ASYMMETRIC LOOPED TRANSFORMERS: FIXED GLOBAL REPRESENTATIONS FOR AN ITERATIVE DECODER

Anonymous authors
Paper under double-blind review

ABSTRACT

Looped Transformers increase effective depth by repeatedly applying shared layers, offering a way to scale computation without increasing parameter count. However, full-stack looping places context construction and iterative state refinement within the same recurrent computation, without explicitly distinguishing their roles. We introduce the **Asymmetric Looped Transformer (AsymLoop)**, which separates single-pass context encoding from recurrent state refinement. A single-pass causal encoder produces layer-specific Global key–value (KV) caches that remain fixed across iterations, while a weight-tied recurrent decoder generates Local KV from its evolving state. Joint attention over both sources under a single softmax allows each update to integrate persistent contextual information with the current intermediate state. For training, we apply a heavy-tail-guided layerwise learning-rate method to the encoder and decoder. Experiments on FineWeb-Edu across four model scales, from 60M to 1B, show consistently lower validation perplexity and higher average downstream accuracy than Full Loop under matched effective depth and training token budgets. At 1B, AsymLoop improves average zero-shot accuracy across seven commonsense reasoning benchmarks from 49.58% to 50.85%. Further experiments extend these benefits to longer training budgets, an OLMo-style backbone, and sparse mixture-of-experts models. These results support asymmetric context encoding and state refinement as an effective design for recurrent language models.

1 INTRODUCTION

Scaling language models typically involves increasing parameters, training data, and computation together (Henighan et al., 2020). Looped Transformers offer another dimension for scaling computation: repeatedly applying a shared block increases effective depth without assigning independent parameters to every executed layer (Dehghani et al., 2019; Lan et al., 2020). Recent work further demonstrates that recurrent depth can improve reasoning performance through additional computation at test time (Geiping et al., 2025). This opportunity makes the organization of recurrence a central design concern. Performance depends not only on how many iterations a model executes, but also on which computations it repeats and how contextual information remains available across iterations.

Recurrence can be organized through different layer topologies, depending on which layers are repeated and which non-recurrent layers precede or follow them (Huang et al., 2026). A straightforward choice is to loop the entire Transformer stack, feeding the output of each pass back into the same stack for the next (Zhu et al., 2025); we refer to this topology as *Full Loop*. However, analyses of pretrained language models suggest that different depths support distinct computational roles, including contextualizing input tokens, refining features, and forming predictions (Lad et al., 2025). Repeating the entire stack places these roles together within the recurrent computation, without explicitly separating input processing from iterative refinement. Analyses of looped models likewise observe multiple feedforward-like processing stages recurring within each iteration (Blayney et al., 2026). These findings motivate organizing recurrence around a division of roles: distinguishing context construction that can be performed once from state refinement to which additional computation can be allocated.

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.

Partial looping has been explored in recurrent language models, repeating only a selected subset of layers while executing the remaining layers once (Geiping et al., 2025; Huang et al., 2026). Controlled comparisons with matched parameter counts, effective depth, and training budgets report improvements over full-stack looping on several multi-step reasoning benchmarks (Huang et al., 2026). Such designs make the distinction between recurrent and non-recurrent computation explicit, but choosing the scope of recurrence alone does not determine how the two components should work together. We argue that their different computational roles should guide a broader *asymmetric design*: non-recurrent layers can specialize in context construction, while recurrent layers perform iterative state updates, with information exchange connecting these two types of computation.

We introduce the **Asymmetric Looped Transformer (AsymLoop)**, which connects a single-pass causal encoder to a weight-tied recurrent decoder through joint attention to fixed Global KV and evolving Local KV. A causal encoder runs once and projects its output into per-decoder-layer Global key–value (KV) caches. These caches serve as contextual anchors that remain fixed across all iterations of a forward pass. A weight-tied decoder generates Local KV from its current hidden states at each iteration. The two sources offer complementary views: Global KV preserves access to the context encoded by the encoder, while Local KV reflects the decoder's current intermediate state. Queries allocate attention jointly across both sources under a single softmax, enabling each update to integrate global contextual information with the current local state. All components are trained end to end.

For optimization, we adopt the heavy-tail-guided layerwise learning-rate method of He et al. (2026) for the encoder and decoder. The same spectral allocation rule is applied to matrices in both components. Encoder–decoder learning-rate differences arise from their measured spectral statistics. We use Asymmetric Learning Rate Schedule (ALRS) as a shorthand for this training configuration and examine the resulting allocation patterns and their empirical effects in our experiments.

