

Weight-Tied Looped Depth for Data-Constrained Pretraining

Sebastian Miles
miless@chalmers.se

Abstract

The BabyLM Strict-Small track fixes the training corpus at 10M words and the budget at 10 epochs. Data cannot be added, so any gain has to come from the architecture. We ask whether weight-tied looped depth, applying the same transformer block stack twice per forward pass, helps under that constraint. On a 50M-parameter model built from nanochat it does. Held-out bits-per-byte (`val_bpb`, a per-byte cross-entropy that is comparable across tokenizers) falls from 1.3222 to 1.3175 as a three-seed mean, and every looped run lands below every baseline run, at no parameter cost. Downstream the effect points the same way, but we cannot resolve it: averaged over five zero-shot benchmarks under the official harness, the looped mean is 50.00 against the baseline’s 49.90, a gap of 0.10 points against a standard error of 0.17. The second pass is paid for in computation. Training takes $1.71\times$ as long, and a 100M-parameter model trained for a comparable wall-clock time still scores better on both metrics. What looping buys here is parameter efficiency, not compute efficiency.

The baseline these experiments run on was tuned by a human-directed agentic loop. The researcher proposed each hypothesis; an agent implemented it, trained for the fixed budget around the clock on one GPU, and kept or reverted the change under a fixed rule on `val_bpb`. We treat the loop as methodology and report its rejected hypotheses alongside the accepted ones.

1 Introduction

The BabyLM challenge (Charpentier et al., 2025) asks how well a language model can perform when it is given a developmentally plausible amount of text. The gap between that budget and the one large language models enjoy is enormous. As the challenge organizers note, Chinchilla (Hoffmann et al., 2022) sees about 1.4 trillion tokens during

training, well over 10,000 tokens for every single word a 13-year-old child has heard in their entire life. The Strict-Small track limits the corpus to 10M words and training to 10 epochs, so a model sees at most 100M words in total (Choshen et al., 2026).

That budget leaves the model underprovisioned. Scaling laws (Kaplan et al., 2020) show that model size and data have to grow together, and a model whose size grows alone will overfit. A 50M-parameter model such as ours would ideally see on the order of a billion tokens, roughly twenty per parameter (Hoffmann et al., 2022). Ten epochs over 10M words is an order of magnitude short of that, and re-reading a fixed corpus returns less with each pass (Muennighoff et al., 2023). The same bottleneck is now visible at the frontier: the stock of high-quality human-generated text is finite and, on current trends, is projected to be largely exhausted within the decade (Villalobos et al., 2024). With the corpus fixed, the levers that remain are the architecture and the training recipe.

Weight-tied looped depth. Our main contribution is empirical. In this data-constrained setting, *weight-tied looped depth* lowers validation loss at no parameter cost. The construction is simple: the same transformer stack is applied P times per forward pass, so an L -layer model computes at an effective depth of PL while its parameter count stays that of the L -layer model. It is also old, running from Universal Transformers (Dehghani et al., 2019) and ALBERT (Lan et al., 2020) to a recent wave of looped language models (Giannou et al., 2023; Bae et al., 2025). What we add is the measurement in this regime: re-applying the stack with the same weights improves held-out bits-per-byte at every seed we ran, and the downstream benchmarks move in the same direction, though by less than the spread between seeds. Recent at-scale studies report the opposite trade (Saunshi et al., 2025; Zhu

et al., 2025b): there looping buys a better reasoning inductive bias and gives up perplexity. We attribute the difference to the data budget. Those models are trained close to compute-optimal, where the loss is limited by capacity; ours is mildly underfit on a corpus it has already read ten times, and extra computation still has something to do.

Routing over depth did not help on top of our baseline. We also pursued, at length, a way of enriching how information moves between layers (Pagliardini et al., 2024; Zhu et al., 2025a). In the form we tried, *delta attention residuals* (Luo et al., 2026), each layer attends through learned routing weights over the updates that earlier sublayers wrote into the residual stream ($\mathbf{v}_i = \mathbf{h}_{i+1} - \mathbf{h}_i$). This looked especially promising under the loop, since the second pass can route over the first pass’s updates. It was neutral or worse in every run we did, single-pass and looped alike, costing validation loss with no compensating benchmark gain and adding substantially to training time in our implementation (Section 5). We read the result narrowly. Routing has little left to add *on this baseline*, which already carries two cheaper paths that do similar work; that is not evidence against depth routing in the low-data regime in general.

Found and validated by a human-agent research loop. The baseline itself came out of a loop we built by modifying Karpathy’s autoresearch framework (Karpathy, 2025a), which scheduled training runs and evaluations automatically, around the clock, on a single GPU. The division of labour was fixed: the human researcher proposed the architectural directions, and the agent implemented each change, ran the fixed 10-epoch budget, and reported both validation bits-per-byte and the downstream benchmark results. The default rule was mechanical. A change was kept if val_bpb improved beyond a noise threshold and reverted otherwise. The human intervened when val_bpb and the benchmarks disagreed, and in those cases the benchmarks decided. We treat the loop as methodology (Section 3.3). It is not the contribution. What it produced is the tuned baseline of Section 4.1, together with a record of everything that failed along the way.

Findings. On a fixed 50M-parameter model and a fixed 10-epoch budget over the 10M-word corpus, weight-tied looped depth lowers val_bpb from 1.3222 to 1.3175 as a three-seed mean, and ev-

ery looped run falls below every baseline run, at *zero* additional parameters. Downstream the picture is weaker. On the average of our five zero-shot benchmarks (BLiMP, BLiMP-supplement, EWoK, COMPS, and entity tracking), scored under the official harness, the looped model gains 0.10 points (50.00 against 49.90, both three-seed means) with a standard error of 0.17. The direction agrees with val_bpb, but three seeds cannot resolve an effect that size, and no individual benchmark separates the two models (Section 4.2). Nor does the loop substitute for capacity: a 100M-parameter, depth-12 model reaches 50.19 over two seeds and keeps a clear val_bpb advantage (1.2982 against 1.3175), for a comparable wall-clock cost. What a second pass buys, then, is a small and repeatable gain in validation loss that costs parameters nothing and computation a great deal. Against it stands a long list of *rejected* hypotheses: depth routing over sublayers, deep supervision, and many dataset modifications including cleaning, deduplication, and register re-weighting. Each of these matched the baseline or hurt it.

2 Related Work

Weight-tied and looped transformers. Applying one block stack repeatedly to grow effective depth at fixed parameter count has a long lineage. Universal Transformers (Dehghani et al., 2019) recur a single layer in depth with a shared weight matrix, and ALBERT (Lan et al., 2020) shares parameters across all layers to cut model size without losing quality. More recent work studies the loop explicitly: Looped Transformers (Giannou et al., 2023) show that a weight-tied stack iterated in depth can express iterative computation, and Relaxed Recursive Transformers (Bae et al., 2025) recover most of the quality lost to hard weight sharing by adding small layer-wise adapters. A recent wave scales the idea to full language models. Saunshi et al. (2025) and Ouro (Zhu et al., 2025b) pre-train looped models at the billion-parameter scale, and both report the same trade at fixed effective depth: looping strengthens the reasoning inductive bias and pays for it in memorization and perplexity. We measure the opposite sign on perplexity, and Section 1 gives our reading of why.

