracy on Dyck Languages, a demonstration-sensitive task, up to $k = 32$.

SMELT benefits more from in-context examples. Figure 6(b) averages accuracy over 16 few-shot tasks, the 15 DCLM tasks that take demonstrations plus MMLU, as the number of in-context examples k grows from 0 to 8. The gap is 0.9 pp at $k = 0$, widens to 1.9 pp once a single demonstration is provided, and then holds through $k = 8$. Dyck Languages shows the effect more sharply (Figure 6(c)), because both architectures score near 0% without demonstrations and can improve only by reading the example answers. On this task, which we extend to $k = 32$, the gap opens at larger k and reaches 29.8% against 26.4% (Appendix D.4). In the Dyck head, whose ablation raises Dyck perplexity the most, the second visit moves attention mass from the BOS sink onto the demonstration answers, cutting average BOS mass from 0.60 to 0.02 (Figure 7). Appendix B.5 shows the same sink reduction across heads on general held-out data, which suggests that freeing the sink is how the second pass uses the extra context that long samples and demonstrations provide.

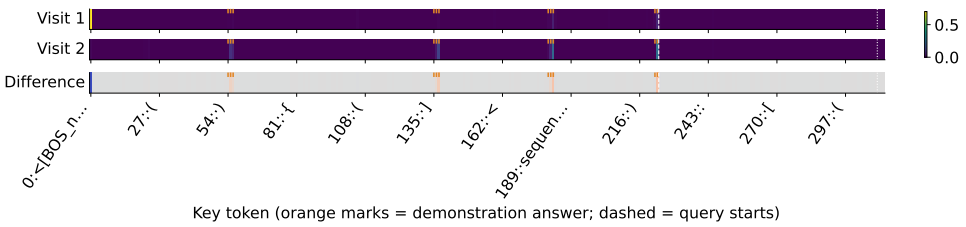


Figure 7: Attention of the Dyck head at 1.6B ($S \approx 85\%$, Dyck $k = 4$) over the prompt tokens, for the second of three prediction targets (Appendix B.4 shows all three). In the difference strip, blue marks lost mass and red marks gained mass, and orange ticks mark the demonstration answers. The second visit moves mass from the BOS attention sink onto the demonstration answers.

5 Conclusion

We demonstrate that depth reuse is a net win for MoE Transformers under budget matching. By searching the loop configuration space under budget matching, we identify the SMELT recipe, which loops the middle half of layers twice at a larger effective depth-to-width ratio and consistently beats the Baseline across four scales and four sparsity levels, saving 14.7–18.0% of training compute on the frontier at 10^{21} FLOPs. The advantage is largest on Code among validation domains and on long samples, and on downstream benchmarks it exceeds what validation loss predicts and grows with in-context examples. Our mechanistic probes find a consistent internal signature: the second visit reuses retrieval coordinates, amplifies residual writes, and frees attention mass from the sink. This suggests that looping acts as a refinement step rather than simply adding capacity. Appendix E discusses the limitations of this study and directions for future work.

Taken together, our results suggest that weight sharing across depth is a viable axis for improving language models at matched compute, complementary to the established axes of scaling width, depth, and expert count. We look forward to seeing how far this axis can be pushed as compute budgets and model scales continue to grow.

AI USE STATEMENT

Generative AI tools assisted with manuscript formatting, drafting and editing text, and writing analysis and visualization code. The authors reviewed all AI-assisted work and take responsibility for the final content, including text and artifacts produced with generative AI assistance.

ETHICS STATEMENT

This work studies pretraining architectures on an internal text corpus and involves no human subjects, no new data collection, and no model deployment.

REPRODUCIBILITY STATEMENT

Section 2.1 specifies the training protocol (architecture, key definitions, and metrics); Appendix F provides full optimizer, data, and evaluation details; Section 2.2 derives the budget-matching procedure; Appendix F.4 lists every model configuration. The scaling-law fit procedure is in Section 3.3 and Appendix C.4; the downstream evaluation protocol is in Appendix F.3; training curves for every grid cell are in Appendix G. The pretraining corpus is proprietary, but its token budget, domain structure, and contamination filtering are reported in Section 2.1 and Appendix F, and all benchmarks are public.

REFERENCES

Samira Abnar, Harshay Shah, Dan Busbridge, Alaaeldin El-Nouby, Joshua M. Susskind, and Vimal Thilak. Parameters vs FLOPs: Scaling laws for optimal sparsity for mixture-of-experts language models. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 204–230, 2025. URL https://proceedings.mlr.press/v267/abnar25a.html.

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 the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 4895–4901. Association for Computational Linguistics, 2023. URL https://doi.org/10.18653/v1/2023.emnlp-main.298.

Ibrahim Alabdulmohsin and Xiaohua Zhai. Recursive inference scaling: A winning path to scalable inference in language and multimodal systems. In *Advances in Neural Information Processing Systems*, volume 38, 2025. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/9cf64a372841aee60c059923e3289048-Abstract-Conference.html.

Zeyuan Allen-Zhu and Yuanzhi Li. Physics of language models: Part 3.3, knowledge capacity scaling laws. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=FxNNiUgtfa.

Sangmin Bae, Adam Fisch, Hrayr Harutyunyan, Ziwei Ji, Seungyeon Kim, and Tal Schuster. Relaxed recursive transformers: Effective parameter sharing with layer-wise LoRA. In *International Conference on Learning Representations*, 2025a. URL https://openreview.net/forum?id=WwpYSOkkCt.

Sangmin Bae, Yujin Kim, Reza Bayat, Sungnyun Kim, Jiyoun Ha, Tal Schuster, Adam Fisch, Hrayr Harutyunyan, Ziwei Ji, Aaron C. Courville, and Se-Young Yun. Mixture-of-recursions: Learning dynamic recursive depths for adaptive token-level computation. In *Advances in Neural Information Processing Systems*, volume 38, 2025b. URL http://papers.nips.cc/paper_files/paper/2025/hash/8b08bbf8b420faa6eeb4020720582ec7-Abstract-Conference.html.

Federico Barbero, Álvaro Arroyo, Xiangming Gu, Christos Perivolaropoulos, Michael M. Bronstein, Petar Veličković, and Razvan Pascanu. Why do LLMs attend to the first token? In *Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=tu4dFUsW5z.

Akshita Bhagia, Jiacheng Liu, Alexander Wettig, David Heineman, Oyvind Tafjord, Ananya Harsh Jha, Luca Soldaini, Noah A. Smith, Dirk Groeneveld, Pang Wei Koh, Jesse Dodge, and Hannaneh Hajishirzi. Establishing task scaling laws via compute-efficient model ladders. In *Second Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=FeAM2RVO8l.

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

Ziyang Cai, Xingyu Zhu, Yihe Dong, Yinghui He, and Sanjeev Arora. T2MLR: Transformer with temporal middle-layer recurrence. In *Latent and Implicit Thinking Workshop at ICLR*, 2026. URL https://openreview.net/forum?id=fQbk1EQWBO.

Wenkai Chen, Tianshu Li, Wenyong Huang, Yichun Yin, Lifeng Shang, and Chengwei Qin. LoopMoE: Unifying iterative computation with mixture-of-experts for language modeling. *arXiv preprint arXiv:2606.04438*, 2026a. URL https://arxiv.org/abs/2606.04438.

Yilong Chen, Naibin Gu, Junyuan Shang, Zhenyu Zhang, Yuchen Feng, Jiawei Sheng, Tingwen Liu, Shuohuan Wang, Yu Sun, Hua Wu, and Haifeng Wang. Mixture of universal experts: Scaling virtual width via depth-width transformation. *arXiv preprint arXiv:2603.04971*, 2026b. URL https://arxiv.org/abs/2603.04971.

Aidan Clark, Diego de Las Casas, Aurelia Guy, Arthur Mensch, Michela Paganini, Jordan Hoffmann, Bogdan Damoc, Blake Hechtman, Trevor Cai, Sebastian Borgeaud, George van den Driessche, Eliza Rutherford, Tom Hennigan, Matthew Johnson, Albin Cassirer, Chris Jones, Elena Buchatskaya, David Budden, Laurent Sifre, Simon Osindero, Oriol Vinyals, Marc'Aurelio Ranzato, Jack Rae, Erich Elsen, Koray Kavukcuoglu, and Karen Simonyan. Unified scaling laws for routed language models. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 4057–4086, 2022. URL https://proceedings.mlr.press/v162/clark22a.html.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. BoolQ: Exploring the surprising difficulty of natural yes/no questions. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 2924–2936, 2019. doi: 10.18653/v1/N19-1300. URL https://aclanthology.org/N19-1300.

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

Róbert Csordás, Kazuki Irie, Jürgen Schmidhuber, Christopher Potts, and Christopher D. Manning. MoEUT: Mixture-of-experts universal transformers. In *Advances in Neural Information Processing Systems*, volume 37, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/321387ba926b8e58d3591c0aeb52ffc2-Abstract-Conference.html.

Timothée Darcet, Maxime Oquab, Julien Mairal, and Piotr Bojanowski. Vision transformers need registers. In *International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=2dnO3LLiJ1.

Tom Davidson, Jean-Stanislas Denain, Pablo Villalobos, and Guillem Bas. AI capabilities can be significantly improved without expensive retraining. *arXiv preprint arXiv:2312.07413*, 2023. URL https://arxiv.org/abs/2312.07413.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Lukasz Kaiser. Universal transformers. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=HyzdRiR9Y7.

Maha Elbayad, Jiatao Gu, Edouard Grave, and Michael Auli. Depth-adaptive transformer. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=SJg7KhVKPH.

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. URL https://jmlr.org/papers/v23/21-0998.html.

Tianyu Fu, Yichen You, Zekai Chen, Guohao Dai, Huazhong Yang, and Yu Wang. Think-at-hard: Dynamic looped transformers for improved reasoning. *arXiv preprint arXiv:2511.08577*, 2025. URL https://arxiv.org/abs/2511.08577. Accepted by ICML 2026.

Zitian Gao, Yilong Chen, Yihao Xiao, Xinyu Yang, Ran Tao, Joey Zhou, and Bryan Dai. Loop the Loopies! *arXiv preprint arXiv:2607.16051*, 2026. URL https://arxiv.org/abs/2607.16051.

Jonas Geiping, Sean McLeish, Neel Jain, John Kirchenbauer, Siddharth Singh, Brian R. Bartoldson, Bhavya Kailkhura, Abhinav Bhatele, and Tom Goldstein. Scaling up test-time compute with latent reasoning: A recurrent depth approach. In *Advances in Neural Information Processing Systems*, volume 38, 2025. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/3b01972cf31e6fa0fe29e4b8b5c2a0a1-Abstract-Conference.html.

Xiangming Gu, Tianyu Pang, Chao Du, Qian Liu, Fengzhuo Zhang, Cunxiao Du, Ye Wang, and Min Lin. When attention sink emerges in language models: An empirical view. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=78Nn4QJTEN.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=d7KBjmI3GmQ.

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, Tamay Besiroglu, Ege Erdil, Zifan Carl Guo, David Owen, Robi Rahman, David Atkinson, Neil Thompson, and Jaime Sevilla. Algorithmic progress in language models. In *Advances in Neural Information Processing Systems*, volume 37, 2024. URL http://papers.nips.cc/paper_files/paper/2024/hash/6b066da6a23bc55f9b887e7298102884-Abstract-Conference.html.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Jack W. Rae, Oriol Vinyals, and Laurent Sifre. Training compute-optimal large language models. In *Advances in Neural Information Processing Systems*, volume 35, pp. 30016–30030, 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/hash/c1e2faff6f588870935f114ebe04a3e5-Abstract-Conference.html.

Shengding Hu, Yuge Tu, Xu Han, Chaoqun He, Ganqu Cui, Xiang Long, Zhi Zheng, Yewei Fang, Yuxiang Huang, Weilin Zhao, Xinrong Zhang, Zheng Leng Thai, Kaihuo Zhang, Chongyi Wang, Yuan Yao, Chenyang Zhao, Jie Zhou, Jie Cai, Zhongwu Zhai, Ning Ding, Chao Jia, Guoyang Zeng, Dahai Li, Zhiyuan Liu, and Maosong Sun. Minicpm: Unveiling the potential of small language models with scalable training strategies. *arXiv preprint arXiv:2404.06395*, 2024. URL https://arxiv.org/abs/2404.06395.

Martin Jaggi. Tying the loop – tied expert layers in mixture-of-experts language models, 2026. URL https://arxiv.org/abs/2606.16825.

Kaggle. 200,000+ Jeopardy! questions. https://www.kaggle.com/datasets/tunguz/200000-jeopardy-questions, 2019. Dataset underlying the DCLM Jeopardy task; no accompanying publication.

Ferdinand Kapl, Emmanouil Angelis, Kaitlin Maile, Johannes von Oswald, and Stefan Bauer. From growing to looping: A unified view of iterative computation in LLMs. *arXiv preprint arXiv:2602.16490*, 2026. URL https://arxiv.org/abs/2602.16490.

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.

Sanghoon Kim, Dahyun Kim, Chanjun Park, Wonsung Lee, Wonho Song, Yunsu Kim, Hyeonwoo Kim, Yungi Kim, Hyeonju Lee, Jihoo Kim, Changbae Ahn, Seonghoon Yang, Sukyung Lee, Hyunbyung Park, Gyoungjin Gim, Mikyoung Cha, Hwalsuk Lee, and Sunghun Kim. SOLAR 10.7B: Scaling large language models with simple yet effective depth up-scaling. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 6: Industry Track)*, pp. 23–35, 2024. doi: 10.18653/v1/2024.naacl-industry.3. URL https://aclanthology.org/2024.naacl-industry.3/.