Our main contributions are:

- **An asymmetric recurrent architecture.** We jointly design one-time context encoding and weight-tied recurrent updates, connecting them through joint attention to fixed Global KV and dynamic Local KV.
- **Analysis of context access under recurrence.** Through component ablations and comparisons across recurrent depths, we examine the roles of fixed Global KV and evolving Local KV in recurrent state refinement.
- **Performance gains across model sizes.** Across four model sizes, AsymLoop achieves lower validation perplexity and higher average zero-shot accuracy than Full Loop under matched training budgets and effective depth. Perplexity improves by 0.98 and 0.30 at 135M and 350M, respectively, while the seven-task average accuracy at 1B rises from 49.58% to 50.85%.

2 Related Work

Looped Transformers and Recurrent Depth. Weight-tied recurrence in Transformers traces back to Universal Transformers (Dehghani et al., 2019) and ALBERT (Lan et al., 2020), which established that parameter sharing across layers can achieve competitive linguistic representations. Theoretical investigations have formalized looped Transformers as programmable computers (Giannou et al., 2023) capable of learning complex in-context algorithms (Yang et al., 2024) and latent thoughts (Saunshi et al., 2025). More recently, scaling up test-time compute through recurrent depth has emerged as a promising alternative to chain-of-thought token generation (Geiping et al., 2025). Studies such as Parcae (Prairie et al., 2026) and FLT (Fu et al., 2026) explore stability in looped models. Beyond full-stack recurrence, partial-loop architectures already separate recurrent computation from non-recurrent input and output processing (Geiping et al., 2025). Our focus is the information exchange between these components: AsymLoop provides the recurrent decoder with layer-specific Global KV from a single-pass encoder, jointly accessed with evolving Local KV at every iteration.

Causal Encoder-Decoder Architectures. Standard autoregressive language modeling typically employs decoder-only stacks. YOCO lets upper layers reuse KV produced by lower causal layers (Sun et al., 2024). The Causal Encoder-Decoder (CED) proposed in DeepSeek-V4.1-Flash (DeepSeek-AI et al., 2026) divides the Transformer into an encoder that generates a static

Global KV cache once, and a decoder that attends to this cache alongside standard causal dependencies. While CED demonstrated KV cache savings during inference, it was conceived as a non-recurrent ($K = 1$) feedforward model. Their asymmetry also suggests a way to organize recurrent language models: build a causal representation of the input once, then let a recurrent stack iteratively refine its state while retaining access to that representation.

Heavy-Tailed Self-Regularization and Layerwise Optimization. Heavy-Tailed Self-Regularization (HT-SR) characterizes weight spectra through fitted power-law tails (Martin & Mahoney, 2021; Martin et al., 2021). He et al. (2026) use these spectral exponents to allocate learning rates across layers and modules. We apply their allocation rule to the non-recurrent encoder and the weight-tied decoder. Encoder parameters are used once per forward pass, whereas decoder parameters are reused across iterations and accumulate gradient contributions from each use. This difference motivates allowing their learning rates to vary.