Dense depth routing and residual mixing. We also studied, and rejected, a new technique to move information between layers. DenseFormer (Pagliardini et al., 2024) adds learned depth-weighted av-

eraging of all previous layer outputs, and Hyper-Connections (Zhu et al., 2025a) generalize the single residual stream to multiple learned connections. Our own model already carries a scalar residual-and-embedding mix at each layer, in the ResFormer and value-residual family (Zhou et al., 2025). The delta attention residuals we tried replace that scalar mix with a per-token softmax over the individual sublayer updates. Under the loop they can also route over an earlier pass’s updates, which is a source no single-pass model has. We report the line as a negative result (Section 5): the new cross-pass sources did not turn into a gain, and in this data-constrained regime the plain residual stream proved hard to beat.

Sample-efficient pretraining. The BabyLM challenge (Charpentier et al., 2025) constructs the data-constrained setting and the accompanying zero-shot evals (BLiMP, Warstadt et al., 2020; the BLiMP supplement, Charpentier et al., 2025; EWoK, Ivanova et al., 2025; COMPS, Misra et al., 2023; and entity tracking, Kim and Schuster, 2023). GPT-BERT (Charpentier and Samuel, 2024) shows that mixing causal and masked-language-model objectives improves data efficiency, a technique we adopt in our baseline.

Optimizer and base-model components. We use the Muon optimizer (Jordan et al., 2024) for hidden matrices and AdamW for embeddings and scalars, rotary position embeddings (Su et al., 2024), SwiGLU feed-forward layers (Shazeer, 2020). Two of our baseline’s retained wins are also established methods: stochastic weight averaging (Izmailov et al., 2018) and multi-token prediction (Gloeckle et al., 2024; DeepSeek-AI et al., 2025).

Agent-assisted research. The methodology adapts Karpathy’s autoresearch (Karpathy, 2025a), part of a broader line of work on language-model agents in machine-learning research. We do not claim autonomy: ours is a human-in-the-loop process in which the researcher steers and the agent implements, runs, and analyzes. We use it as a tool for automatically scheduling training and evaluations around the clock.

3 Method and Setup

3.1 Base model and budget

The starting point is a $\sim 50\text{M}$ -parameter decoder-only transformer with depth $L=8$,

model dimension $d=512$, head dimension $d_h=128$, RoPE, SwiGLU MLPs, and the modded-nanogpt/nanochat value-embedding mechanism (Karpathy, 2025b), inspired by ResFormer-style value residuals (Zhou et al., 2025). Since the value-embedding mechanism exists only as code in its source (to our knowledge), we state it explicitly. Every second attention layer, with the last layer always included, carries a lookup table $U_i \in \mathbb{R}^{|V| \times d}$ alongside the usual token embedding. In that layer’s attention the values are updated as $v \leftarrow v + gu$, where $u := U_i[w]$ is the table entry for the current token id w , and $g := 2\sigma(W_i^g x_{[:32]}) \in (0, 2)^H$ is a learned gate with one scalar per attention head. Here $x_{[:32]} \in \mathbb{R}^{32}$ denotes the first 32 channels of the layer input x , $W_i^g \in \mathbb{R}^{H \times 32}$ is a per-layer learned projection with H the number of attention heads, and σ is the logistic sigmoid; each head’s gate scalar scales that head’s slice of u . Because u depends on the token id alone, this adds a context-independent per-token component to the attention values, bypassing the residual stream. At each block the input is a learned scalar mix of the running residual x and the embedding skip x_0 , the normalized token embedding,

$$\text{base}_i = \lambda_i x + \mu_i x_0, \quad (1)$$

with learned scalars λ_i, μ_i ; the embedding skip is likewise inherited from the modded-nanogpt baseline (Karpathy, 2025b). Hidden matrices are optimized with Muon; embeddings, projections, and per-layer scalars are optimized with AdamW. Training is a fixed budget of 10 full passes over the corpus at sequence length 2048. On a single RTX 3060 (12 GB) a single-pass run takes roughly 1 hour, and the submitted looped model roughly 1.9 hours (Section 4.2). Training uses the detoxified BabyLM 2026 Strict-Small corpus (Choshen et al., 2026) (9.99M words; CHILDES, Project Gutenberg, OpenSubtitles, Simple Wikipedia, BNC spoken, and Switchboard) with a fixed Morfessor-based (Virpioja et al., 2013) subword tokenizer (vocabulary 8192, byte fallback).

The submitted model keeps this backbone and differs from the initial baseline in five ways. First, training follows the GPT-BERT hybrid objective (Charpentier and Samuel, 2024), which trains one model on both a causal and a masked objective: 15% of microbatches are trained as a bidirectional masked LM and the remaining 85% causally. Second, a multi-token-prediction head (Gloeckle et al.,

2024) adds an auxiliary loss that predicts the token at $t+2$ from the same final hidden state, giving the representation a second, longer-range target to satisfy. Third and fourth, we hold weight decay constant on the hidden matrices instead of scheduling it, and apply decoupled weight decay to the value-embedding tables, which are large enough to overfit on their own (Section 4.1). Fifth, the weights we evaluate are not the final iterate but a stochastic weight average (Izmailov et al., 2018), the uniform mean of the weights over the final 40% of training. On top of these, the block stack is applied twice per forward pass (Section 3.2). How each of these changes was found, and what each contributes, is traced in Section 4. Full-harness evaluations of this configuration, and of every other configuration we scored that way, are in Appendix A, Table 4.

3.2 Architecture: looped depth, and the routing extension we tested

Weight-tied looped depth. Let the block stack be $B_0, \dots, B_{L-1}$, where each block represents a standard attention sublayer followed by an MLP sublayer, and let x_0 be the normalized token embedding. A standard forward pass applies each block once. Looped depth instead applies the whole stack P times with the *same* weights, so the effective depth is $P \cdot L$ while the parameter count is exactly that of the depth- L model. At each block the input mix base_i of Section 3.1 is formed as usual, the block is applied to it, and the block’s output becomes the running residual x seen by the next block; the P passes simply repeat this over the stack. Re-injecting the input at every iteration is standard in looped and recurrent-depth models (Bai et al., 2019; Geiping et al., 2025). The baseline uses $P=1$ and in our submitted model we use $P=2$.