Jonas Knupp, Jan Hendrik Metzen, Jeremias Bohn, Georg Groh, and Kristian Kersting. Depth-recurrent attention mixtures: Giving latent reasoning the attention it deserves. *arXiv preprint arXiv:2601.21582*, 2026. URL https://arxiv.org/abs/2601.21582.

Vedang Lad, Wes Gurnee, and Max Tegmark. The remarkable robustness of LLMs: Stages of inference? *arXiv preprint arXiv:2406.19384*, 2024. URL https://arxiv.org/abs/2406.19384.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. ALBERT: A lite BERT for self-supervised learning of language representations. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=H1eA7AEtvS.

Ryan Lee, Jacob Biloki, Edward J. Hu, and Jonathan May. Sparse layers are critical to scaling looped language models. *arXiv preprint arXiv:2605.09165*, 2026. URL https://arxiv.org/abs/2605.09165.

Hector J. Levesque, Ernest Davis, and Leora Morgenstern. The Winograd schema challenge. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Thirteenth International Conference (KR)*. AAAI Press, 2012. URL https://aaai.org/papers/59-4492-the-winograd-schema-challenge/.

Houyi Li, Ka Man Lo, Shijie Xuyang, Ziqi Wang, Wenzhen Zheng, Haocheng Zhang, Zhao Li, Shuigeng Zhou, Xiangyu Zhang, and Daxin Jiang. Mixture-of-experts can surpass dense LLMs under strictly equal resource. *arXiv preprint arXiv:2506.12119*, 2025. URL https://arxiv.org/abs/2506.12119.

Jeffrey Li, Alex Fang, Georgios Smyrnis, Maor Ivgi, Matt Jordan, Samir Gadre, Hritik Bansal, Etash Guha, Sedrick Keh, Kushal Arora, Saurabh Garg, Rui Xin, Niklas Muennighoff, Reinhard Heckel, Jean Mercat, Mayee Chen, Suchin Gururangan, Mitchell Wortsman, Alon Albalak, Yonatan Bitton, Marianna Nezhurina, Amro Abbas, Cheng-Yu Hsieh, Dhruba Ghosh, Josh Gardner, Maciej Kilian, Hanlin Zhang, Rulin Shao, Sarah Pratt, Sunny Sanyal, Gabriel Ilharco, Giannis Daras, Kalyani Marathe, Aaron Gokaslan, Jieyu Zhang, Khyathi Chandu, Thao Nguyen, Igor Vasiljevic, Sham Kakade, Shuran Song, Sujay Sanghavi, Fartash Faghri, Sewoong Oh, Luke Zettlemoyer, Kyle Lo, Alaaeldin El-Nouby, Hadi Pouransari, Alexander Toshev, Stephanie Wang, Dirk Groeneveld, Luca Soldaini, Pang Wei Koh, Jenia Jitsev, Thomas Kollar, Alexandros G. Dimakis, Yair Carmon, Achal Dave, Ludwig Schmidt, and Vaishaal Shankar. DataComp-LM: In search of the next generation of training sets for language models. In *Advances in Neural Information Processing Systems*, volume 37, 2024a. doi: 10.52202/079017-0455. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/19e4ea30dded58259665db375885e412-Abstract-Datasets_and_Benchmarks_Track.html.

Zhiyuan Li, Hong Liu, Denny Zhou, and Tengyu Ma. Chain of thought empowers transformers to solve inherently serial problems. In *International Conference on Learning Representations*, 2024b. URL https://openreview.net/forum?id=3EWTEy9MTM.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL https://openreview.net/forum?id=Bkg6RiCqY7.

Jan Ludziejewski, Jakub Krajewski, Kamil Adamczewski, Maciej Pióro, Michal Krutul, Szymon Antoniak, Kamil Ciebiera, Krystian Król, Tomasz Odrzygózdz, Piotr Sankowski, Marek Cygan, and Sebastian Jaszczur. Scaling laws for fine-grained mixture of experts. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 33270–33288, 2024. URL https://proceedings.mlr.press/v235/ludziejewski24a.html.

William Merrill and Ashish Sabharwal. The parallelism tradeoff: Limitations of log-precision transformers. *Transactions of the Association for Computational Linguistics*, 11:531–545, 2023. URL https://doi.org/10.1162/tacl_a_00562.

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

John X. Morris, Chawin Sitawarin, Chuan Guo, Narine Kokhlikyan, G. Edward Suh, Alexander M. Rush, Kamalika Chaudhuri, and Saeed Mahloujifar. How much do language models memorize? *arXiv preprint arXiv:2505.24832*, 2025. URL https://arxiv.org/abs/2505.24832.

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.

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*, volume 37, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/b6341525cd84f3be0ef203e4d7cd8556-Abstract-Conference.html.

Hayden Prairie, Zachary Novack, Taylor Berg-Kirkpatrick, and Daniel Y. Fu. Parcae: Scaling laws for stable looped language models. *arXiv preprint arXiv:2604.12946*, 2026. URL https://arxiv.org/abs/2604.12946.

Zihan Qiu, Zekun Wang, Bo Zheng, Zeyu Huang, Kaiyue Wen, Songlin Yang, Rui Men, Le Yu, Fei Huang, Suozhi Huang, Dayiheng Liu, Jingren Zhou, and Junyang Lin. Gated attention for large language models: Non-linearity, sparsity, and attention-sink-free. In *Advances in Neural Information Processing Systems*, volume 38, 2025. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/904e89bb4e632e75fb47f093b620b257-Paper-Conference.pdf.

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 *Logical Formalizations of Commonsense Reasoning, Papers from the 2011 AAAI Spring Symposium*. AAAI, 2011. URL http://www.aaai.org/ocs/index.php/SSS/SSS11/paper/view/2418.

Valeria Ruscio, Umberto Nanni, and Fabrizio Silvestri. What are you sinking? A geometric approach on attention sink. In *Advances in Neural Information Processing Systems*, volume 38, 2025. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/b2c4b7d34b3d96b9dc12f7bce424b7ae-Abstract-Conference.html.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. WinoGrande: An adversarial Winograd schema challenge at scale. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence*, pp. 8732–8740. AAAI Press, 2020. doi: 10.1609/aaai.v34i05.6399.

Nikunj Saunshi, Nishanth Dikkala, Zhiyuan Li, Sanjiv Kumar, and Sashank J. Reddi. Reasoning with latent thoughts: On the power of looped transformers. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=din0lGfZFd.

Kristian Schwethelm, Daniel Rueckert, and Georgios Kaissis. How much is one recurrence worth? iso-depth scaling laws for looped language models. *arXiv preprint arXiv:2604.21106*, 2026. URL https://arxiv.org/abs/2604.21106.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc V. Le, Geoffrey E. Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=B1ckMDqlg.

Aarohi Srivastava, Abhinav Rastogi, Abhishek Rao, Abu Awal Md Shoeb, Abubakar Abid, Adam Fisch, et al. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models. *Transactions on Machine Learning Research*, 2023. URL https://openreview.net/forum?id=uyTL5Bvosj.

Mingjie Sun, Xinlei Chen, J. Zico Kolter, and Zhuang Liu. Massive activations in large language models. *arXiv preprint arXiv:2402.17762*, 2024. URL https://arxiv.org/abs/2402.17762.

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

Ian Tenney, Dipanjan Das, and Ellie Pavlick. BERT rediscovers the classical NLP pipeline. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 4593–4601, 2019. doi: 10.18653/v1/P19-1452.

Howe Tissue, Venus Wang, and Lu Wang. Scaling law with learning rate annealing. In *Advances in Neural Information Processing Systems*, volume 38, 2025. URL http://papers.nips.cc/paper_files/paper/2025/hash/830b1abc6d2da85f23d41169fa44d185-Abstract-Conference.html.

Shaowen Wang, Bingrui Li, Ge Zhang, Wenhao Huang, Shen Yan, and Jian Li. On the residual scaling of looped transformers: Stability and transferability. *arXiv preprint arXiv:2606.18524*, 2026. URL https://arxiv.org/abs/2606.18524.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. In *International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=NG7sS51zVF.

Liu Yang, Kangwook Lee, Robert D. Nowak, and Dimitris Papailiopoulos. Looped transformers are better at learning learning algorithms. In *International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=HHbRxoDTxE.

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

Biao Zhang and Rico Sennrich. Root mean square layer normalization. In *Advances in Neural Information Processing Systems 32*, pp. 12360–12371, 2019.

Wanjun Zhong, Ruixiang Cui, Yiduo Guo, Yaobo Liang, Shuai Lu, Yanlin Wang, Amin Saied, Weizhu Chen, and Nan Duan. AGIEval: A human-centric benchmark for evaluating foundation models. In *Findings of the Association for Computational Linguistics: NAACL 2024*, pp. 2299–2314, 2024. doi: 10.18653/v1/2024.findings-naacl.149. URL https://aclanthology.org/2024.findings-naacl.149/.

Rui-Jie Zhu, Zixuan Wang, Kai Hua, Tianyu Zhang, Ziniu Li, Haoran Que, Boyi Wei, Zixin Wen, Fan Yin, He Xing, Lu Li, Jiajun Shi, Kaijing Ma, Shanda Li, Taylor Kergan, Andrew K. Smith, Xingwei Qu, Mude Hui, Bohong Wu, Qiyang Min, Hongzhi Huang, Xun Zhou, Wei Ye, Jiaheng Liu, Jian Yang, Yunfeng Shi, Chenghua Lin, Enduo Zhao, Tianle Cai, Ge Zhang, Wenhao Huang, Yoshua Bengio, and Jason K. Eshraghian. Scaling latent reasoning via looped language models. *arXiv preprint arXiv:2510.25741*, 2025. URL https://arxiv.org/abs/2510.25741.

Appendix Overview

Appendix A reviews related work. Appendix B examines what the looped layers compute on their second visit, from expert routing and residual writes to the attention sink. Appendices C and D extend the scaling and downstream results of Sections 3 and 4, and Appendix E discusses limitations and future work. Appendix F gives the experimental setup, model configurations, and full design-ablation results, and Appendix G shows the training curves of every grid cell.

A Related Work

Looped Transformers. The Universal Transformer (Dehghani et al., 2019) introduced weight sharing across depth, applying a shared block iteratively to decouple effective depth from unique parameter count. Huginn (Geiping et al., 2025) scales this idea to 3.5B parameters with a prelude–recur–coda layout (2–4–2 layers at its largest scale) that loops the middle 50%, establishing the template that most subsequent work adopts: an unshared entry block, a shared middle block that is iterated, and an unshared exit block. Ouro (Zhu et al., 2025) takes a different partition: it loops the entire L-layer stack at 1.4–2.6B. Both show that looped models match or exceed much larger unlooped models on reasoning benchmarks. A recurring finding across this body of work is that the middle layers benefit most from repetition. Saunshi et al. (2025) compare looped models against both iso-parameter and iso-FLOP baselines: looped models have worse perplexity than the iso-FLOP baseline but disproportionately improve reasoning tasks, and iterating only a middle span, the prelude-recur-coda layout above, further improves perplexity over full-stack looping. Kapl et al. (2026) report that, among the block placements tested in their setting, adapting a localized middle block for looping yields the strongest overall results. A related but untied construction appears in SOLAR (Kim et al., 2024), whose depth-up-scaling method also targets the middle layers, duplicating them to grow a pretrained model before continued pretraining. ALBERT (Lan et al., 2020) shares weights across all layers for parameter efficiency, an early instance of full cross-layer tying that predates the partial-loop designs above. Beyond the choice of which layers to loop, other work explores how to loop: Relaxed Recursive Transformers (Bae et al., 2025a) add per-layer LoRA adapters to relax strict weight tying, Mixture-of-Recursions (Bae et al., 2025b) routes each token to a learned recursion depth, and Think-at-Hard (Fu et al., 2025) learns when to iterate on hard tokens. Among these studies, Saunshi et al. (2025) additionally report an iso-FLOP comparison in which the looped model uses fewer unique parameters than the untied baseline. RINS (Alabdulmohsin & Zhai, 2025) matches total training compute by training its early-block recursive model on fewer tokens than its baseline, whereas we additionally match the token budget. Most other works above compare looped models against parameter-matched baselines, so the reported gains conflate architectural advantage with the extra FLOPs that repetition spends; whether looping helps or hurts under jointly matched compute and token budgets remains open.

Mixture-of-Experts and loop recurrence. Answering the question above requires holding three budgets fixed at once: per-token FLOPs, total parameters, and KV cache (Table 1). MoE makes this feasible by decoupling total parameters from per-token FLOPs (Shazeer et al., 2017; Fedus et al., 2022): a looped model can pay for its second visit by narrowing the hidden dimension, then recover the lost capacity by adding experts. MoEUT (Csordás et al., 2024) proposes an MoE-based Universal Transformer that, for the first time, matches standard Transformers on language modeling. Lee et al. (2026) find that sparse layers are critical to scaling looped models, because routing divergence across visits recovers expressivity lost to weight tying. Jaggi (2026) similarly reuses experts across middle layers and reinvests the saved parameters in larger expert pools. Several concurrent works train MoE Looped Transformers at scale: LoopMoE (Chen et al., 2026a) matches total parameters and per-token FLOPs at 3B and 9B; Gao et al. (2026) train 6B and 20B models under wall-clock matching and independently find two loop steps optimal, corroborating our loop-count ablation (Section 2.5). Knupp et al. (2026) (DREAMER) recurrently apply a single MoE layer 16–32 times with attention over previous depths under matched FLOPs, parameters, and memory; we view this single-layer recurrence as closer to implicit-depth models than to the block-level looping studied here. Chen et al. (2026b) share a universal expert pool across layers, converting depth into virtual MoE width rather than looping the full layer block. These concurrent studies show that looping improves MoE models. We ask a complementary question: does the advantage hold across a full scaling ladder when per-token FLOPs, total parameters, and KV cache are all matched? For block-level looping,

our comparison is the first multi-scale study that closes all three budgets simultaneously and fits a separate scaling surface for each architecture (Section 3).