3 The Asymmetric Looped Architecture

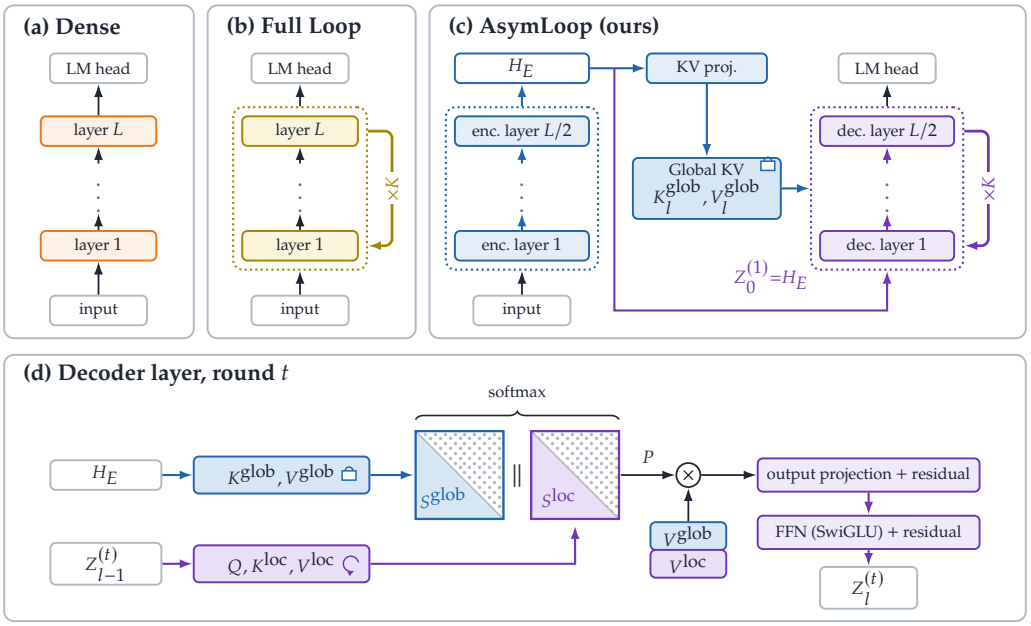

Figure 1: **AsymLoop architecture and Global–Local attention.** **(a)** Dense executes each layer once. **(b)** Full Loop repeats the entire stack with shared parameters. **(c)** AsymLoop runs a causal encoder once and a shared-parameter decoder for K iterations. The encoder output provides layer-specific Global KV that remains fixed across iterations. **(d)** Each decoder layer jointly attends to fixed Global KV and Local KV computed from its current state, using a single softmax with causal masking for both sources.

3.1 Asymmetric Encoder–Decoder Design

As shown in Figure 1, AsymLoop consists of a non-recurrent causal encoder and a weight-tied recurrent decoder. The encoder uses the same causal Transformer blocks as a standard decoder-only language model, without changing their architecture. We split L physical layers into E encoder layers and D decoder layers, with $E = D = L/2$ in our experiments. A single forward pass through the encoder produces contextual representations $H_E \in \mathbb{R}^{N\times d}$, where N is the sequence length and d is the hidden dimension. The encoder output serves as the input to the first decoder iteration. Separate projection matrices for each decoder layer map this output into layer-specific Global KV caches, which are reused across all K iterations.

The decoder runs for K iterations, feeding the output of each iteration into the next. Denote its D layers by $\mathcal{D}_1, \ldots, \mathcal{D}_D$, and the Global KV for layer $\mathcal{D}_l$ by $\mathcal{G}_l$, defined below. At iteration t, $X^{(t)}$ denotes the input to the entire decoder, and $Z_l^{(t)}$ denotes the output of decoder layer $\mathcal{D}_l$. Thus, $Z_D^{(t)}$ is the output of the final decoder layer $\mathcal{D}_D$ and of the entire decoder at that iteration. The first iteration receives the encoder output; each subsequent iteration receives the preceding iteration's decoder output. Within each iteration, the first layer processes $X^{(t)}$, and the remaining layers process the preceding layer's output:

$$\begin{aligned} X^{(1)} &= H_E, \qquad X^{(t)} = Z_D^{(t-1)} \quad (t = 2, \ldots, K), \\ Z_1^{(t)} &= \mathcal{D}_1\Big(X^{(t)}; \mathcal{G}_1\Big), \\ Z_l^{(t)} &= \mathcal{D}_l\Big(Z_{l-1}^{(t)}; \mathcal{G}_l\Big) \qquad\qquad (l = 2, \ldots, D). \end{aligned} \tag{1}$$

Each $\mathcal{D}_l$ has distinct parameters shared across all K iterations. The final state $Z_D^{(K)}$ is used for next-token prediction, and the model is trained end to end through all iterations.

3.2 Global–Local Fusion via Joint Attention

Each decoder layer attends to two sources: fixed Global KV from the encoder and evolving Local KV from the current decoder state. For layer l, Global KV is obtained using layer-specific projections:

$$\bar{K}_l^{\text{glob}} = H_E W_l^{K,\text{glob}}, \qquad V_l^{\text{glob}} = H_E W_l^{V,\text{glob}}. \tag{2}$$

At each iteration, the layer projects its current RMS-normalized input into queries and Local KV. Before attention, we apply QKNorm (Henry et al., 2020), followed by RoPE, to queries and both Global and Local keys. The resulting Global keys and values form $\mathcal{G}_l = (K_l^{\text{glob}}, V_l^{\text{glob}})$, which remains fixed across all K iterations of a forward pass; queries and Local KV are recomputed at each iteration.