Cross-pass delta attention residuals. On top of the loop we evaluated a routing mechanism based on delta attention residuals (Luo et al., 2026), intended to work together with the newly acquired model depth. We describe its form here because our analysis of *why* it fails (Section 5) depends on it. During the looped forward pass we maintain a growing set of *sources*, comprising the embedding x_0 together with every per-sublayer delta

$$v_j = x_j - x_{j-1}, \quad j = 1, \dots, 2LP, \quad (2)$$

where sublayers are indexed consecutively across all P passes and x_{j-1} , x_j denote the residual

stream entering and leaving sublayer j . The definition is the same for both sublayer kinds and each sublayer adds its output to the stream, so v_j is the attention output when j is an attention sublayer and the MLP output when j is an MLP sublayer; every block thus contributes two separate deltas per pass, and each is routed over individually. Before routing sublayer s , with input h , a per-token softmax routes over the sources seen so far and adds the mix back into the sublayer input:

$$\begin{aligned} \ell_j &= q_s^\top \text{norm}(v_j), \\ h' &= h + \sum_j \text{softmax}_j(\ell) v_j, \end{aligned} \quad (3)$$

where v_j is the stored source value and $q_s \in \mathbb{R}^d$ the learned query of routing sublayer s , the only new parameters. Everything is zero-initialized so that the model starts exactly at the looped-depth baseline. The idea is that second-pass sublayers can route over every first-pass delta, including those of sublayers deeper than themselves, which no single pass can provide.

3.3 The human-agent research loop

The baseline above was built with the help of an agentic loop directed by a human. We describe it here as methodology. The loop operates on a small, fixed codebase split in two. `prepare.py` holds the data, the tokenizer, the dataloader, and the `evaluate_bpb` evaluator, which together define the task and the metric, and it is read-only. `train.py` holds the model, the optimizer, and the training loop, and it is the only file the agent may edit. Architecture, optimizer, hyperparameters, batch size, and model size are therefore all in scope; the evaluator and the dependencies are not.

Each iteration runs as follows:

- (i) edit `train.py` with one hypothesis and commit
- (ii) train for exactly 10 epochs
- (iii) read `val_bpb` and the per-epoch validation curve from the run log
- (iv) *keep* the change if `val_bpb` improves beyond a noise threshold (~ 0.005 , picked by hand rather than estimated from repeated runs), otherwise `git reset` to the previous best
- (v) document the result

Architectural hypotheses come from the human researcher. The agent applies the specified edits, schedules the training runs and evaluations, and logs the results. The point of the arrangement is scheduling: the human can read papers and implementations while the GPU stays busy, which on a single consumer card is most of the constraint.

Metric. The in-loop objective is validation bits-per-byte (`val_bpb`), a per-byte cross-entropy on held-out text. Measuring per byte instead of per token makes the number comparable across tokenizers, which matters here because the search changed the tokenizer as well as the model. In later iterations we also ran the zero-shot benchmarks (BLiMP, BLiMP-supplement, EWoK, COMPS, entity tracking) to check hypotheses that `val_bpb` might mislead us about. It is an imperfect proxy for them: some changes improve linguistic generalization while leaving the causal cross-entropy flat or slightly worse, and a rule reading `val_bpb` alone would reject them. The masked-LM objective of Section 3.1 is exactly such a case.

What the loop was and was not allowed to decide. Agentic search invites two failure modes, unvalidated code and a metric that drifts along with the search, and we constrained both. The evaluator, the data, and the metric live in the read-only half of the codebase, so no run can change what a number means and every configuration is scored by the same code. Each iteration is a single commit whose diff the human read before the run started, and every run log we quote from is released with the code. As a reproduction check, we re-ran the $P=1$ baseline at seed 42 from a later commit and recovered the original `val_bpb` to 4×10^{-5} .

We also separate two kinds of number in what follows. The headline comparison of Section 4.2 is confirmatory in intent: three seeds of each model, all five benchmarks under the official full harness, and an average over all five with no task dropped for being noisy. Everything else, including the fast evaluations and the single-seed screens of Section 5 and Appendix A, is the output of a greedy search and is labelled as such where it appears. We did not pre-register the protocol, and the search that produced the baseline was history-dependent by construction, so this is a separation we impose on the reporting and not a property of how the experiments were run.

Change (kept)	Type	val_bpb
Baseline (depth 8, $d=512$)	base	1.5598
Muon LR 0.04 $\rightarrow$ 0.06	optim	1.5468
Muon LR 0.06 $\rightarrow$ 0.08	optim	1.5325
Constant weight decay	reg	1.4996
Weight decay 0.2 $\rightarrow$ 0.3	reg	1.4815
Full-attention window (L)	arch	1.4738
Stochastic weight averaging	avg	1.3898 [†]
Muon LR 0.08 $\rightarrow$ 0.10	optim	1.3824 [†]
Value-embed weight decay 0.1	reg	1.3385 [†]
Muon weight decay 0.3 $\rightarrow$ 0.2	reg	1.3312 [†]
SWA start 0.5 $\rightarrow$ 0.6	avg	1.3266 [†]
Sequence-length warmup	signal	1.3242 [†]
Multi-token prediction ($K=2$)	signal	1.3173[†]

Table 1: The tuned baseline, in discovery order. Each kept change is an optimizer, regularization, averaging, or training-signal change on a fixed-capacity model. [†]From Stochastic weight averaging onward, `val_bpb` is evaluated on the SWA-averaged weights (Section 3.1), not the raw final-epoch weights; rows above are raw.

4 Results

4.1 A strong regularization baseline

Both architectural changes of Section 3.2 are evaluated on top of a heavily tuned baseline. We tuned first and changed the architecture second so that any improvement would be attributable to the loop or the routing, and not to optimization slack the baseline had left on the table. Table 1 lists the chain of *kept* changes the loop produced, from the initial model to that baseline. Most rows cleared the ~ 0.005 noise threshold of Section 3.3 on their own. Two did not: the SWA start shift (0.5 $\rightarrow$ 0.6) and sequence-length warmup were kept on human judgement, so the threshold was in practice applied softly. Over the whole chain `val_bpb` falls from 1.560 to 1.317.

The single largest step is stochastic weight averaging (Izmailov et al., 2018), worth -0.084 . Rather than score the final, noisy iterate, it scores a uniform average of the weights over the last stretch of training, and it composes cleanly with the learning-rate increase that follows it. Every other large move was regularization. Switching to a constant weight-decay schedule removed an overfitting tail in the back half of training, and adding weight decay to the value-embedding tables changed the shape of the validation curve entirely: it had been U-shaped, bottoming out around epoch 6 and rising after, and it became monotonically descending across nearly the whole budget (Figure 1).