Scaling laws. Kaplan et al. (2020) established power-law relations between loss, model size, and training tokens. Hoffmann et al. (2022) (Chinchilla) refined the compute-optimal allocation and showed that parameters and tokens should scale at roughly equal rates with compute. For MoE models, Clark et al. (2022) and Abnar et al. (2025) extend these laws with routing variables such as expert count and sparsity; Li et al. (2025) study the conditions under which MoE surpasses dense models at equal resources. Recent work extends these analyses to looped architectures. Schwethelm et al. (2026) fit a joint scaling law under an iso-depth protocol that holds effective depth (and therefore FLOPs and KV cache) fixed while reducing unique parameters. In their law, the effective parameter count of the recurrent block is $N_{\text{once}} + r^{0.46} N_{\text{rec}}$: r recurrent executions contribute like $r^{0.46}$ unique-block equivalents. The sublinear exponent means recurrence yields diminishing capacity per iteration, so the looped model loses at matched compute. Prairie et al. (2026) fit compute-optimal recurrence at fixed unique parameter count, but depth, inference FLOPs, and KV cache all grow with r, so the fit cannot separate the per-parameter sharing cost from the growing budgets. These two results bracket the question from opposite sides: Schwethelm et al. price looping's cost at fixed depth, Prairie et al. find looping's benefit at fixed parameters. We fit a separate Chinchilla-style surface for each architecture on MoE models while closely matching per-token FLOPs, total non-embedding parameters, and KV cache, which lets us isolate the architectural effect and measure the compute saving on the frontier (Section 3).

Attention sinks. Autoregressive Transformers concentrate attention mass on initial tokens regardless of semantic content. Xiao et al. (2024) identify this attention-sink phenomenon and exploit it for streaming inference; Gu et al. (2025) characterize when sinks emerge during pretraining; Sun et al. (2024) link them to massive activations in the residual stream, with an analogous register effect in vision models (Darcet et al., 2024); Barbero et al. (2025) argue that sinks counteract over-mixing and therefore strengthen as depth grows, a trend also observed geometrically by Ruscio et al. (2025); and Qiu et al. (2025) show that a query-dependent sigmoid gate after attention strongly mitigates the sink in the evaluated models. Our mechanistic analysis (Appendix B) adds a dynamic observation: within a single forward pass, a second weight-tied visit reduces sink mass and redirects it toward content-relevant tokens.

B Inside the Second Pass

Sections 3–4 quantify the advantage of the second visit. To build intuition for what happens inside the model, we run all matched pairs on a 1M-token held-out validation sample and probe expert routing, residual-stream updates, and attention patterns, then follow the attention sink from a Dyck-language case study to general data.

B.1 Do the two visits activate the same experts?

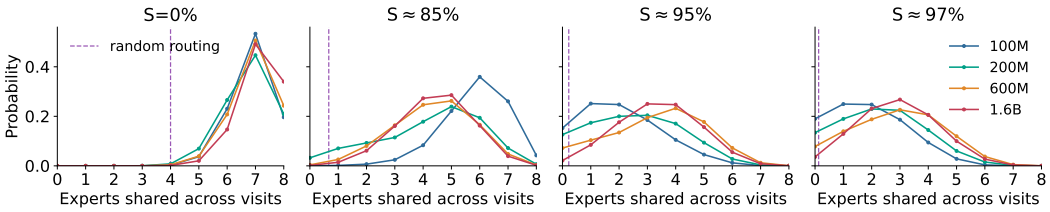


Figure 8: Experts shared between the two visits for the same token and physical gate, at the four sparsity levels. The dashed purple line marks the overlap expected under independent random routing, $8^2/n$ for a pool of n experts.

Each MoE layer selects a subset of experts for each token. When the same token passes through the same physical layer on the second visit, does the router pick the same experts, or does the updated residual-stream input lead it to a different subset?

Measurement. For each token and each repeated physical MoE layer, we record the top-8 expert sets selected by the gating network on visit 1 and visit 2, then count how many experts appear in both sets. This overlap count ranges from 0 (completely different routing) to 8 (identical routing). Figure 8 plots the distribution of this overlap count across all tokens at four sparsity levels, $S{=}0\%$ and $S \approx \{85\%, 95\%, 97\%\}$.

The router reuses a core subset and diversifies the rest. At $S{=}0\%$, where each layer has the smallest candidate pool in the grid, the router reuses nearly all 8. As sparsity increases and the pool grows by more than an order of magnitude, the overlap drops to 2–3. Even there it stays far above what independent random routing over a pool that large would give (dashed line in Figure 8), so the router deliberately reuses a core subset. The second visit reuses part of the first visit's expert set and diversifies the rest.

B.2 Does the second visit write more or less into the residual stream?

Each Transformer layer updates the residual stream as $x_{\ell+1} = x_\ell + \Delta x_\ell$. When the same physical layer executes twice, is the second write $\Delta x_\ell^{(2)}$ larger or smaller than the first $\Delta x_\ell^{(1)}$? And do the two writes point in the same direction, or does the second visit write something orthogonal?

Setup. In a pre-norm Transformer, each layer decomposes into an attention write and an MoE write (we write LN for the pre-norm; our implementation uses RMSNorm (Zhang & Sennrich, 2019)). Within the looped span, every sublayer's residual update is scaled by $1/r$ where r is the loop count (Section 2.1):

$$\Delta x_\ell^{\text{attn}} = \tfrac{1}{r}\,\text{Attn}\big(\text{LN}(x_\ell)\big), \tag{7}$$

$$\Delta x_\ell^{\text{moe}} = \tfrac{1}{r}\,\text{MoE}\big(\text{LN}(x_\ell + \Delta x_\ell^{\text{attn}})\big), \tag{8}$$

so the full layer update is $\Delta x_\ell = \Delta x_\ell^{\text{attn}} + \Delta x_\ell^{\text{moe}}$ and the MoE sub-layer sees the already-scaled attention write in its input. Outside the looped span $r = 1$, recovering the standard pre-norm form. For each of these three quantities, we measure the ℓ_2 norm on visit 1 and visit 2 and report the ratio $\|\Delta x_\ell^{(2)}\|/\|\Delta x_\ell^{(1)}\|$; a ratio above 1 means the second visit writes a larger update.

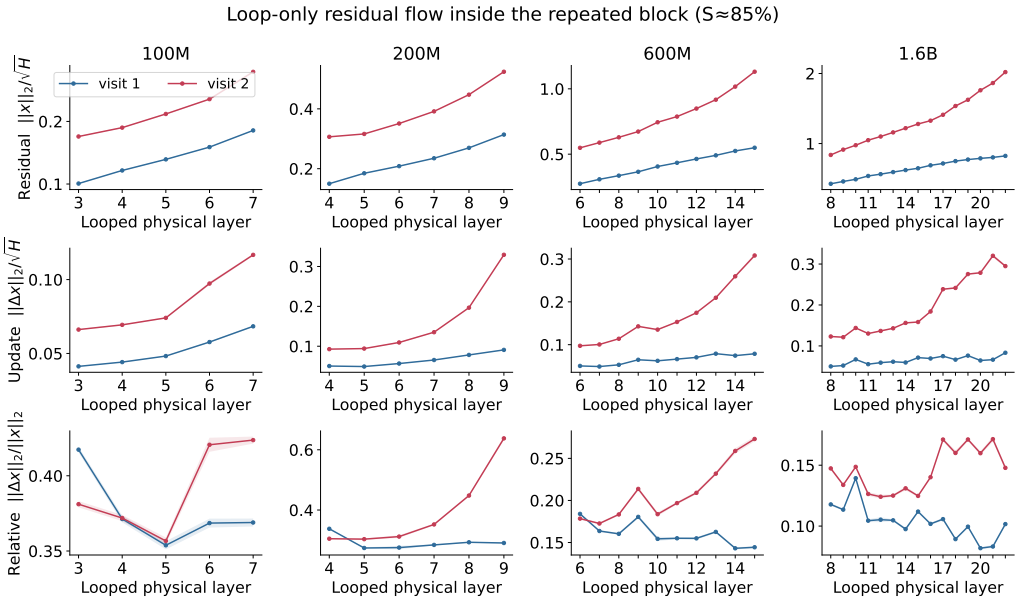


Figure 9: Residual flow inside the repeated block at $S \approx 85\%$ and four scales. The rows show the residual RMS $\|x_\ell\|_2/\sqrt{H}$, the update RMS $\|\Delta x_\ell\|_2/\sqrt{H}$, and the relative update $\|\Delta x_\ell\|_2/\|x_\ell\|_2$. Visit 2 (red) writes larger updates than visit 1 (blue).

Visit 2 writes more, both absolutely and relatively. Figure 9 shows the layer-by-layer residual flow inside the repeated block at $S \approx 85\%$ across four scales. Row 2 plots the update norm $\|\Delta x_\ell\|$: at all four scales, visit 2 (red) writes a larger update than visit 1 (blue) at every layer. Row 3 normalizes by the residual itself, plotting $\|\Delta x_\ell\|/\|x_\ell\|$. At the three larger scales the relative update is also consistently higher on visit 2. At 100M, whose repeated block spans only five layers, a few early layers are comparable or reversed, likely because the residual has not yet diverged enough between visits.

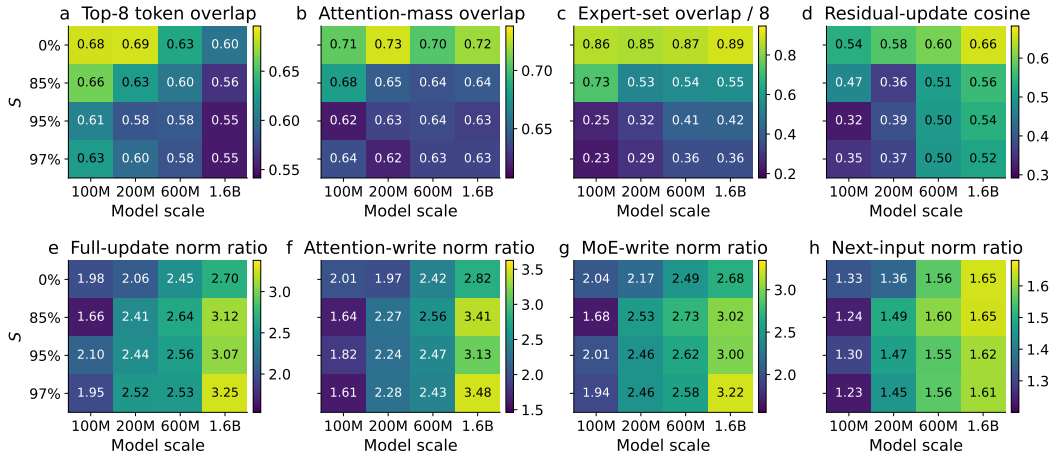


Figure 10: Eight cross-visit diagnostics over the 4×4 grid, with scale on the horizontal axis and compute-equivalent sparsity S on the vertical axis. Panels (a)–(d) measure the similarity between the two visits through top-8 attended-token overlap, attention-mass overlap, expert-set overlap divided by 8, and residual-update cosine. Panels (e)–(h) give the visit-2 / visit-1 norm ratios of the full update Δx_ℓ, the attention write $\Delta x_\ell^{\text{attn}}$, the MoE write $\Delta x_\ell^{\text{moe}}$, and the RMSNorm output that feeds the next sub-layer. All four norm ratios exceed 1 in all 16 cells.

Both attention and MoE sub-layers contribute. Where does the extra write come from, attention or MoE? Figure 10 collects eight cross-visit diagnostics over the 4×4 grid. Panels e–h each give the visit-2 / visit-1 norm ratio of one component: the full update Δx_ℓ (panel e), the attention write $\Delta x_\ell^{\text{attn}}$ (panel f), the MoE write $\Delta x_\ell^{\text{moe}}$ (panel g), and the RMSNorm output that feeds the next sub-layer (panel h). Panels a–d carry the similarity measures of Sections B.1–B.3 across the same grid, so the single-cell results in those sections can be checked at every scale and sparsity. In all 16 cells, all four norm ratios exceed 1 (range 1.2–3.5×). Both the attention and MoE sub-layers contribute: panel h shows that the RMSNorm output fed to each sub-layer already has a larger norm on visit 2. Since RMSNorm removes input scale, this difference reflects a directional shift in the residual stream: the element-wise gain g amplifies different dimensions unequally, so a change in direction alone changes the output norm. Both sub-layer writes grow on top of this larger input.

The two writes are aligned. Knowing that the second visit writes more, we next ask whether it writes in the same direction. We compute the pairwise cosine similarity between the residual updates Δx_ℓ of all executed layers and show the resulting matrix for the 1.6B model in Figure 11. The Baseline panel shows the ordinary layer-to-layer structure of an unlooped model with 30 layers, and the SMELT panel lists its 45 executed layers in flattened order. The informative entries in the SMELT panel pair each physical layer of the repeated block with itself on the other visit. Along this cross-visit diagonal, the cosine ranges from 0.42 to 0.65 with a mean of 0.56, against a mean of 0.16 for cross-visit pairs of different layers. The second visit therefore writes in a similar direction to the first at each layer: it amplifies the signal the first visit established rather than overriding it.