Each attention head reads both KV sources under a single softmax. Omitting layer and iteration indices, the attention output is

$$A = \text{softmax}\left(\frac{[Q(K^{\text{glob}})^\top \mid Q(K^{\text{loc}})^\top]}{\sqrt{d_h}} + [M \mid M]\right) \begin{bmatrix} V^{\text{glob}} \\ V^{\text{loc}} \end{bmatrix}, \tag{3}$$

where d_h is the head dimension and M is the causal mask. The two sources are concatenated along the key-position axis; a query at position i can access only positions $j \le i$ in either source. The shared softmax allocates attention between encoder context and the current decoder state. Standard output projection, residual connections, and FFN follow.

3.3 Application of Spectral Layerwise Learning Rates

We use the layerwise learning-rate method of He et al. (2026) to train AsymLoop. The method uses spectral exponents from Heavy-Tailed Self-Regularization (HT-SR) (Martin & Mahoney, 2019) to assign learning rates to individual weight matrices in the encoder and decoder under the same allocation rule.

For a weight matrix W_m, let $\mathcal{T}_m$ contain the largest half of its squared singular values and let $\lambda_{m,\min}$ be the smallest value in this tail. At refresh step s, we estimate the exponent of the power-law density $p(\lambda) \propto \lambda^{-\alpha}$ using

$$\alpha_m(s) = 1 + \frac{|\mathcal{T}_m|}{\sum_{\lambda \in \mathcal{T}_m} \log(\lambda / \lambda_{m,\min})}. \tag{4}$$

Following the allocation rule of He et al. (2026), larger exponents receive larger learning-rate multipliers. With $\alpha_{\min}(s)$ and $\alpha_{\max}(s)$ denoting the extrema across the attention, FFN, and Global KV projection matrices, the target learning rate is

$$\eta_m^{\text{target}}(u; s) = \eta_{\text{base}}(u) \left[1 + 4 \frac{\alpha_m(s) - \alpha_{\min}(s)}{\alpha_{\max}(s) - \alpha_{\min}(s)}\right], \tag{5}$$

where $\eta_{\text{base}}(u)$ is the common warmup/cosine schedule at training progress u. This maps the exponents to multipliers in $[1, 5]$. Following He et al. (2026), the input embedding and output head use the upper multiplier; normalization parameters retain the base schedule.

Following He et al. (2026), ALRS periodically updates the learning-rate multipliers during the first 20% of training, then holds them fixed while the base learning-rate schedule continues. Scheduling details are provided in Appendix B.1.

4 Experiments

We evaluate AsymLoop on LLaMA-style architectures at the 60M, 135M, 350M, and 1B scales following He et al. (2026). Section 4.1 defines the baselines, training budgets, and effective-depth comparisons, with full configurations and hardware details in Appendix A. We then report pretraining results across model sizes (Section 4.2), downstream performance (Section 4.3), the effect of recurrent depth (Section 4.4), ablation studies (Section 4.5), and transfer to an OLMo-style backbone (Appendix G).

4.1 Experimental Setup

Pre-training. All model families in the main experiments use the LLaMA-style backbone configurations following He et al. (2026) and are trained from scratch on FineWeb-Edu (Penedo et al., 2024). To study the scaling behavior of AsymLoop, we evaluate the 60M, 135M, 350M, and 1B configurations. Training token budgets follow the Chinchilla scaling law (Hoffmann et al., 2022), which prescribes approximately 20 training tokens per parameter, giving 1.68B, 3B, 7B, and 20B tokens at the four model sizes. Detailed data splits, model configurations, and training budgets are provided in Appendix A.

Baselines and depth matching. We compare AsymLoop with two vanilla baselines: a standard non-recurrent Transformer (*Dense*), and a homogeneous looped Transformer that repeats the entire Transformer stack with weights shared across iterations (*Full Loop*). We compare recurrent models at matched effective depth, defined as the total number of Transformer layers executed per forward pass. The additional parameters and computation introduced by Global KV are detailed in Appendix D.