One final change completes the baseline. On top of the tuned configuration we add the GPT-

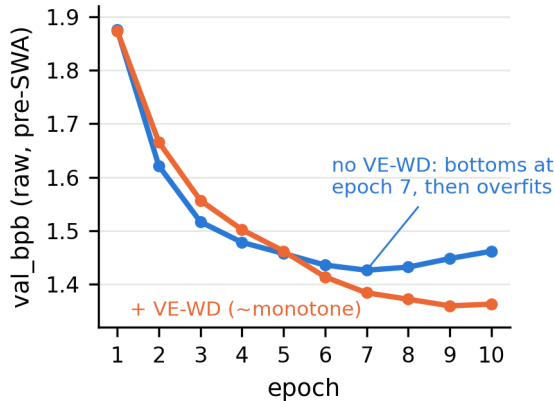


Figure 1: Per-epoch val_bpb (raw, pre-SWA) across the full 10-epoch budget for the last kept step before adding weight decay to the value-embedding tables (blue) and the step that adds it (orange). Without value-embedding weight decay the curve bottoms at epoch 7 and then rises as the 17M-parameter value tables overfit; with it the curve descends monotonically through epoch 9, with only a negligible uptick at the final epoch.

BERT masked-LM objective (Charpentier and Samuel, 2024) on 15% of microbatches. This raises val_bpb slightly, from 1.3173 to 1.3221, because the metric is measured causally and cannot see the bidirectional benefit, while improving downstream grammar. This MLM-augmented model, at val_bpb 1.3221, is the baseline for all architectural experiments below.

Finally we observe that the tuned model mildly *underfits* at 10 epochs: extending the budget to 14 epochs improves val_bpb by a further -0.008 before flattening (Appendix A.2).

4.2 Looped depth: a second pass over the block stack at zero parameter cost

Applying the block stack twice per forward pass ($P=2$) is our main confirmed result. It lowers val_bpb at all three seeds we ran. The three looped draws are 1.3148, 1.3178, and 1.3199 at seeds 42, 43, and 44, giving a mean of 1.3175 and a standard deviation of 0.0026. The same three seeds of the $P=1$ baseline span 1.3202 to 1.3249 (mean 1.3222, s.d. 0.0024; seeds 43 and 44 come from an earlier commit that reproduces seed 42’s val_bpb to 4×10^{-5}). The two ranges do not overlap. The gap between the means is -0.0047 with a standard error of 0.0020. That gap is narrower than the ~ 0.005 threshold of Section 3.3, which is worth stating plainly: the threshold is a single-run accept rule and does not bound a difference of means, so

the two numbers are not in tension. The looped runs also reach a lower final training loss, so the gain is not an artifact of the averaging window.

Downstream, the direction agrees but the size does not resolve. Table 3 reports the full zero-shot suite for the same three seeds of each model, scored under the official harness. Looped depth raises the five-task average by 0.10 points (50.00 against 49.90) with a standard error of 0.17. Per task, four of five move up and one moves down: BLiMP-supplement $+0.64$, BLiMP $+0.19$, entity tracking $+0.16$, COMPS $+0.13$, EWOK -0.63 . None of these is larger than its own seed spread. The baseline seeds span 0.51 points on the average (49.59–50.10) and the looped seeds 0.18 (49.93–50.11), so at three seeds the downstream effect is not resolvable, and we do not claim one. Two of the five tasks are also weak instruments at this scale. Entity tracking sits within a point or two of chance for every configuration we trained, looped or not, and the reading-time correlations that the harness also reports are near zero everywhere and are omitted from the average (Appendix A). Supervised fine-tuning tells the same story: on the BabyLM GLUE tasks the model lands near majority-class baselines (Appendix A). Whatever competence a 50M-parameter model acquires from 10M words shows up in the grammatical probes, and BLiMP and its supplement are the only two tasks where we would expect an architectural change of this size to be visible at all.

What the second pass costs. The looped model has exactly the same 50M parameters as the baseline and spends more of the fixed budget refining its representation in depth. Table 2 prices that. At $1.71\times$ the baseline’s training time, the second pass is expensive, and two cheaper or comparable ways of spending the same compute do at least as well on val_bpb. Training the $P=1$ baseline for 14 epochs instead of 10 reaches 1.3140 in $1.24\times$ the baseline’s time, below every looped draw. That run exceeds the track’s epoch limit, so we report it as a probe and not as a submittable configuration, but it bounds what the loop is worth against the simpler option of training longer. A 100M-parameter depth-12 model reaches 1.2979 at $1.80\times$ the time, close to the looped model’s cost and clearly ahead of it on both val_bpb and the downstream average (50.19 over two seeds against 50.00). We therefore state the claim as parameter efficiency and not compute efficiency: at a fixed parameter count the loop

Configuration (seed 42)	Par.	Time	val_bpb	BLiMP	Model	BLiMP	Supp.	EWoK	COMPS	Ent.	Avg.
Baseline $P=1$, 10 epochs	50M	1.00×	1.3215	71.27	Baseline, s42	71.27	54.69	51.42	51.67	18.91	49.59
Baseline $P=1$, 14 epochs [‡]	50M	1.24×	1.3140	71.19	Baseline, s43	71.72	55.37	52.59	51.16	19.16	50.00
Looped $P=2$, 10 epochs	50M	1.71×	1.3148	71.09	Baseline, s44	70.64	56.64	51.50	51.98	19.73	50.10
Looped $P=3$, 10 epochs	50M	2.45×	1.3142	–	3-seed mean	71.21	55.57	51.84	51.60	19.27	49.90
Deep (depth 12), 10 epochs	100M	1.80×	1.2979	71.86	Looped, s42	71.09	56.56	51.31	51.71	19.07	49.95
					Looped, s43	71.42	56.69	50.89	51.27	19.36	49.93
					Looped, s44	71.68	55.37	51.43	52.23	19.86	50.11
					3-seed mean	71.40	56.21	51.21	51.74	19.43	50.00

Table 2: Compute-matched comparison at seed 42, on one RTX 3060. “Time” is wall-clock training time relative to the baseline’s 4031 s. Looping is not the cheapest way to buy val_bpb: a longer schedule gets below it for less, and a model with twice the parameters gets well below it for a comparable cost. [‡]Exceeds the track’s 10-epoch limit; reported as a probe only (Appendix A.2).

is a reliable gain, and at a fixed compute budget it is not the best available move. It also does not follow that the 14-epoch run is strictly better, since its full BLiMP is flat at 71.19 against 71.27 for the same configuration at 10 epochs (Appendix A.2): val_bpb and the benchmarks come apart there too.

Why two passes and not three. A third pass buys almost nothing. $P=3$ lands at 1.3142 at seed 42, 0.0006 below $P=2$ ’s 1.3148 and well inside the seed-to-seed spread, while costing another full pass of block computation per token (Table 2). We can only speculate about why the returns fall off so sharply, and we did not run the experiments that would settle it. The reading we find most plausible is that the passes are not doing the same kind of work. The first pass has to turn embeddings into contextual representations, and the second refines representations that already exist, which is a cheaper problem and one the same weights can plausibly serve; by the third, the input to the stack is close to what the second pass already produced, and re-applying the same map to it has little left to change. The training budget also pushes in this direction. Each extra pass costs a fixed fraction of the total compute whether or not it helps, and with the data fixed at 10M words the model is only mildly underfit to begin with (Section 4.1), so there is a limited amount of loss for extra computation to remove. We fix $P=2$ for the submitted model and did not explore $P>3$.