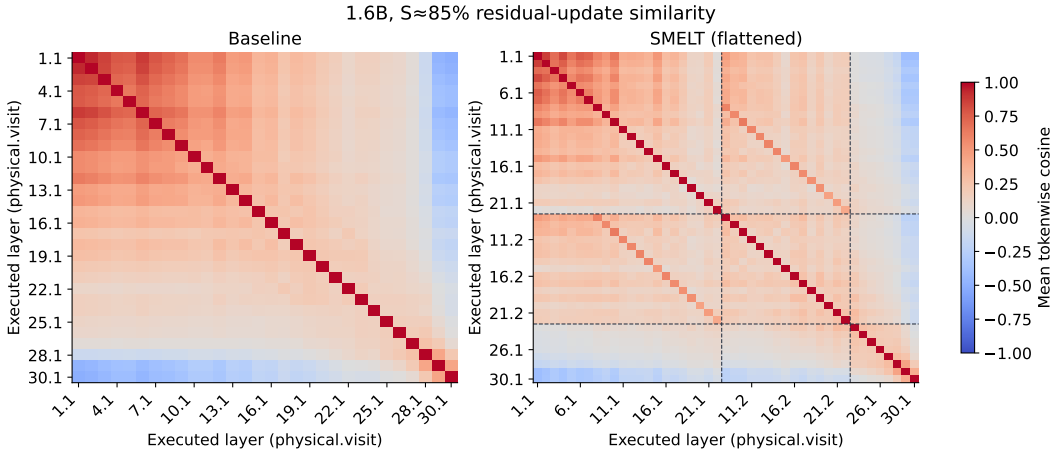


Figure 11: Pairwise residual-update cosine for the 1.6B model. The Baseline panel (left) shows its 30 layers, and the SMELT panel (right) shows its 45 executed layers in flattened order, with dashed lines marking the repeated block. The two visits of each physical layer write in closely aligned directions.

B.3 Does the second visit look at the same places or different ones?

The second visit recomputes Q, K, and V from the updated residual stream; it does not reuse cached activations from the first visit. Does this lead to different retrieval patterns (different Q and K, hence attending to different positions), or does the model preserve where it looks and only change what it reads?

Setup. Recall that the attention write from Eq. 7 decomposes as

$$\underbrace{\hat{x}_\ell}_{X} = \mathrm{LN}(x_\ell), \quad Q = \hat{x}_\ell W_Q, \quad K = \hat{x}_\ell W_K, \quad V = \hat{x}_\ell W_V, \quad P = \mathrm{softmax}\Big(\frac{QK^\top}{\sqrt{d_k}}\Big), \quad (9)$$

where Q and K set the retrieval coordinates (which positions to attend to), V carries the content at those positions, and P is the attention weight matrix. The attended context is PV and the final output written into the residual stream is $O = PV\,W_O$. For each of these six quantities (X, Q, K, V, PV, O), we compute the cosine similarity between visit 1 and visit 2 to see which stages change across visits and which stay fixed.

Q and K stay similar; V diverges. Figure 12 shows the result across the full 4×4 grid. The shared input X already has cross-visit cosine 0.84–0.93, which sets the reference level for the three projections that read it. Q and K sit at 0.89–0.93 across all 16 cells, at or above that level: the first visit establishes the retrieval coordinates and the second visit preserves them. V falls below it to 0.65–0.74, and the downstream stages PV and O diverge further (0.56–0.69 and 0.49–0.64). The divergence is therefore a property of the value projection rather than of the input it reads: the two visits retain substantially overlapping attention patterns while V changes more than Q/K.

The two visits attend to nearly the same tokens. The Q/K similarity predicts that the two visits should attend to similar positions. We verify this at the discrete level: for each head, we record the top-8 attended tokens on each visit and measure set intersection. As a control, we pair each head with a different head at the same layer and visit, which shares the same positional context but uses different learned weights; the pairing is a fixed offset of half the head count, so every head is used exactly once. Figure 13 reports the result across all heads in the repeated block. The top-8 attended tokens overlap by 56–66% across visits (Figure 13b). Selecting 8 out of up to 4,096 positions and recovering more than half the same set is a strong signal that the retrieval pattern is preserved. The other-head control sits at 28–34%, confirming that this overlap is head-specific, not an artifact of positional bias. The gap is narrower on top-1 retention (Figure 13a), where every head in a segment tends to land on the sink token and at the larger scales the control therefore starts high. Together with the Q/K–V split above, this section's picture is: the two visits retain substantially overlapping attention patterns while V changes more than Q/K.

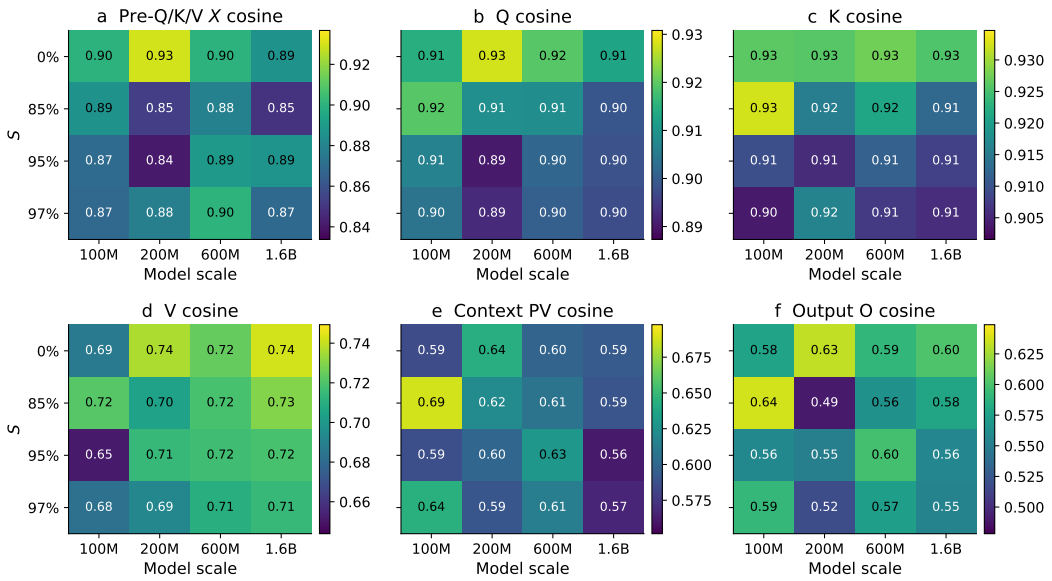


Figure 12: Cross-visit cosine similarity at six stages of attention on the full 4×4 grid, from (a) the shared pre-norm input X through (b) Q, (c) K, and (d) V to (e) the attended context PV and (f) the output O. Q and K stay at or above the similarity of their input, while V, PV, and O fall below it.

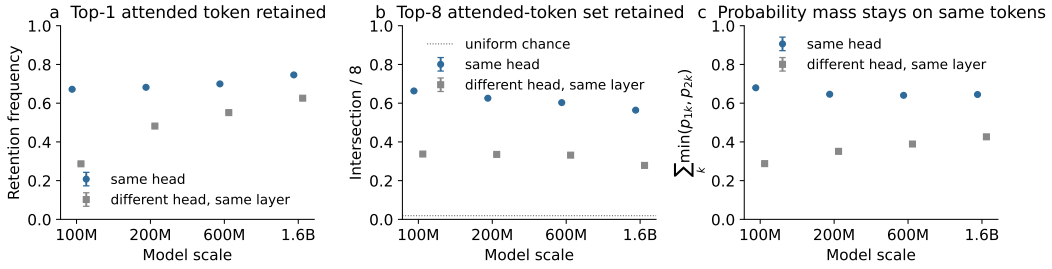


Figure 13: Attention token overlap between the two visits of the repeated block ($S \approx 85\%$, four scales): (a) top-1 retention, (b) top-8 set intersection, and (c) overlap of the full attention distributions. Blue compares the same head across visits; grey is a control that pairs each head with a different head in the same layer and visit, sharing the positional context but not the learned weights. The same head attends to far more of the same tokens across visits than the control does.

B.4 Case study: Dyck language

Section 4.2 shows that SMELT's advantage grows with the number of in-context examples. To understand why, we study Dyck-language bracket matching, a demonstration-sensitive exact-match task where the correct answer is unambiguous: to close a bracket, the model must retrieve the matching opening symbol from a demonstration in the prompt. This lets us trace, at the individual token level, whether the second visit improves retrieval of the right demonstration.

Setup. We construct nested Dyck prompts at $k \in \{1, 2, 4, 8\}$ shots, with three fixed target positions whose correct closing symbol already appears in the $k \geq 1$ prefix. Each shot presents an input bracket sequence followed by its completion; we call the completion tokens that follow its "Output:" marker the *demonstration answer*. For each scale, we identify the *Dyck head*: we ablate each attention head in the repeated block one at a time and select the head whose removal increases Dyck perplexity the most. We then visualize this head's full attention pattern to see where mass moves between the two visits.

The attention sink nearly vanishes on the second visit. Figure 14 shows the Dyck head's attention map at 1.6B for a $k = 4$ example. On visit 1, the head concentrates mass on the attention sink at the

segment-start BOS token and on nearby positions, as if the model does not yet know where to look. On visit 2, the sink nearly vanishes: mass shifts from BOS onto the demonstration answers in the prefix. The difference row makes the redistribution explicit. The four shots contain 11 demonstration-answer tokens, which the figure marks in orange. Averaged over the three targets, BOS mass falls from 0.60 to 0.02 while the demonstration answers rise from 0.24 to 0.85, so the sink loses almost exactly what the demonstrations gain.

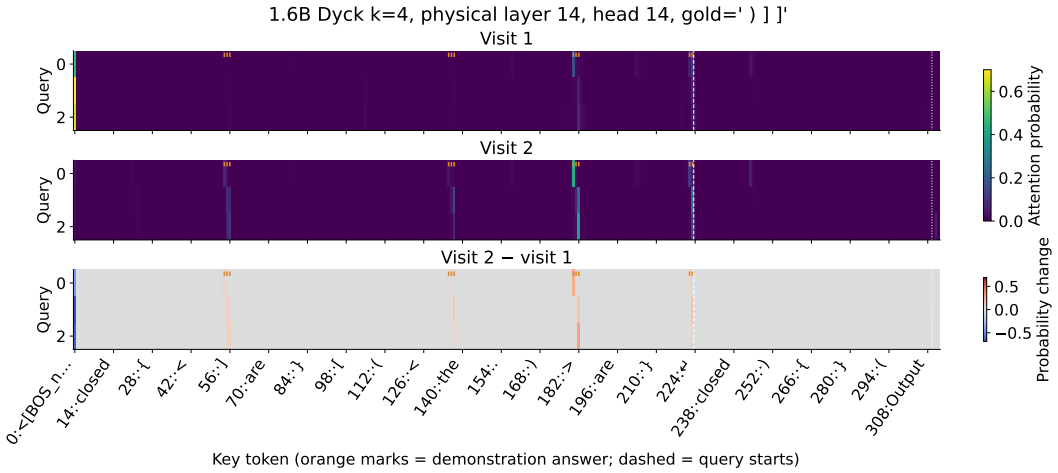


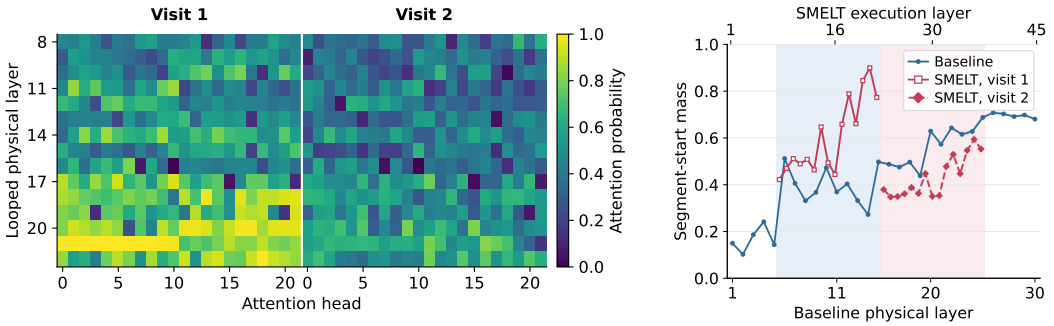
Figure 14: Token-level attention map for the *Dyck head* at 1.6B, $S \approx 85\%$, Dyck $k = 4$, for all three prediction targets. The Dyck head is the head in the repeated block whose ablation raises Dyck perplexity the most, identified separately at each scale. Rows are the three prediction targets and columns are key positions, with visit 1 on top, visit 2 in the middle, and their difference at the bottom. Orange marks are the 11 demonstration-answer tokens of the four shots, the completions that follow each shot's "Output:" marker. The second visit moves mass from the BOS sink onto the demonstration answers.

We suspect this pattern is related to the ICL advantage reported in Section 4.2: more demonstrations give the second visit more answer spans to attend to, and the sink reduction frees mass to reach them.

B.5 Does the sink reduction generalize beyond Dyck?

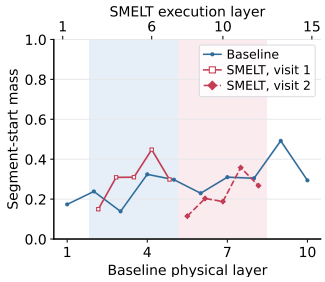
Dyck is a controlled task, so we repeat the sink measurement on the full 1M-token held-out sample spanning all five source categories. Figure 15a shows the segment-start attention mass of every head in the repeated block at 1.6B. The bright, high-mass cells of visit 1, concentrated in the later looped layers, cool broadly on visit 2.