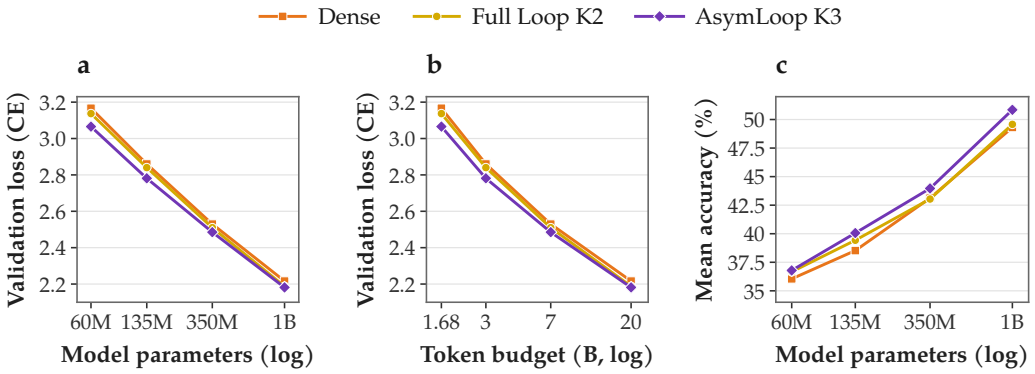

Figure 2: **Validation and zero-shot performance across model sizes.** Full Loop ($K = 2$) and AsymLoop ($K = 3$) have the same effective depth. **(a, b)** Validation cross-entropy versus model parameter count and training token budget, respectively. **(c)** Average zero-shot accuracy across seven tasks.

Evaluation. We report validation cross-entropy and perplexity on held-out FineWeb-Edu data to assess language modeling performance. We further evaluate the pretrained models on seven commonsense reasoning benchmarks under zero-shot and five-shot settings: PIQA (Bisk et al., 2020), Social IQa (Sap et al., 2019), HellaSwag (Zellers et al., 2019), WinoGrande (Sakaguchi et al., 2021), ARC-Challenge (ARC-c) (Clark et al., 2018), ARC-Easy (ARC-e) (Clark et al., 2018), and Open-

BookQA (OBQA) (Mihaylov et al., 2018). We use the standard prompts from the LM Evaluation Harness (Gao et al., 2024) and report per-task accuracy and the unweighted mean across tasks. Evaluation settings are summarized in Appendix A.

4.2 Pre-training under the Chinchilla Scaling Law

To examine the scalability of AsymLoop, we jointly increase model size and training token budget following the Chinchilla scaling law. Figure 2(a, b) shows that AsymLoop consistently achieves lower validation loss than Dense and Full Loop across all four scales, from 60M to 1B. Its advantage over Full Loop is maintained at matched effective depth and training budgets within each scale. These results demonstrate the effectiveness of AsymLoop across model sizes, with sustained improvements at the billion-parameter scale. These gains also extend to an OLMo-style backbone (Groeneveld et al., 2024), where AsymLoop reduces validation perplexity by 14.8% relative to the depth-matched Full Loop (Appendix G).

AsymLoop consistently improves both language-modeling loss and zero-shot downstream accuracy over Full Loop across all evaluated scales. Notably, the downstream advantage grows monotonically with model scale, from 0.13 percentage points at 60M to 1.27 points at 1B. We examine the task-level results in Section 4.3.

Table 1: **Zero- and five-shot commonsense reasoning accuracy (↑, %).** Full Loop ($K = 2$) and AsymLoop ($K = 3$) have the same effective depth. Avg. is the mean across seven tasks. Darker/lighter blue mark the best/second-best results at each model size and shot setting.