5 Negative Results

Depth routing is dead weight, and looping does not revive it. The negative axis we explored most thoroughly is delta attention routing. We pursued it because it looked like it *should* work. We started with routing on its own, in an ordinary single for-

Table 3: Weight-tied looped depth ($P=2$) against the tuned baseline ($P=1$), official zero-shot harness, three seeds each. “Avg.” is the mean of the five columns to its left. Baseline seeds 43 and 44 are from an earlier commit (Section 4.2).

ward pass, and tried four formulations: a replacement formulation over cumulative hidden states, a delta formulation, a gated-additive variant, and the paper-faithful per-sublayer version with $2L+1$ sources, each at depth 8 and depth 12. All were neutral to negative, on val_bpb and downstream alike.

The loop should have given routing something to work with. Once the stack runs twice, a second-pass sublayer can route over the first pass’s per-sublayer deltas, which are new relative to anything in its own residual input, and no single-pass model can offer them. It did not help. val_bpb regressed at both seeds we ran, to 1.3224 against the loop’s 1.3148 at seed 42 and 1.3242 against 1.3178 at seed 43. The downstream picture was flat rather than negative: on the fast evaluation, the harness’s reduced-sample suite that we used as an in-loop screen (Appendix A), the five-task average came out at 50.52 and 51.48 against the loop’s 51.34 and 50.56. We therefore rejected the extension on val_bpb and on cost, not on the benchmarks.

The learned router weights show why. Figure 2 plots them, and they collapse. Almost all of the routing mass concentrates on the token embedding emb, which is the residual-and-embedding skip the model already carries, plus one early source from within the same pass. The cross-pass deltas that the loop makes newly available, the whole justification for the extension, receive almost no weight. Given the choice, the router declines to use them and re-derives the skip it already had.

We offer one reading of this, and we offer it as a hypothesis: we have not established it. The baseline may have already saturated the dimension the routing operates in. It carries ResFormer-style per-token value residuals (Zhou et al., 2025), in the

form of the value-embedding mechanism of Section 3.1, and a learned residual-and-embedding mix at every block. Much of what a routed mix over sub-layer deltas could contribute may already be reachable through those two paths, and the router’s preference for exactly the skip that the scalars provide is what that account predicts. We have one piece of supporting evidence and it is indirect: removing the value-embedding tables from the looped model costs 0.0096 val_bpb and drops full BLiMP from 71.09 to 70.55 at seed 42, so that path is doing real work in the model. The experiment that would actually test the hypothesis is delta routing on a baseline stripped of value embeddings and of the embedding skip, and we did not run it. A baseline without those paths might well respond to routing differently, and we do not claim otherwise.

6 Conclusion

At the 10M-word budget, applying the same block stack twice per forward pass lowers val_bpb at every seed we ran, at zero additional parameters, on top of a baseline we had already tuned hard. That direction is the opposite of what recent at-scale work reports, where looping buys reasoning and gives up perplexity. Our reading is that the data budget accounts for the difference: a model that is mildly underfit on a fixed corpus can still spend computation on quality, while a compute-optimal one cannot. The gain is real but small, and it is not free. It costs $1.71\times$ the training time, and at comparable cost a model with twice the parameters does better (Section 4.2). Looping buys parameter efficiency, not compute efficiency, and we would not recommend it to anyone whose constraint is compute rather than parameters.

Delta attention residual routing, the other architectural direction we pursued, was neutral or worse everywhere we tried it (Section 5).

Two directions follow from the positive result. Exact weight tying already helps, so a small per-pass parameter budget (Bae et al., 2025) might let the two passes specialize into distinct functions at close to zero parameter cost. And because the second pass pays broadly while the third barely pays at all, per-token adaptive depth, in the spirit of the dynamic halting of Universal Transformers (Dehghani et al., 2019), would spend the extra computation only where it earns its keep.

On the methodology: the human-directed agentic loop was a useful way to keep a single GPU

busy, and its mechanical accept rule made the failures legible after the fact. Its limitation showed up on the bidirectional objective, where the accept metric and the target evaluations disagreed and a human had to override the rule. Searches of this kind should either choose an acceptance metric that tracks the target evaluations or score against those evaluations directly. We release the code and the full training and evaluation logs.¹

Limitations

Our study has several limitations.

Few seeds, one GPU. The headline looped-depth result replicates across three seeds, but most other configurations were trained once or twice. We also chose the ~ 0.005 val_bpb accept threshold by hand: it is a guess at the run-to-run noise and we never measured that noise directly, so improvements near the threshold should be read as suggestive. Every run is on one consumer GPU (RTX 3060, 12 GB), which bounded the batch and model sizes we could explore at all and caps the seed budget at three.

The currency here is compute. The looped model costs $1.71\times$ the baseline’s training time at the same parameter count. Two cheaper uses of that compute do at least as well on val_bpb: a longer schedule for the same model and a larger model at comparable cost (Table 2). We restrict the claim to parameter efficiency for that reason. Looping is not the best way to spend a fixed compute budget at this scale, and our own measurements are what say so.

Single-seed negative results. Most rejected hypotheses were evaluated at one or two seeds. We reject them because they failed to beat the baseline beyond the noise threshold at the point they were tried, not because we have excluded a smaller effect that a larger seed budget could resolve. One result deserves a specific flag: the fast-eval BLiMP of 72.46 from cross-pass routing (Table 5) is the highest BLiMP number in the project, and it is a single draw that falls to 71.20 at the next seed. We do not read it as evidence that the extension helps. We also cannot fully exclude a BLiMP-specific effect that a val_bpb-driven accept rule would never have kept.

¹Code and experiment logs: <https://github.com/quesswho/looped-transformer>.