This reduction runs against the ordinary depth trend. Figure 15b compares SMELT and the unlooped Baseline at the same physical depth. In the Baseline, sink mass grows toward later layers, consistent with the sink strengthening with depth in standard Transformers (Barbero et al., 2025; Ruscio et al., 2025). SMELT's visit 2 instead lies below visit 1 at every looped layer, so the second visit reverses the depth trend rather than inheriting the larger sink that its later position in the execution trace would predict. Figures 15c–15e repeat the comparison at the three smaller scales. At every scale, the later half of the Baseline's layers carries more sink mass on average than the earlier half, while SMELT's visit 2 stays below visit 1 at every looped layer. Averaged over the repeated block, the second visit lowers sink mass from 0.30 to 0.23 at 100M, from 0.45 to 0.31 at 200M, from 0.51 to 0.37 at 600M, and from 0.60 to 0.43 at 1.6B.

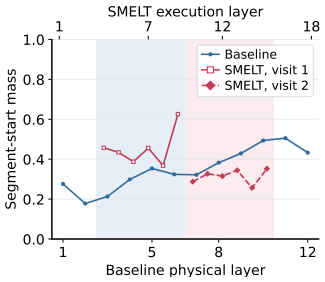


(a) Per-head mass in the repeated block, 1.6B.

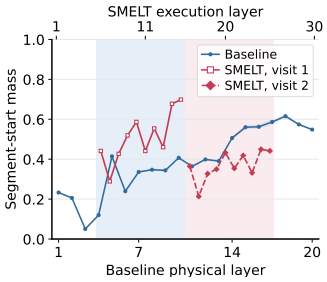
(b) Mass per layer, 1.6B.



(c) Mass per layer, 100M.



(d) Mass per layer, 200M.



(e) Mass per layer, 600M.

Figure 15: Segment-start attention mass (the attention sink) on the 1M-token held-out sample at $S \approx 85\%$. In (b)–(e), bottom axes index Baseline physical layers and top axes index SMELT execution layers. The second visit reduces the sink at every looped layer, against the Baseline's depth trend.

C Scaling Laws: Extended Results and Fit Details

This appendix extends the scaling results of Section 3. It first breaks the CE Gain down by validation domain, then checks the frontier exponent cell by cell and compares the compute-optimal allocations of the two architectures, and it closes with the fit protocol and bootstrap behind these numbers.

C.1 CE Gain by validation domain

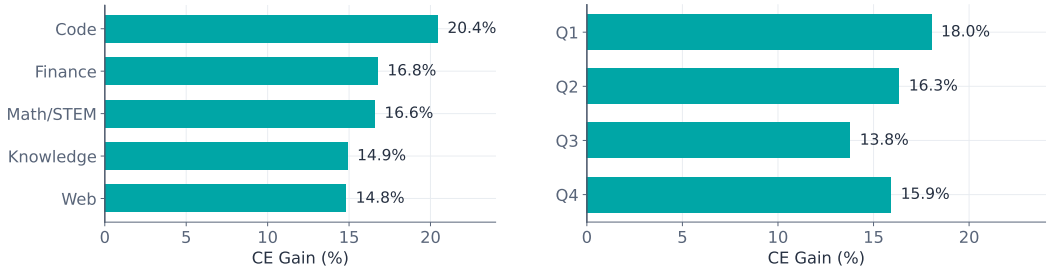
Section 3.3 reports CE Gain averaged over all validation domains, and this subsection asks whether it is uniform across data types.

Method. To compute CE Gain on a subset of the data, we need a scaling surface restricted to that subset. A single subset does not have enough data to independently fit all six coefficients in Eq. 2. We therefore share the exponents (a, b, c) from the aggregate fit (Table 3) and fit only the intercepts (E_s, A_s, K_s) per subset, separately for the Baseline and for SMELT. This lets each subset converge to its own entropy floor while borrowing curvature from the data-rich aggregate fit. We then apply the CE Gain inversion from Section 3.2 at $C = 10^{21}$ and $S \approx 95\%$. We apply this procedure under two different groupings.

By domain category. The held-out validation set spans five categories (Appendix F.2): Code, Math/STEM, Finance, Knowledge, and Web. We fit per-category intercepts and compute CE Gain for each.

By Baseline validation loss. Instead of using human-defined categories, we can also group sources by how well the Baseline already models them. Here we fit intercepts per individual source rather than per group, which gives a CE Gain for each of the 39 sources. Within each matched pair we then rank the 39 sources by the Baseline's per-source validation loss and split them into four near-equal groups (10, 10, 10, 9): Q1 holds the sources with the lowest Baseline loss (data the Baseline already predicts well) and Q4 the highest. The CE Gain of each source is fixed (determined by the per-source

intercept fit), but its quartile assignment can change from pair to pair because the ranking is based on each pair's Baseline loss; the average is taken within each quartile and then across all 96 pairs.



(a) By domain category.

(b) By Baseline-loss quartile.

Figure 16: CE Gain on the compute-optimal frontier at $C = 10^{21}$ and $S \approx 95\%$. (a) From per-category intercepts, all five categories are positive and Code leads. (b) Within each matched pair, the 39 validation sources are ranked by Baseline per-source loss and split into four near-equal groups, from Q1 (lowest loss) to Q4 (highest). Averaged per-source CE Gain is highest in Q1, and the profile is U-shaped with its minimum at Q3.

Results. CE Gain is positive on all five categories (Figure 16(a)). Code leads at 20.4%; Finance and Math/STEM follow at 16.8% and 16.6%; Knowledge and Web sit at 14.9% and 14.8%. The ranking tracks the degree of internal structure: Code has strict syntax and long-range dependencies; Web text is less constrained.

On the Baseline-loss axis (Figure 16(b)), CE Gain is not monotone: it falls from Q1 (18.0%) through Q2 (16.3%) to a minimum at Q3 (13.8%), then rebounds at Q4 (15.9%). The sources the Baseline already models well gain the most, but the hardest quartile is not the one that gains least. We read the Q1 end as the informative one: it lines up with the category breakdown, since Code has both the strongest structure and the lowest Baseline loss, and it has the highest CE Gain. Why the high-loss tail rebounds is not something our data settles; one possibility is that Q4 mixes genuinely noisy sources, where neither architecture can improve much, with hard-but-structured ones.

C.2 Reducible loss by cell

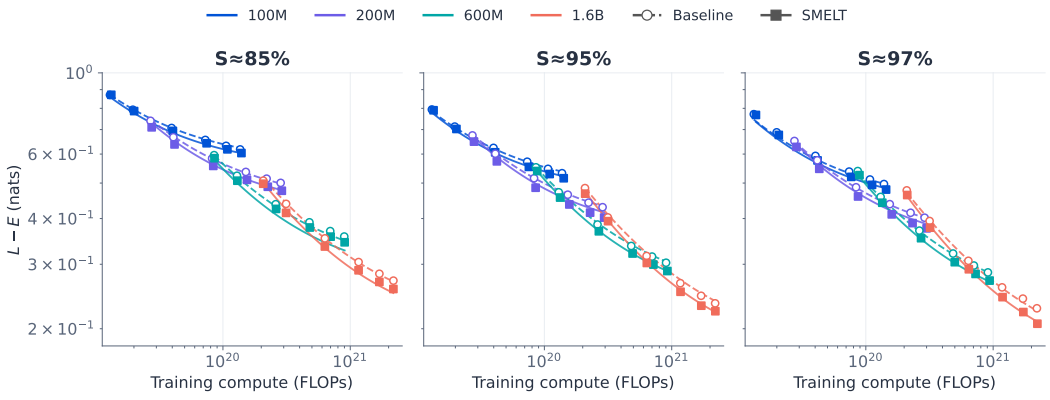


Figure 17: Reducible loss $\mathcal{L} - E$, where E is each architecture's own irreducible loss, vs. training compute on the sparse scaling-law grid $S \approx \{85\%, 95\%, 97\%\}$. The $S = 0\%$ level is omitted because its distinct regime is excluded from the sparse-grid fit. At every scale and sparsity, SMELT's reducible loss decreases faster with compute.

Section 3.3 finds a larger frontier exponent γ for SMELT than for the Baseline, and this subsection checks that every cell of the sparse grid shares the steeper decline. Figure 17 plots reducible loss $\mathcal{L} - E$ against training compute on log–log axes, subtracting each architecture's own irreducible loss E so that both curves show only the part of the loss that training can still reduce. Markers are the

six decay-branch endpoints of each cell, and each line follows the architecture's fitted surface at the cell's own per-token FLOPs and sparsity as training tokens grow. On these axes, a steeper line means that loss falls faster with compute. SMELT's line (solid) is steeper than the Baseline's (dashed) in all 12 cells, and a straight-line fit to the markers alone gives the same ordering in every cell, so the larger γ is not an artifact of the fitted form.

C.3 Compute-optimal allocation

Section 3.2 states that the compute-optimal per-token FLOPs and tokens are power laws in C. Balancing the capacity and data terms of Eq. 2 under $C = F \cdot D$ gives

$$F^* \;=\; \left(\frac{a\, A\, (1-S)^b\, C^c}{c\, K} \right)^{1/(a+c)}, \tag{10}$$

and the corresponding optimal tokens $D^* = C/F^*$. The separate surfaces therefore let us ask whether SMELT should allocate compute differently from the Baseline. From Eq. 10, the optimal per-token FLOPs scale as $F^* \propto C^{c/(a+c)}$. A larger $c/(a+c)$ means a larger share of budget goes to model size rather than to tokens. We measure this through tokens per dense-equivalent parameter (TPP $= D^*/(F^*/6)$), where $F/6$ converts per-token FLOPs to an approximate active parameter count via the standard 6 FLOPs-per-parameter-per-token ratio (Kaplan et al., 2020). Table 5 reports TPP at $C = 10^{21}$, with cell-bootstrap intervals (Appendix C.4).

Table 5: Compute-optimal tokens per dense-equivalent parameter (TPP) at $C = 10^{21}$. Brackets are cell-bootstrap 95% intervals.

S	Baseline TPP	SMELT TPP
$\approx 85\%$	56 [51, 65]	56 [44, 77]
$\approx 95\%$	78 [71, 90]	75 [60, 101]
$\approx 97\%$	91 [83, 104]	86 [70, 114]

The two architectures want nearly the same split: SMELT's TPP point estimate is within 6% of the Baseline's at every sparsity, and the bootstrap intervals overlap almost completely (Table 5). The data term in Table 3 explains why the two stay close. SMELT has both a larger data coefficient K (5.264×10^6 vs. 1.975×10^6) and a larger data exponent c (0.7011 vs. 0.6594). The larger K makes SMELT's data-limited loss K/D^c higher at small D, below the crossover at $D \approx 1.6 \times 10^{10}$ tokens. The larger c makes the term fall faster as D grows. The two effects pull the optimal split in opposite directions and roughly cancel, so SMELT's compute saving comes from reaching a lower loss at the same allocation rather than from reallocating the budget. Both architectures sit well above the $\sim$20 TPP that Chinchilla reports for dense models (Hoffmann et al., 2022); sparse MoE models generally prefer more tokens per active parameter, consistent with prior work (Abnar et al., 2025).

C.4 Fit protocol and cell bootstrap

This subsection specifies how Section 3.3 fits Eq. 2 for each architecture. Both fits use the same Huber-loss ($\delta = 10^{-3}$) minimization on log-loss with L-BFGS-B (Hoffmann et al., 2022). Keeping the procedure identical for both is important, because compute-optimal conclusions are sensitive to fitting choices (Porian et al., 2024). Not all endpoints enter the same fit. The $S = 0\%$ level lies far from the three sparse levels; including it in a joint fit worsens the surface RMSE on the sparse grid, because the optimizer compromises between two regimes. We therefore exclude $S = 0$ from the scaling-law fit. The sparse fit uses 72 Baseline endpoints and 72 SMELT endpoints from the three levels $S \approx \{85\%, 95\%, 97\%\}$. These 144 endpoints span 1.3×10^{19} to 2.2×10^{21} cumulative training FLOPs, which is the window the fitted surfaces are supported on. Note that E_{SMELT} is slightly higher than E_{base} in Table 3, but the gap (0.005 nats) is smaller than the fit RMSE of either architecture, so we attribute it to fitting noise rather than a genuine difference in asymptotic floors. Predicting frontier behavior at much larger budgets (e.g. $C = 10^{25}$) would require training runs at those scales to anchor the fit.

Cell bootstrap. The brackets in Tables 4 and 5 come from a *cell bootstrap*. We resample the twelve sparse grid cells with replacement, refit both surfaces on each resample, recompute the quantity of

interest, and report the 2.5th and 97.5th percentiles over the draws. Table 4 uses 2,000 draws, and Table 5 uses 200 draws because each draw refits both surfaces and then re-solves the allocation.

D Extended Downstream Results

This appendix collects the downstream results behind Section 4, in the order the main text uses them.

D.1 Downstream benchmarks at every sparsity level

Section 4.1 shows the three benchmarks at $S \approx 97\%$, and Figures 18–20 show them at every sparsity level.

On DCLM Completion (Figure 18), SMELT has the lower loss in all 96 matched pairs. Its lead is largest at the $S = 0\%$ level, 0.051 nats on average against 0.020–0.023 nats at the three sparse levels, and it narrows with scale as both losses fall, from 0.044 nats at 100M to 0.018 nats at 1.6B.

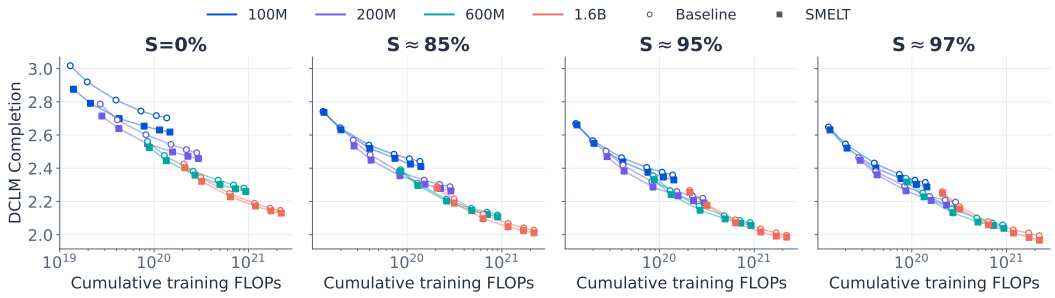


Figure 18: DCLM Completion, a gold-completion loss (lower is better), vs. cumulative training FLOPs at every compute-equivalent sparsity level. Colors mark the four scales, open circles mark Baseline endpoints, and filled squares mark SMELT.

On DCLM Core (Figure 19), SMELT scores higher in 83 of 96 pairs, including all 24 at the $S = 0\%$ level. Twelve of the 13 exceptions lie on the three earliest decay branches, forked at steps 10,000 to 50,000, and the remaining one is the longest branch at 600M and $S \approx 85\%$.

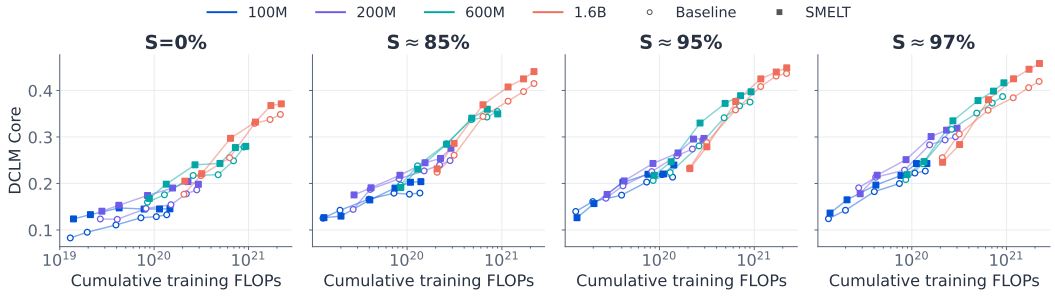


Figure 19: DCLM Core, in centered accuracy (higher is better), vs. cumulative training FLOPs at every compute-equivalent sparsity level, with markers as in Figure 18.

On MMLU (Figure 20), only 5 of the 48 Baseline endpoints below 600M score at least 10 percentage points above chance (25%), and pairwise comparisons near that floor are unstable. We therefore restrict the win-rate summary to the 30 pairs across all scales whose Baseline clears this bar, and SMELT wins 29 of them. The one exception is the longest branch at 100M and $S \approx 97\%$, the only 100M pair above the bar. At 600M and 1.6B, where MMLU is furthest from chance, SMELT scores higher in 42 of 48 pairs, by 3.1 pp on average.

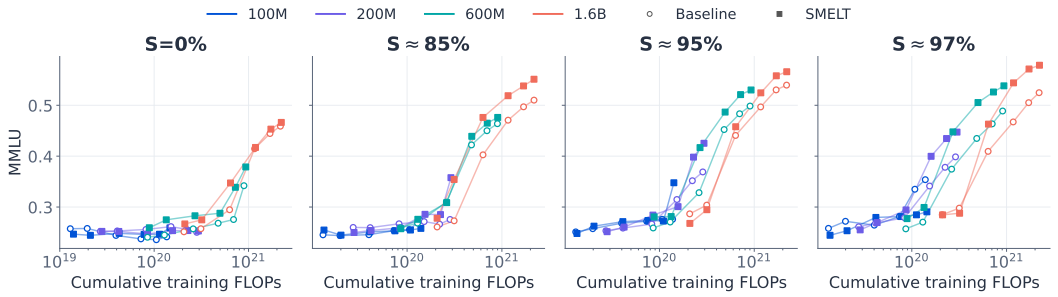


Figure 20: MMLU accuracy (higher is better) vs. cumulative training FLOPs at every compute-equivalent sparsity level, with markers as in Figure 18. The dotted line marks chance accuracy (25%).

D.2 Calibration on all three benchmarks

Section 4.1 calibrates DCLM Core against validation loss with the sigmoid in Eq. 5. We apply the same four-parameter family to DCLM Completion and MMLU, fitting a, b, α, and τ separately for each metric on the 96 Baseline endpoints. DCLM Completion is a loss that increases with validation loss, so for it we flip the sign inside the exponential to obtain an increasing sigmoid, and we flip the sign of the residual in Eq. 6 so that a positive residual still means SMELT outperforms the Baseline curve. The fits are tight: $R^2 = 0.997$ for DCLM Completion, 0.974 for DCLM Core, and 0.911 for MMLU (Figure 21). In all three panels, SMELT's endpoints lie on the favorable side of the Baseline curve, below it on DCLM Completion and above it on DCLM Core and MMLU.

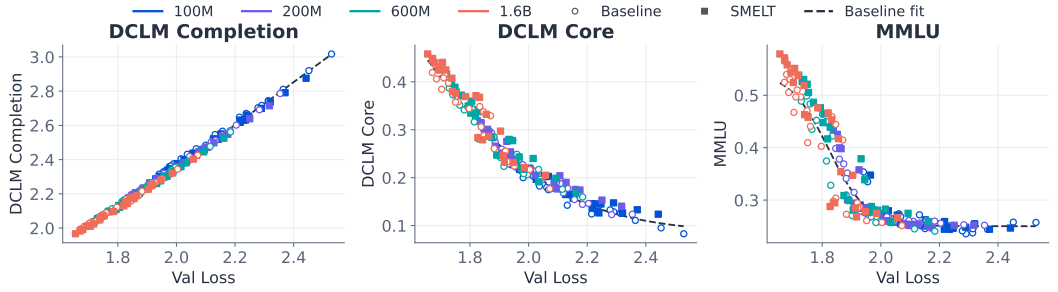
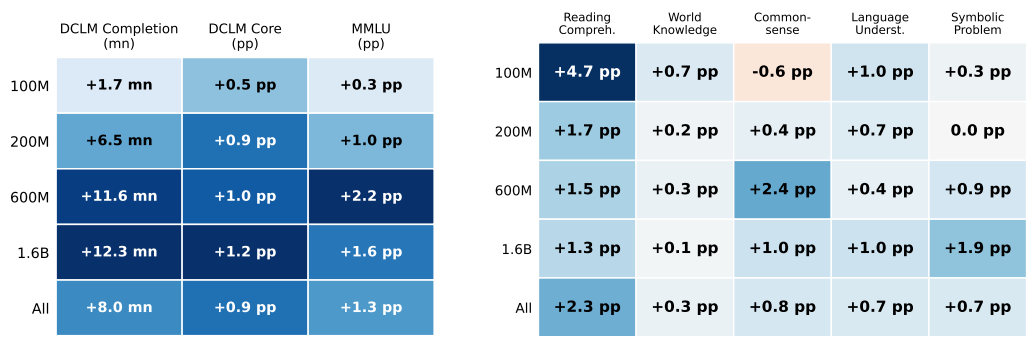


Figure 21: Benchmark scores against validation loss. DCLM Completion is a loss (lower is better); DCLM Core and MMLU are scores (higher is better). The dashed curve is the monotone sigmoid of Eq. 5 fitted to the 96 Baseline endpoints of each metric. SMELT's downstream advantage exceeds what its validation-loss improvement alone predicts.

Figure 22a breaks down the mean residual by scale. The mean residual is positive on every benchmark at every scale. It grows monotonically with scale on DCLM Completion (from 1.7 to 12.3 millinats) and DCLM Core (from 0.5 to 1.2 pp), and on MMLU the two larger scales exceed the two smaller ones.



(a) By benchmark.

(b) By DCLM task category.

Figure 22: Mean downstream residual of SMELT (Eq. 6) by model scale, (a) for the three benchmarks and (b) for the five DCLM task categories. Units are millinats (mn, 10^{-3} nats) for DCLM Completion and percentage points (pp) otherwise; DCLM Core and the task categories use centered accuracy, in which the DCLM-v2 reference baseline maps to zero, while MMLU uses raw accuracy. The "All" row pools residuals over endpoints from all four scales rather than fitting a separate row, and a positive residual means that SMELT outperforms the Baseline-fitted curve.

D.3 Residual by DCLM task category and scale

Section 4.1 reports the DCLM Core residual per task category pooled over all scales. Figure 22b splits it by scale, using the same per-task fits: the decreasing sigmoid of Eq. 5 is fitted independently to each of the 22 DCLM tasks (Appendix F.3) before the chance-centered residuals are averaged within each category. Reading Comprehension shows the largest excess when all scales are pooled (2.3 pp), driven mainly by 100M (4.7 pp). The dominant category shifts as models grow: Commonsense Reasoning leads at 600M (2.4 pp), and Symbolic Problem Solving overtakes both at 1.6B (1.9 pp). This suggests that the benefit of the second visit on symbolic problem solving grows with model capacity.

D.4 Per-task ICL accuracy curves

The shot sweep of Section 4.2 covers 16 few-shot tasks: the 15 DCLM Core tasks that take in-context demonstrations, plus MMLU. This sweep is a separate evaluation from the aggregate metrics: it reruns each task at $k \in \{0, 1, 2, 4, 8\}$ (and QA Wikidata, ARC Easy, CS Algorithms, Dyck Languages, and Operators additionally at $k = 16$ and 32) rather than at the fixed shot counts of Appendix F.3, which is why the accuracies here do not match the DCLM Core numbers elsewhere. Figure 23 shows the 14 of those 16 tasks whose accuracy gain from $k = 0$ to the largest k tested is significant ($p < 0.05$); BoolQ and LSAT AR miss this cut. Each curve averages first within each (scale, sparsity) grid cell where both architectures score above chance, then across cells, so that no single scale dominates.

SMELT ends ahead of the Baseline at the largest k on 13 of the 14 tasks, with the largest gains on SQuAD (4.2 pp), Operators (4.0 pp), and CommonsenseQA (3.8 pp). On HellaSwag, Language ID, and MMLU, the gap is already present at $k = 0$ and stays roughly constant as k grows, so SMELT's lead there does not depend on the demonstrations. On Jeopardy, Dyck Languages, and Repeat Copy, both models start near zero at $k = 0$, and the gap opens only once demonstrations are added: at $k = 1$ on Jeopardy, at $k = 4$ on Repeat Copy, and gradually from $k = 8$ to $k = 32$ on Dyck Languages. CS Algorithms, the fourth task that starts near zero, is the one exception: SMELT leads at $k = 1$ and 2 but falls behind from $k = 4$ onward and ends at 49.4% against 50.9% at $k = 32$.

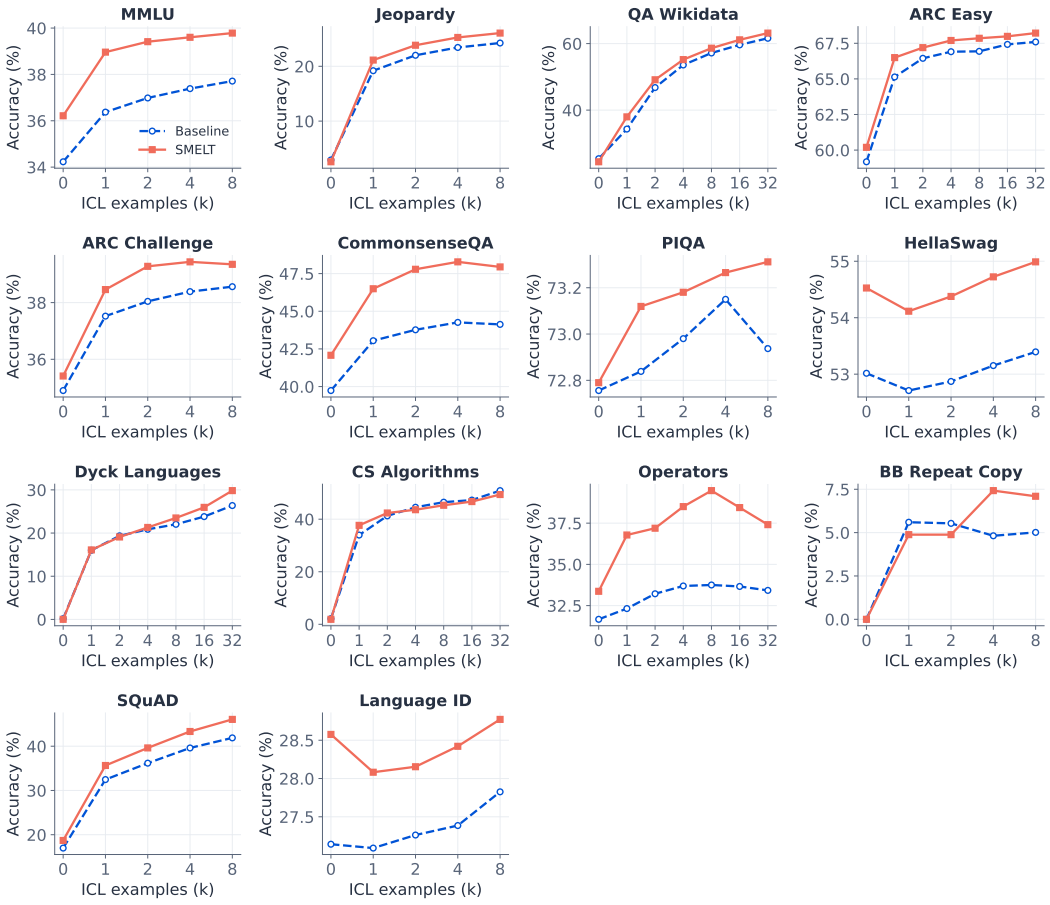


Figure 23: Per-task accuracy vs. number of in-context examples k for the 14 of 16 tasks whose accuracy rises significantly with k ($p < 0.05$). Baseline is dashed and blue, and SMELT is solid and red. Each curve averages first within each grid cell, then across cells.