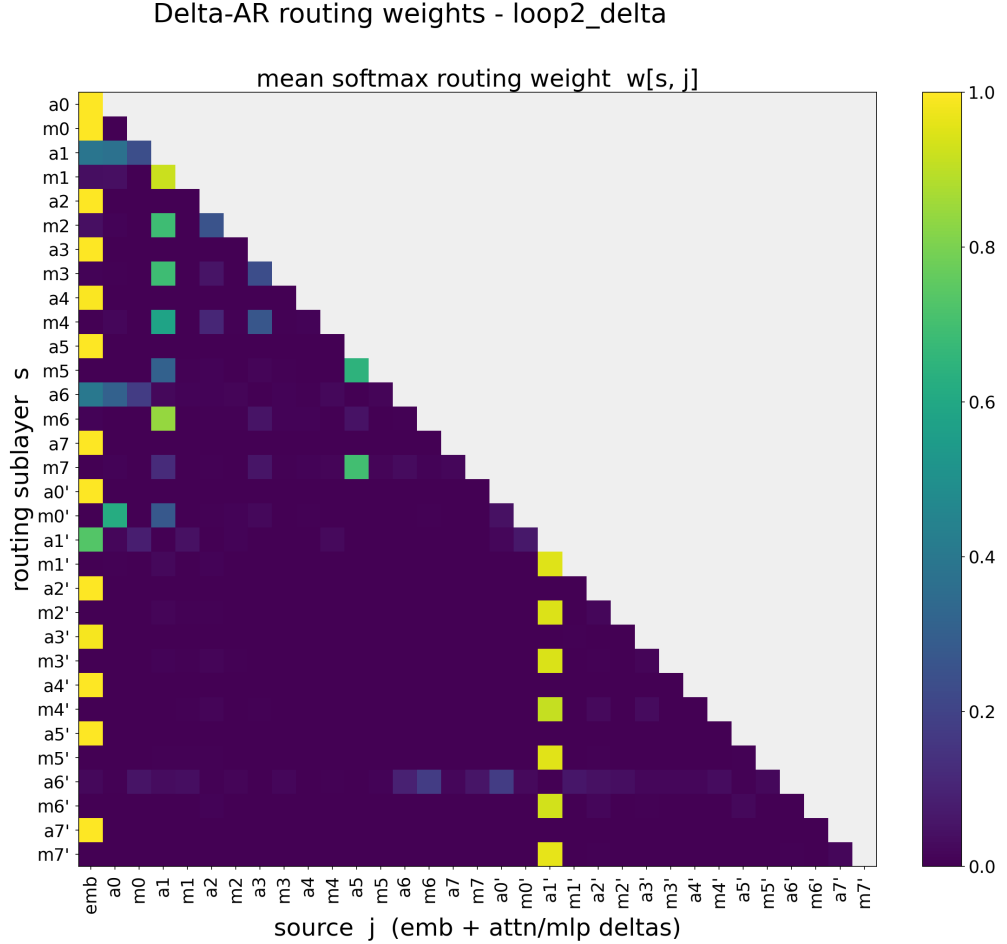


Figure 2: Mean softmax routing weights $w[s, j]$ for the cross-pass delta-routing variant (loop2_delta). Rows are the $2L$ routing sublayers of each of the two passes; columns are the available sources (emb plus each earlier attention/MLP delta). Despite having the first pass’s per-sublayer deltas available, the router collapses onto the embedding skip emb and one early attention delta from within the same pass (a1 in the first pass, a1’ in the second), leaving the first pass’s deltas near zero in the primed rows. The routing re-derives the model’s existing skip rather than exploiting the new information the loop exposes.

Greedy, history-dependent search. The loop is greedy. It keeps any change that beats the current best and reverts the rest, so the final configuration is path-dependent. A technique rejected early, residual dropout or added depth for instance, might have helped on top of a later configuration, and the order in which we explored the changes conditions the outcome. Table 1 traces one path through the space; it is not an ablation of the final model, and no row there should be read as the isolated contribution of that change.

Proxy-metric optimization. The loop optimizes val_bpb and not the downstream benchmarks. The masked-LM objective is a case where the two came apart visibly enough for a human to catch it. Other divergences may exist that neither the loop nor our out-of-loop evaluation saw, particularly among

changes the loop rejected on val_bpb without ever scoring them downstream.

Scope of conclusions. Our claims concern BabyLM Strict-Small: 10M words, $\sim 50M$ parameters, this corpus and this tokenizer. Whether any of it transfers to other data scales, model sizes, or corpora is untested, and establishing that would require replication we did not have the compute to run.

References

Sangmin Bae, Adam Fisch, Hrayr Harutyunyan, Ziwei Ji, Seungyeon Kim, and Tal Schuster. 2025. [Relaxed recursive transformers: Effective parameter sharing with layer-wise LoRA](#). In *International Conference on Learning Representations*.

- Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. 2019. [Deep equilibrium models](#). In *Advances in Neural Information Processing Systems*, pages 688–699.
- Lucas Charpentier, Leshem Choshen, Ryan Cotterell, Mustafa Omer Gul, Michael Y. Hu, Jing Liu, Jaap Jumelet, Tal Linzen, Aaron Mueller, Candace Ross, Raj Sanjay Shah, Alex Warstadt, Ethan Gotlieb Wilcox, and Adina Williams. 2025. [Findings of the third BabyLM challenge: Accelerating language modeling research with cognitively plausible data](#). In *Proceedings of the First BabyLM Workshop*, pages 399–420, Suzhou, China. Association for Computational Linguistics.
- Lucas Georges Gabriel Charpentier and David Samuel. 2024. [GPT or BERT: why not both?](#) In *Proceedings of the 2nd BabyLM Challenge at the 28th Conference on Computational Natural Language Learning*.
- Leshem Choshen, Ryan Cotterell, Mustafa Omer Gul, Jaap Jumelet, Tal Linzen, Aaron Mueller, Suchir Salhan, Raj Sanjay Shah, Alex Warstadt, and Ethan Gotlieb Wilcox. 2026. [BabyLM turns 4 and goes multilingual: Call for papers for the 2026 BabyLM workshop](#). *Preprint*, arXiv:2602.20092.
- DeepSeek-AI, Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, and 181 others. 2025. [DeepSeek-V3 technical report](#). *Preprint*, arXiv:2412.19437.
- Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. 2019. [Universal transformers](#). In *International Conference on Learning Representations*.
- Jonas Geiping, Sean McLeish, Neel Jain, John Kirchenbauer, Siddharth Singh, Brian R. Bartoldson, Bhavya Kailkhura, Abhinav Bhatele, and Tom Goldstein. 2025. [Scaling up test-time compute with latent reasoning: A recurrent depth approach](#). In *Advances in Neural Information Processing Systems*.
- Angeliki Giannou, Shashank Rajput, Jy yong Sohn, Kangwook Lee, Jason D. Lee, and Dimitris Papailiopoulos. 2023. [Looped transformers as programmable computers](#). In *Proceedings of the 40th International Conference on Machine Learning*, pages 11398–11442.
- Fabian Gloeckle, Badr Youbi Idrissi, Baptiste Rozière, David Lopez-Paz, and Gabriel Synnaeve. 2024. [Better & faster large language models via multi-token prediction](#). In *Proceedings of the 41st International Conference on Machine Learning*.
- 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, and 3 others. 2022. [Training compute-optimal large language models](#). In *Advances in Neural Information Processing Systems*.
- Anna A. Ivanova, Aalok Sathe, Benjamin Lipkin, Unnathi Kumar, Setayesh Radkani, Thomas H. Clark, Carina Kauf, Jennifer Hu, R. T. Pramod, Gabriel Grand, Vivian Paulun, Maria Ryskina, Ekin Akyürek, Ethan Wilcox, Nafisa Rashid, Leshem Choshen, Roger Levy, Evelina Fedorenko, Joshua Tenenbaum, and Jacob Andreas. 2025. [Elements of world knowledge \(EWoK\): A cognition-inspired framework for evaluating basic world knowledge in language models](#). *Transactions of the Association for Computational Linguistics*, 13:1245–1270.
- Pavel Izmailov, Dmitrii Podoprikin, Timur Garipov, Dmitry Vetrov, and Andrew Gordon Wilson. 2018. [Averaging weights leads to wider optima and better generalization](#). In *Proceedings of the 34th Conference on Uncertainty in Artificial Intelligence*.
- Keller Jordan, Yuchen Jin, Vlado Boza, Jiacheng You, Franz Cesista, Laker Newhouse, and Jeremy Bernstein. 2024. [Muon: An optimizer for hidden layers in neural networks](#).
- Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. [Scaling laws for neural language models](#). *Preprint*, arXiv:2001.08361.
- Andrej Karpathy. 2025a. autoresearch. <https://github.com/karpathy/autoresearch>.
- Andrej Karpathy. 2025b. nanochat. <https://github.com/karpathy/nanochat>.
- Najoung Kim and Sebastian Schuster. 2023. [Entity tracking in language models](#). In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3835–3855, Toronto, Canada. Association for Computational Linguistics.
- Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2020. [ALBERT: A lite BERT for self-supervised learning of language representations](#). In *International Conference on Learning Representations*.
- Cheng Luo, Zefan Cai, and Junjie Hu. 2026. [Delta attention residuals](#). *Preprint*, arXiv:2605.18855.
- Kanishka Misra, Julia Rayz, and Allyson Ettinger. 2023. [COMPS: Conceptual minimal pair sentences for testing robust property knowledge and its inheritance in pre-trained language models](#). In *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics*, pages 2928–2949, Dubrovnik, Croatia. Association for Computational Linguistics.