E Limitations and Future Work

This appendix discusses the limitations of the study summarized in Section 5 and the directions they open. Due to compute constraints, we conduct the design ablations at the 200M scale (up to 3.9B non-embedding parameters); the optimal loop span or count may differ at larger scales. We also study only the simplest form of looping: repeating a contiguous block with fully shared weights. Richer variants relax this in different directions, including per-visit low-rank adapters (Bae et al., 2025a), token-level adaptive recursion depth (Bae et al., 2025b), learned halting (Dehghani et al., 2019; Elbayad et al., 2020), block-selective sharing (Kapl et al., 2026), and cross-token state reuse (Cai et al., 2026). Whether these designs preserve or amplify the CE Gain under budget matching is left to future work. Separately, our budget matching equates arithmetic FLOPs, parameters, and KV-cache size, not wall-clock cost; serial block re-execution and sparse routing might introduce hardware-efficiency gaps that require systems-level optimization to close. Finally, understanding exactly which mechanisms inside the second visit are responsible for the gain remains an open question; our probes in Appendix B offer a descriptive starting point, and we expect future work to build on it toward a causal account.

F Experimental Setup and Model Configurations

This appendix collects the training, data, and evaluation details that Section 2.1 defers here, the full model configurations for both architectures, and the full results of the design ablations in Section 2.

F.1 Optimizer and Schedule

We use AdamW (Loshchilov & Hutter, 2019) with a warmup-stable-decay (WSD) schedule (Hu et al., 2024). The stable phase trains at a constant learning rate over 196,075 steps with a global batch size of 256 sequences (≈1M tokens per step). From each stable run we branch six cosine-decay schedules at steps 10,000, 20,000, 50,000, 100,000, 150,000, and 196,075, each decaying over 10B additional tokens. The six branch points give six token horizons per configuration.

F.2 Data

We pretrain on an internal corpus. The stable phase of the WSD schedule consumes 205B tokens; each cosine-decay branch then trains on 10B additional tokens drawn from fresh data not seen during the stable phase, bringing the longest branch to approximately 215B tokens with no repeated data. All matched Baseline / SMELT pairs at the same scale and sparsity level are trained on identical token sequences. The held-out validation set covers 39 individual sources grouped into five categories: Code, Math/STEM, Knowledge, Finance, and Web. Training sequences are packed into 4096-token contexts with segment-level attention masks, where each segment is one document within the packed context.

F.3 Evaluation Metrics and Protocol

We score three metrics throughout. Validation loss is the token-weighted cross-entropy on the held-out validation set, averaged across all 39 sources.

Task suite. We evaluate on the 22-task DCLM Core suite (Li et al., 2024a) and, separately, on MMLU 5-shot (Hendrycks et al., 2021). MMLU is not part of the official DCLM Core set but is widely used as a standalone benchmark; we report it individually rather than folding it into our aggregate metrics. We use the v2 centered-accuracy baselines released by the DCLM project. Table 6 lists the full task–category mapping.

DCLM Core (centered accuracy). For each task t with raw accuracy a_t and DCLM-v2 reference baseline ρ_t (the "Random baseline" field in DCLM's evaluation metadata), the centered score is

$$c_t \;=\; \frac{a_t - \rho_t}{1 - \rho_t}\,, \tag{11}$$

so that 0 corresponds to the DCLM-v2 reference baseline and 1 to perfect accuracy. DCLM Core is the unweighted mean of c_t over the 22 tasks.

DCLM Completion (gold-completion micro loss). Nine of the 22 tasks ask for a free-form answer rather than a choice among options, so every item in them has a gold answer string: CoQA, SQuAD, Jeopardy, LAMBADA, BB QA Wikidata, BB CS Algorithms, BB Dyck Languages, BB Operators, and BB Repeat Copy. For each item we compute the per-token cross-entropy loss on the ground-truth answer tokens (the "gold completion"). DCLM Completion is the token-weighted (micro) average of these losses over all items of those nine tasks. Because it is a loss rather than an accuracy, lower is better. This metric complements Core by providing a continuous, fine-grained signal that is less affected by the discretization inherent in accuracy.

Multi-seed protocol. Each few-shot task is evaluated with 10 independent random seeds that determine the selection and ordering of the few-shot demonstrations. Reported scores are the mean over these 10 runs. Zero-shot tasks (e.g. LAMBADA, Winograd) are deterministic and therefore evaluated with a single run. All evaluations preserve the beginning-of-sequence (BOS) token when truncating context to the model's maximum sequence length. To prevent benchmark contamination, we filtered known evaluation items from the pretraining corpus before training.

Table 6: Evaluation tasks by domain category, each cited to its original source. The 22 DCLM Core tasks are grouped into five categories following the official DCLM; MMLU 5-shot is listed separately from the Core set. Parentheses give the number of tasks in each category, and the right column the number of in-context demonstrations used.

Category	Tasks	Shots
Reading Comprehension (3)	BoolQ (Clark et al., 2019)	10
	CoQA (Reddy et al., 2019)	0
	SQuAD (Rajpurkar et al., 2016)	10
World Knowledge (4)	ARC-Challenge (Clark et al., 2018)	10
	ARC-Easy (Clark et al., 2018)	10
	BB QA Wikidata (Srivastava et al., 2023)	10
	Jeopardy (Kaggle, 2019)	10
Commonsense Reasoning (4)	CommonsenseQA (Talmor et al., 2019)	10
	COPA (Roemmele et al., 2011)	0
	OpenBookQA (Mihaylov et al., 2018)	0
	PIQA (Bisk et al., 2020)	10
Language Understanding (6)	BB Language ID (Srivastava et al., 2023)	10
	HellaSwag 10-shot (Zellers et al., 2019)	10
	HellaSwag 0-shot (Zellers et al., 2019)	0
	LAMBADA (Paperno et al., 2016)	0
	Winograd (Levesque et al., 2012)	0
	Winogrande (Sakaguchi et al., 2020)	0
Symbolic Problem Solving (5)	LSAT AR (Zhong et al., 2024)	3
	BB CS Algorithms (Srivastava et al., 2023)	10
	BB Dyck Languages (Srivastava et al., 2023)	10
	BB Operators (Srivastava et al., 2023)	10
	BB Repeat Copy (Srivastava et al., 2023)	10
Reported separately (1)	MMLU 5-shot (Hendrycks et al., 2021)	5

F.4 Model Configurations

Tables 7 and 8 list the Baseline and SMELT configurations in the 4×4 grid. The Baseline configurations come from a tuned MoE family. Scale identifies each matched compute tier by its Baseline active-parameter scale. The Active params column gives the non-embedding parameter footprint of one physical traversal: attention, normalization, and router weights together with the selected top-8 expert weights. A shared loop block contributes once to Active params. The S column gives the measured compute-equivalent sparsity from Eq. 1, whose FLOPs F count every block execution. Eq. 1 scales the reference model's parameter count by the FLOPs ratio, so a model that spends half the reference model's per-token FLOPs is treated as activating half its parameters. The paper refers to the three sparse levels by the integer labels $S \approx \{85\%, 95\%, 97\%\}$. The Looped Transformer reference model is not dense. To approximately match the Baseline reference model's per-parameter compute intensity, we narrow H and raise the expert count (Section 2.2), so this model is an MoE that routes each token to a top-8 subset of a larger pool (e.g., 16 experts per layer at 200M).

Implementation note. Our models are built on a proprietary Transformer family whose complete architecture and training stack cannot be released. We report the quantities directly relevant to the matched comparisons: model scale, hidden dimension, physical depth, expert count, active parameter count, non-embedding parameter count, and compute-equivalent sparsity. Both architectures are constructed using the same scaling rules and training stack; differences are restricted to the loop configuration and the budget-matching adjustments described in Section 2.2. For physical depth L and loop-span length $m = L/2$, we place $\lfloor (L-m)/2 \rfloor$ layers before the repeated span; with 1-based indexing, the looped layers are 3–7, 4–9, 6–15, and 8–22 for $L = 10$, 12, 20, and 30, respectively. Note that we use different H values at the 600M scale: $H = 1408$ for $S = 0$ and $S \approx 85\%$, and $H = 1512$ for $S \approx 95\%$ and 97%. This is because budget matching jointly adjusts H, the expert intermediate dimension, and attention-head geometry to obtain the closest match in per-token FLOPs, total parameters, and KV cache.

Table 7: Baseline configurations for all 16 grid cells. H is the hidden dimension and L the physical depth. Total experts and Active experts give the stored pool size and the number selected by each routing decision, respectively. Parameter counts exclude embeddings.

Scale	H	L	Total experts	Active experts	Active params	S	Total params
100M	672	10	8	8	0.100B	0.0%	0.10B
100M	672	10	64	8	0.100B	86.0%	0.71B
100M	672	10	192	8	0.101B	95.2%	2.12B
100M	672	10	336	8	0.102B	97.2%	3.70B
200M	1280	12	8	8	0.207B	0.0%	0.21B
200M	1280	12	64	8	0.207B	84.3%	1.32B
200M	1280	12	192	8	0.209B	94.6%	3.87B
200M	1280	12	336	8	0.212B	96.9%	6.74B
600M	1792	20	8	8	0.663B	0.0%	0.66B
600M	1792	20	64	8	0.665B	84.5%	4.28B
600M	1792	20	192	8	0.670B	94.7%	12.54B
600M	1792	20	336	8	0.675B	96.9%	21.83B
1.6B	2304	30	8	8	1.629B	0.0%	1.63B
1.6B	2304	30	64	8	1.632B	84.5%	10.55B
1.6B	2304	30	192	8	1.641B	94.7%	30.94B
1.6B	2304	30	336	8	1.651B	96.9%	53.89B

Table 8: SMELT configurations for all 16 grid cells. H is the hidden dimension and L the physical depth. Total experts and Active experts give the stored pool size and the number selected by each routing decision, respectively. Loop gives the repeated span and total number of passes. Parameter counts exclude embeddings. The final two columns give the per-token training-FLOPs and KV-cache ratios relative to the corresponding Baseline.

Scale	H	L	Total experts	Active experts	Loop Configuration	Active params	S	Total params	$F_{\mathrm{SMELT}}/F_{\mathrm{Base}}$	$\mathrm{KV}_{\mathrm{SMELT}}/\mathrm{KV}_{\mathrm{Base}}$
100M	576	10	16	8	Mid. 50% $\times 2$	0.071B	0.0%	0.10B	1.081	1.018
100M	576	10	96	8	Mid. 50% $\times 2$	0.067B	86.8%	0.71B	1.024	0.964
100M	576	10	288	8	Mid. 50% $\times 2$	0.069B	95.5%	2.12B	1.031	0.964
100M	576	10	504	8	Mid. 50% $\times 2$	0.070B	97.4%	3.70B	1.039	0.964
200M	1056	12	16	8	Mid. 50% $\times 2$	0.142B	0.0%	0.21B	1.039	1.031
200M	1056	12	96	8	Mid. 50% $\times 2$	0.140B	84.6%	1.32B	1.021	1.031
200M	1056	12	288	8	Mid. 50% $\times 2$	0.142B	94.7%	3.89B	1.029	1.031
200M	1056	12	504	8	Mid. 50% $\times 2$	0.145B	96.9%	6.78B	1.037	1.031
600M	1408	20	16	8	Mid. 50% $\times 2$	0.458B	0.0%	0.67B	1.046	1.031
600M	1408	20	96	8	Mid. 50% $\times 2$	0.443B	85.1%	4.31B	1.007	1.031
600M	1512	20	288	8	Mid. 50% $\times 2$	0.453B	94.7%	12.44B	1.024	0.984
600M	1512	20	504	8	Mid. 50% $\times 2$	0.459B	96.9%	21.70B	1.031	0.984
1.6B	1848	30	16	8	Mid. 50% $\times 2$	1.101B	0.0%	1.64B	1.022	0.984
1.6B	1848	30	96	8	Mid. 50% $\times 2$	1.081B	84.6%	10.45B	1.000	0.984
1.6B	1848	30	288	8	Mid. 50% $\times 2$	1.091B	94.8%	30.90B	1.004	0.984
1.6B	1848	30	504	8	Mid. 50% $\times 2$	1.103B	97.0%	53.90B	1.008	0.984

Worked matching example. Consider a Looped Transformer at the 200M scale and $S \approx 95\%$ that loops the middle 6 of its 12 layers twice, executing 18 layers in total. The matched Baseline has $L = 12$, hidden dimension $H = 1280$, and 192 experts per layer (1.33×10^9 training FLOPs per token, 3.87×10^9 total non-embedding parameters). The 6 extra layer executions cost FLOPs, so we narrow H from 1280 to 1056 to keep per-token FLOPs close. Narrowing H also shrinks every expert's FFN, which reduces total parameters. To recover them, we raise the per-layer expert count from 192 to 288. Expert count is uniform across all layers in both architectures. After both adjustments, per-token FLOPs land at 1.37×10^9 ($+2.9\%$ vs. the Baseline's 1.33×10^9), total parameters at 3.89×10^9 ($+0.4\%$), and KV cache within 4%. Both FLOPs counts are measured at the mean document length within the packed 4096-token contexts (since segment-level masks restrict attention to each document) and include the attention cost that scales with that length.