- Niklas Muennighoff, Alexander M. Rush, Boaz Barak, Teven Le Scao, Aleksandra Piktus, Nouamane Tazi, Sampo Pyysalo, Thomas Wolf, and Colin Raffel. 2023. [Scaling data-constrained language models](#). In *Advances in Neural Information Processing Systems*.
- Matteo Pagliardini, Amirkeivan Mohtashami, Francois Fleuret, and Martin Jaggi. 2024. [DenseFormer: Enhancing information flow in transformers via depth weighted averaging](#). In *Advances in Neural Information Processing Systems*.
- Nikunj Saunshi, Nishanth Dikkala, Zhiyuan Li, Sanjiv Kumar, and Sashank J. Reddi. 2025. [Reasoning with latent thoughts: On the power of looped transformers](#). In *International Conference on Learning Representations*.
- Noam Shazeer. 2020. [GLU variants improve transformer](#). *Preprint*, arXiv:2002.05202.
- Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. 2024. [RoFormer: Enhanced transformer with rotary position embedding](#). *Neurocomputing*, 568:127063.
- Pablo Villalobos, Anson Ho, Jaime Sevilla, Tamay Besiroglu, Lennart Heim, and Marius Hobbhahn. 2024. [Will we run out of data? limits of LLM scaling based on human-generated data](#). In *Proceedings of the 41st International Conference on Machine Learning*.
- Sami Virpioja, Peter Smit, Stig-Arne Grönroos, and Mikko Kurimo. 2013. [Morfessor 2.0: Python implementation and extensions for morfessor baseline](#). Technical Report 25/2013, Aalto University, Helsinki, Finland.
- Alex Warstadt, Alicia Parrish, Haokun Liu, Anhad Mohananey, Wei Peng, Sheng-Fu Wang, and Samuel R. Bowman. 2020. [BLiMP: The benchmark of linguistic minimal pairs for english](#). *Transactions of the Association for Computational Linguistics*, 8:377–392.
- Zhanchao Zhou, Tianyi Wu, Zhiyun Jiang, Fares Obeid, and Zhenzhong Lan. 2025. [Value residual learning](#). In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*.
- Defa Zhu, Hongzhi Huang, Zihao Huang, Yutao Zeng, Yunyao Mao, Banggu Wu, Qiyang Min, and Xun Zhou. 2025a. [Hyper-connections](#). In *International Conference on Learning Representations*.
- 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 Smith, Xingwei Qu, Mude Hui, Bohong Wu, and 14 others. 2025b. [Scaling latent reasoning via looped language models](#). *Preprint*, arXiv:2510.25741.

A Full Evaluation Results

The main text reports a small set of numbers from the loop’s overall record; the entire record is given here. Both evaluation modes come from the official BabyLM evaluation pipeline: *FULL* scores all 67 BLiMP paradigms and the complete supplement/EWoK/COMPS/entity sets (Table 4), while *fast* is the pipeline’s fast suite, whose smaller per-task samples are intended for intermediate checkpoints; we used it as the in-loop screen, and it is correspondingly noisier (Table 5). It does not cover COMPS, so five-task fast averages quoted in the text take COMPS from the matching FULL evaluation. Entity tracking varies by only a few points across the configurations in this appendix and is essentially at chance for every variant we tried; it is nonetheless included in the main-text average (Section 4.2). The reading-time correlations are near zero for every run and are not tabulated here.

A.1 Fast-eval exploratory and rejected runs (higher variance)

Table 5 lists every in-loop fast evaluation; these numbers are comparable only within this table. The looped model ($P=2$) beats the seed-matched baseline at seed 42 (51.34 vs. 48.91) and ties at seed 43 (50.56 vs. 50.70); the delta-routing run’s project-high 72.46 BLiMP at seed 42 falls to 71.20 at seed 43, which is why the main text rests nothing on it.

A.2 Other probes

Several measurements do not fit the zero-shot table and are summarized here.

GLUE fine-tuning. Fine-tuning the 50M MTP model for 10 epochs gives an accuracy-average of 57.7 across the BabyLM GLUE tasks (BoolQ 64.0, MNLI 35.7 $\approx$ chance, MRPC 68.1, MultiRC 57.6, QQP 62.8, RTE 54.0, WSC 61.5), with an official mixed-metric average of 42.3 (F1 collapses to 0 on MultiRC/QQP). These sit near majority/chance baselines, as expected for a 50M-parameter model on 10M words: competence here is zero-shot, not supervised.

Extended-budget probe (14 epochs). To test the underfitting claim of Section 4.1 directly, we re-ran the $P=1$ baseline with the budget extended from 10 to 14 epochs, which exceeds the track’s 10-epoch limit and is therefore a probe only, stretching every

Configuration	val_bpb	BLiMP	Supp.	EWoK	COMPS	Ent.
<i>50M base ($d=512$, depth 8)</i>						
MTP ($K=2$) baseline	1.3173	70.45	55.35	52.05	51.91	17.96
SwiGLU causal baseline	1.3161	69.32	54.64	51.20	51.42	18.12
+ MLM (15%) [<i>arch baseline</i>]	1.3221	70.15	56.15	52.16	51.56	18.57
Baseline $P=1$, seed 42	1.3215	71.27	54.69	51.42	51.67	18.91
Baseline $P=1$, seed 43	1.3202	71.72	55.37	52.59	51.16	19.16
Baseline $P=1$, seed 44	1.3249	70.64	56.64	51.50	51.98	19.73
<i>Scale (wider / deeper), +MLM 15%</i>						
70M wide ($d=640$, depth 8)	1.3088	71.16	54.91	52.62	51.93	18.59
85M deep ($d=640$, depth 10) s42	1.3171	71.76	56.54	50.94	51.68	19.20
depth 10, seed 43	–	71.39	54.59	50.25	51.43	18.11
100M deep ($d=640$, depth 12) s42	1.2979	71.86	57.04	53.01	51.91	17.47
depth 12, seed 43	–	71.58	56.12	51.68	51.52	19.73
<i>Looped depth and routing (50M)</i>						
Looped $P=2$, seed 42	1.3148	71.09	56.56	51.31	51.71	19.07
Looped $P=2$, seed 43	1.3178	71.42	56.69	50.89	51.27	19.36
Looped $P=2$, seed 44	1.3199	71.68	55.37	51.43	52.23	19.86
Delta-AR (gated-additive)	1.3237	70.23	55.80	50.93	51.34	–
Delta-AR (per-sublayer, gated)	–	70.28	56.19	52.69	51.68	–
AttnRes DenseFormer routing	1.540	61.94	53.17	50.65	50.67	–

Table 4: All FULL evaluations, in discovery order. Accuracies are percentages; val_bpb lower is better. “–” = not measured.

progress-based schedule (learning rate, weight decay, sequence-length warmup, MTP) over the new budget. val_bpb improves from 1.3215 (baseline re-run) to 1.3140 (-0.0076), with the per-epoch validation curve flattening around epochs 11–12: the model is still gaining from optimization at 10 epochs, i.e. mildly underfit. The 14-epoch value is also below each of the three looped draws at 10 epochs (Section 4.2), yet its full BLiMP is flat (71.19 against 71.27 for the same configuration at 10 epochs): another case of val_bpb moving without the benchmarks.

B Hyperparameters

Table 6 lists the full configuration of the submitted looped model. The $P=1$ baseline is identical except for the number of block passes. All schedules are driven by training progress, defined as completed epochs divided by the 10-epoch budget.

Configuration (fast eval)	val_bpb	BLiMP	Supp.	EWoK	Ent.
<i>Baseline trajectory</i>					
Best original-data baseline	1.3266	69.03	57.20	51.09	19.37
+ sequence-length warmup	1.3242	69.17	55.60	51.73	21.15
+ MTP ($K=2$, “all”)	1.3173	70.93	58.00	52.27	20.21
MTP $K=3$	–	70.87	56.80	52.18	19.56
<i>GPT-BERT masked-LM sweep (causal backend)</i>					
SwiGLU causal baseline	1.3161	69.59	56.40	52.09	18.64
+ MLM 5%	1.3156	70.20	56.40	53.18	17.66
+ MLM 15%	1.3209	70.54	57.20	55.36	20.44
15% rerun, seed 42	–	71.21	54.00	51.64	20.17
+ MLM 25%	1.3293	70.69	50.80	53.64	18.91
<i>Tokenizer granularity (Morfessor vocab)</i>					
vocab 4096 (finer)	–	70.81	54.40	50.18	19.72
vocab 8192 (default)	1.3173	70.93	58.00	52.27	20.21
vocab 16384 (coarser)	1.3147	69.40	56.40	50.00	19.49
<i>Scale / context / data variants</i>					
70M wide, MLM 0	1.306	71.08	56.80	50.64	19.03
50M, context cap 512	1.387	70.51	56.00	52.18	19.22
Corpus clean + refill	–	69.25	54.00	51.55	19.96
Corpus clean + dedup + refill	–	69.31	56.80	52.45	20.17
Corpus finalized (<10M)	–	69.61	57.20	52.82	18.77
Register tilt (written 56%)	–	69.31	55.20	51.82	19.53
<i>$P=1$ baseline, looped depth, and delta-routing family</i>					
Baseline $P=1$, seed 42	1.3215	71.38	53.60	49.00	18.88
Baseline $P=1$, seed 43	1.3202	71.74	57.20	53.00	20.41
Baseline $P=1$, seed 44	1.3249	71.17	56.00	51.91	20.39
Looped $P=2$, seed 42	1.3148	71.78	59.20	52.73	21.29
Looped $P=2$, seed 43	1.3178	71.53	58.00	52.18	19.80
Looped + delta route, seed 42	1.3224	72.46	56.40	52.64	19.65
Looped + delta route, seed 43	1.3242	71.20	61.60	52.45	20.04
Looped + delta route v2 (carry)	1.3437	69.14	–	–	–
Delta-AR per-sublayer (paper)	1.3255	70.96	59.60	51.18	19.98
Looped + curriculum	1.4047	70.59	–	–	–
Looped, VE ablation (33M)	1.324	70.83	–	–	–
<i>100M depth-12, fast eval</i>					
100M deep, seed 42	1.2979	72.27	58.80	52.27	18.28
100M deep, seed 43	1.2985	72.07	56.00	50.55	20.46

Table 5: All fast evaluations. “–” = not measured.

Setting	Value
<i>Architecture</i>	
Layers L / block passes P	8 / 2
Model dim d / heads	512 / 4
Head dim	128
MLP / norm / positions	SwiGLU / RMSNorm / RoPE
Attention window	full causal, every layer
Value-embedding tables	every 2nd layer + last
Vocabulary (Morfessor)	8192, byte fallback
Context length	2048
<i>Muon (hidden matrices)</i>	
Learning rate	0.10
Momentum (Nesterov)	0.85→0.95 over 300 steps
Newton-Schulz / β_2	5 / 0.95
Weight decay	0.2, constant
<i>AdamW (embeddings, scalars)</i>	
Token-embedding LR	0.6
Unembedding LR	0.004
Per-layer scalar LR	0.5
(β_1, β_2)	(0.8, 0.95)
Value-embedding WD	0.1, decoupled
<i>Budget and schedules</i>	
Epochs	10
Tokens per optimizer step	2^{19} ($\approx 524K$)
Device batch / accum.	$8 \times 2048 / 32$
LR schedule	constant, then linear to 0
over final	50% of training
Sequence-length warmup	256→2048 over first 50%
SWA start (uniform avg.)	60% of training
Multi-token prediction	$K=2$, aux. weight 0.3
Masked-LM microbatches	15%, 30% tokens masked
Seeds	42, 43, 44

Table 6: Full configuration of the submitted looped model ($P=2$); the $P=1$ baseline differs only in the number of block passes.