Matching granularity. The residual mismatch is larger at the $S = 0\%$ level, where the expert pool is small and each hardware-aligned change to model width, expert intermediate dimension,

head geometry, or expert count moves a comparatively large fraction of the budget. The feasible configurations therefore form a coarser grid than at the three sparse levels. Across the 12 sparse cells used in that fit, the maximum absolute mismatches are 3.9% in per-token FLOPs, 1.0% in total parameters, and 3.6% in KV cache. Besides, these residual differences are not treated as exact budget equality in the scaling analysis: every run enters the fit at its measured per-token FLOPs F, and the reported compute-efficiency comparisons evaluate the two separately fitted surfaces at common (C, S) coordinates.

F.5 Design Ablation Results

Tables 9–11 give the full results of the three ablations in Sections 2.3–2.5, adding DCLM Core and DCLM Completion to the validation loss in Figure 2.

Table 9: Loop-span sweep at 200M and $L = 12$. Validation loss is minimized near 50% span for both sparsity levels; we adopt this as the default loop span. Bold marks the best value within each sparsity level. Standard errors reflect variability across 10 few-shot evaluation seeds (Appendix F.3); each configuration is trained once, so they do not capture training-run variability.

Looped layers	Span (%)	Effective layers	Val. loss ↓	DCLM Core ↑	DCLM Completion ↓
		$S \approx 85\%$			
0	0	12	1.9445	24.92 ± 0.13	2.2887
2	17	14	1.9384	26.20 ± 0.14	2.2824
4	33	16	1.9275	25.13 ± 0.16	2.2733
6	50	18	**1.9257**	$\mathbf{27.57 \pm 0.13}$	**2.2635**
8	67	20	1.9374	25.84 ± 0.10	2.2843
10	83	22	1.9413	24.45 ± 0.17	2.2835
12	100	24	1.9322	25.31 ± 0.17	2.2759
		$S \approx 95\%$			
0	0	12	1.8735	29.34 ± 0.10	2.2189
2	17	14	1.8562	30.84 ± 0.20	2.2022
4	33	16	1.8524	31.22 ± 0.18	2.1970
6	50	18	**1.8517**	29.68 ± 0.13	2.1917
8	67	20	1.8544	$\mathbf{31.78 \pm 0.18}$	**2.1891**
10	83	22	1.8572	31.49 ± 0.15	2.1964
12	100	24	1.8601	30.82 ± 0.14	2.2038

At $S \approx 85\%$, all three metrics in Table 9 pick the 50% span. The table also repeats the sweep at $S \approx 95\%$, where validation loss differs by less than 0.003 nats among the 33%, 50%, and 67% spans and both DCLM metrics are best at 67%. At both sparsity levels, every looped span reaches lower validation loss and lower DCLM Completion than the unlooped model.

In the depth sweep of Table 10, the Looped Transformer with 12 physical and 18 executed layers is best on all three metrics. The Baseline's validation loss is lowest at 12 layers, while both DCLM metrics favor 15 layers. Selecting the Baseline's depth by DCLM instead would not change the comparison, since the 15-layer Baseline still trails the 12/18 Looped Transformer by 1.7 pp on DCLM Core and 0.019 nats on DCLM Completion.

In the loop-count sweep of Table 11, two visits are best on all three metrics. Three and four visits still beat the Baseline on all three, by more than 2 pp on DCLM Core, so the narrower models that extra visits require give back only part of the gain of the second visit.

Table 10: Effective-depth sweep at 200M, $S \approx 85\%$, 50% loop span. Physical depth is unique layers; effective depth is layers run in a forward pass. The $12/18$ Looped Transformer wins on all three metrics; its physical depth matches the Baseline's validation loss optimum (12). Bold marks the best value within each architecture.

Physical depth	Effective depth	Val. loss ↓	DCLM Core ↑	DCLM Completion ↓
		Baseline		
9	9	1.9638	24.01 ± 0.14	2.2983
12	12	**1.9445**	24.92 ± 0.13	2.2887
15	15	1.9457	$\mathbf{25.84 \pm 0.15}$	**2.2824**
18	18	1.9542	24.26 ± 0.14	2.2967
		Looped Transformer		
8	12	1.9591	24.39 ± 0.17	2.2932
10	15	1.9389	25.93 ± 0.14	2.2871
12	18	**1.9257**	$\mathbf{27.57 \pm 0.13}$	**2.2635**
14	21	1.9433	26.55 ± 0.23	2.2766

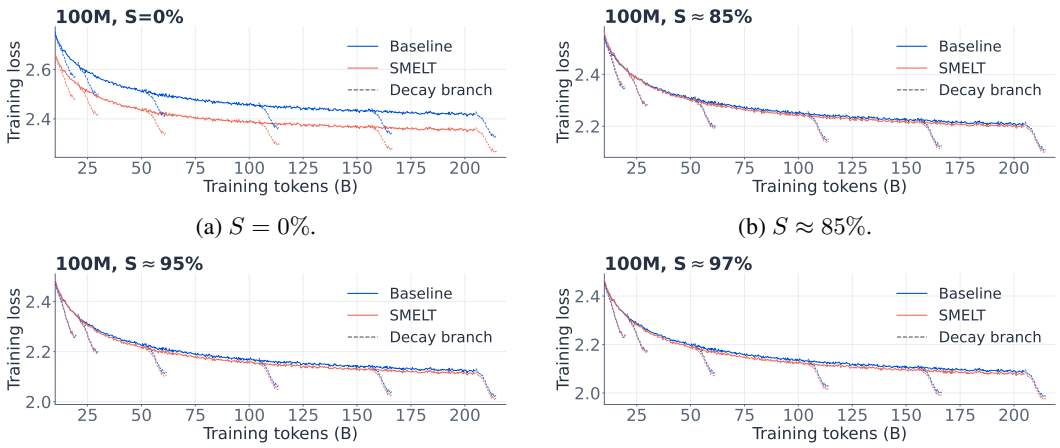
Table 11: Loop-count sweep at 200M, $S \approx 85\%$, 50% loop span, physical depth matched to the Baseline. Two visits give the lowest validation loss and highest DCLM Core; more loops force a thinner model under matched FLOPs. Bold marks the best value.

Model	Effective depth	Val. loss ↓	DCLM Core ↑	DCLM Completion ↓
Baseline ($1\times$)	12	1.9445	24.92 ± 0.13	2.2887
Looped Transformer $2\times$	18	**1.9257**	$\mathbf{27.57 \pm 0.13}$	**2.2635**
Looped Transformer $3\times$	24	1.9385	27.15 ± 0.20	2.2779
Looped Transformer $4\times$	30	1.9360	27.30 ± 0.19	2.2820

G Training Curves for Every Grid Cell

Section 3.1 compares the endpoints of the full grid, and this appendix shows the training curves behind them. Figures 24–27 give one figure per scale, each with one panel per sparsity level, $S{=}0\%$ and $S \approx \{85\%, 95\%, 97\%\}$. Each panel plots training loss against training tokens for one matched Baseline / SMELT pair from 10B tokens onward. Solid lines are the stable constant-learning-rate phase, and dashed lines are the six cosine-decay branches forked at steps 10,000, 20,000, 50,000, 100,000, 150,000, and 196,075. Single logged steps are noisy, so we compare where a branch ends by the mean of its last five logged steps. By this measure, SMELT's longest branch ends below the Baseline's in all 16 cells, by 0.006 to 0.061 nats. Across all 96 decay branches, SMELT ends above the Baseline on seven, all forked at or before step 50,000, and every branch forked at step 100,000 or later ends below it.

At 100M (Figure 24), the $S = 0\%$ pair shows the largest gap in the grid, 0.061 nats on the longest branch, while the three sparse levels end 0.009 to 0.012 nats apart. The earliest decay branch at each sparse level is the exception at this scale: SMELT ends it 0.004 to 0.008 nats above the Baseline.

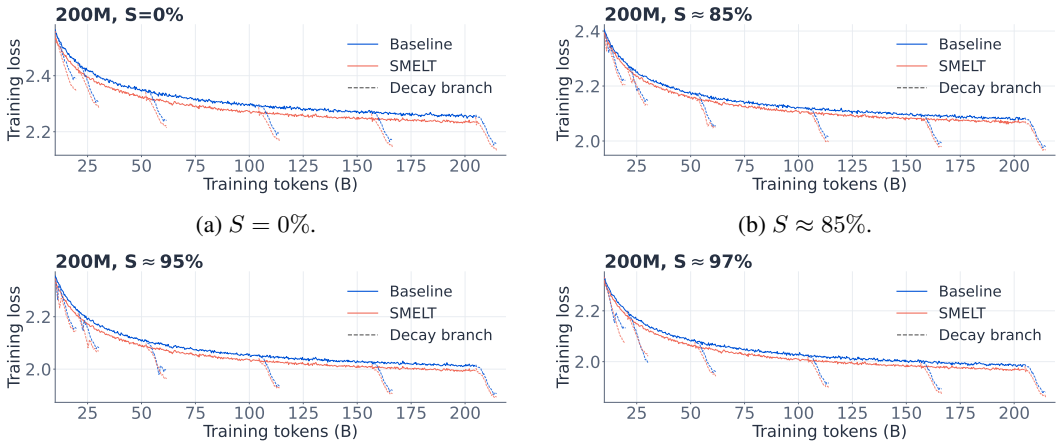


(a) $S = 0\%$. (b) $S \approx 85\%$.

(c) $S \approx 95\%$. (d) $S \approx 97\%$.

Figure 24: Training loss at the 100M scale across all four sparsity levels.

At 200M (Figure 25), the longest branches end 0.012 to 0.017 nats apart at all four sparsity levels. The one reversal is the 20,000-step branch at $S \approx 97\%$, which SMELT ends 0.026 nats above the Baseline.

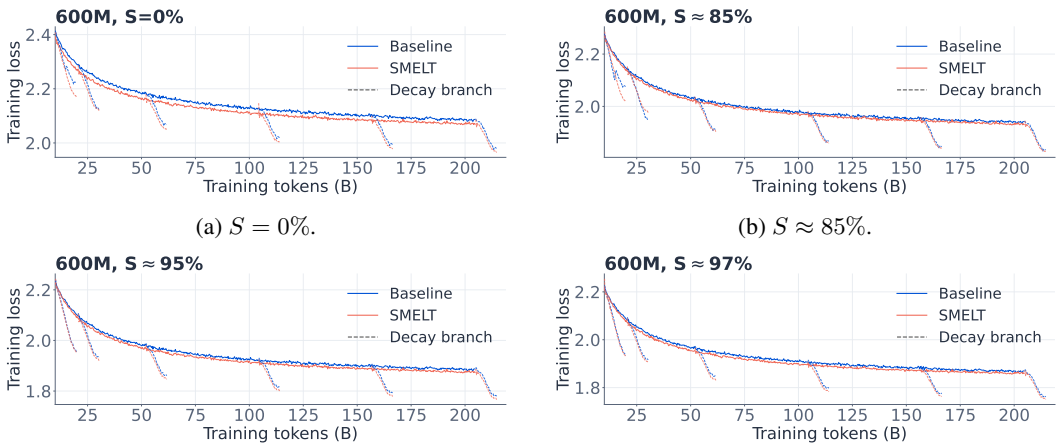


(a) $S = 0\%$. (b) $S \approx 85\%$.

(c) $S \approx 95\%$. (d) $S \approx 97\%$.

Figure 25: Training loss at the 200M scale across all four sparsity levels.

At 600M (Figure 26), the longest branches end 0.006 to 0.012 nats apart. The one reversal is the 20,000-step branch at $S \approx 85\%$, which SMELT ends 0.028 nats above the Baseline.

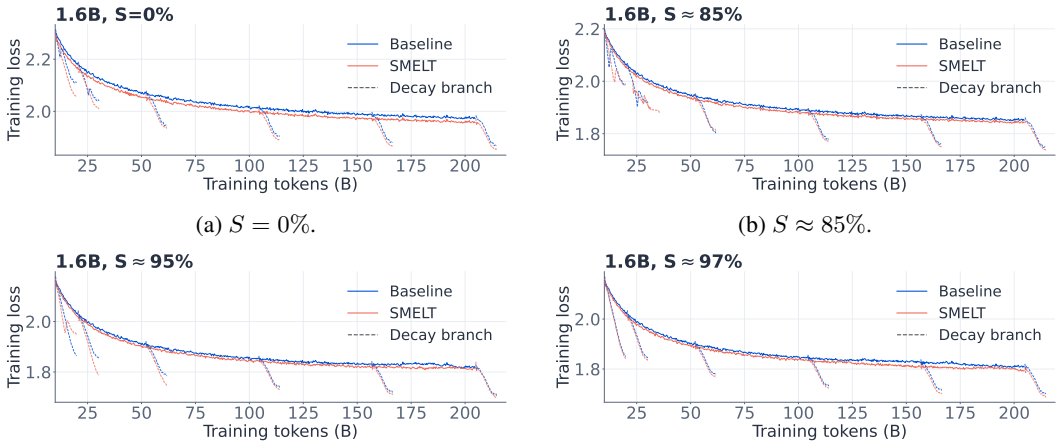


(a) $S = 0\%$. (b) $S \approx 85\%$.

(c) $S \approx 95\%$. (d) $S \approx 97\%$.

Figure 26: Training loss at the 600M scale across all four sparsity levels.

At 1.6B (Figure 27), the longest branches end 0.007 to 0.014 nats apart. Two early branches reverse. The 50,000-step branch at $S \approx 85\%$ ends 0.009 nats above the Baseline. On the 10,000-step branch at $S \approx 95\%$, SMELT's loss spikes midway through the decay and ends 0.080 nats above the Baseline, the largest reversal in the grid. The stable run of that cell shows no spike at the same point, and every later branch forked from it ends below the Baseline, so the spike is confined to that one decay run.



(a) $S = 0\%$. (b) $S \approx 85\%$.

(c) $S \approx 95\%$. (d) $S \approx 97\%$.

Figure 27: Training loss at the 1.6B scale across all four sparsity levels.