Scale	Model	OBQA	WinoGrande	ARC-c	ARC-e	HellaSwag	SIQA	PIQA	Avg.
Zero-shot									
60M	Dense	**17.80**	**50.83**	17.41	45.33	27.39	34.54	59.03	36.05
	Full Loop	16.80	50.28	**20.22**	46.51	27.14	**37.00**	58.71	36.66
	AsymLoop	**17.80**	50.59	18.17	**47.31**	**27.72**	36.49	**59.41**	**36.79**
135M	Dense	18.80	48.54	21.42	52.90	29.06	37.41	61.53	38.52
	Full Loop	19.80	**51.85**	**22.87**	51.52	29.09	**38.69**	**62.19**	39.43
	AsymLoop	**22.00**	50.99	22.35	**54.50**	**30.22**	38.38	61.97	**40.06**
350M	Dense	22.20	**53.35**	26.62	59.39	33.82	39.15	67.30	43.12
	Full Loop	**24.80**	51.78	25.77	58.92	34.57	39.05	66.38	43.04
	AsymLoop	23.60	52.41	**27.05**	**62.12**	**35.59**	**39.20**	**67.79**	**43.97**
1B	Dense	28.40	**59.19**	33.96	69.02	41.47	41.61	71.49	49.31
	Full Loop	29.40	57.77	34.64	68.64	42.67	**42.53**	71.38	49.58
	AsymLoop	**32.40**	57.93	**36.69**	**70.71**	**44.34**	41.81	**72.03**	**50.85**
Five-shot									
60M	Dense	**17.60**	49.57	18.43	42.93	27.09	34.54	58.11	35.47
	Full Loop	15.40	50.51	**20.48**	44.44	27.27	35.21	58.32	35.95
	AsymLoop	16.60	**51.70**	18.86	**45.71**	**27.53**	**36.13**	**59.47**	**36.57**
135M	Dense	18.00	51.70	22.35	50.21	28.75	36.44	61.64	38.44
	Full Loop	17.20	**52.09**	22.44	50.72	28.85	**38.33**	**63.06**	38.96
	AsymLoop	**21.20**	50.28	**23.46**	**53.66**	**30.03**	37.56	62.95	**39.88**
350M	Dense	23.00	53.28	27.65	59.93	33.41	39.00	66.76	43.29
	Full Loop	21.20	52.17	26.88	60.02	34.33	39.15	66.92	42.95
	AsymLoop	**24.00**	**54.38**	**28.92**	**64.48**	**35.03**	**40.69**	**68.17**	**45.10**
1B	Dense	24.20	57.70	36.35	69.57	41.07	42.48	71.60	48.99
	Full Loop	26.60	58.25	36.77	70.54	42.84	43.09	**72.25**	50.05
	AsymLoop	**29.00**	**59.27**	**38.99**	**73.40**	**44.06**	**43.65**	71.76	**51.45**

4.3 Downstream Evaluation

We evaluate the pretrained models on seven commonsense reasoning benchmarks to assess their downstream generalization: PIQA (Bisk et al., 2020), Social IQa (Sap et al., 2019), HellaSwag (Zellers et al., 2019), WinoGrande (Sakaguchi et al., 2021), ARC-Challenge (Clark et al., 2018), ARC-Easy (Clark et al., 2018), and OpenBookQA (Mihaylov et al., 2018). Evaluations cover all four model sizes under both zero-shot and five-shot settings, using the standard prompts provided

by the LM Evaluation Harness (Gao et al., 2024). Table 1 summarizes the per-task and average accuracies.

As shown in Table 1, AsymLoop consistently outperforms Dense and Full Loop in average accuracy across all four model sizes under both zero-shot and five-shot settings. The improvements extend across multiple benchmarks, with particularly broad gains in five-shot evaluation: AsymLoop achieves the best performance on all seven tasks at 350M and six of the seven tasks at 1B. At 350M, its average accuracy exceeds Full Loop by 2.15 percentage points. These results demonstrate that the pre-training gains of AsymLoop translate into stronger downstream commonsense reasoning across both evaluation settings. We further analyze the training dynamics of downstream performance at the 1B scale in Appendix F.

Furthermore, we evaluate the mathematical reasoning performance of the 1B pretrained models through supervised fine-tuning on GSM8K (Cobbe et al., 2021). AsymLoop achieves an accuracy of 11.75%, a 1.52 percentage point improvement over Full Loop. These results suggest that its downstream benefits also extend to supervised fine-tuning; detailed results and training settings are provided in Appendix H.

4.3.1 Downstream Performance under a 5× Chinchilla Budget

To investigate whether the benefits of AsymLoop persist under extended training, we increase the training budget of the 350M models from 7B to 35B tokens, corresponding to five times the Chinchilla budget. We compare AsymLoop with Dense and Full Loop under the same token budget, maintaining matched effective depth between the recurrent models. Table 2 presents their zero-shot and five-shot performance on the seven commonsense reasoning benchmarks.

As shown in Table 2, AsymLoop maintains the highest average accuracy in both zero-shot and five-shot evaluation under the extended training budget. In five-shot evaluation, it matches or surpasses both baselines on all seven tasks, improving average accuracy over Full Loop from 46.67% to 47.41%. These results demonstrate that the downstream benefits of AsymLoop persist with a fivefold increase in training tokens, supporting its effectiveness under extended pre-training.

Table 2: **Zero- and five-shot downstream accuracy (↑, %) of 350M models trained on 35B tokens.** Avg. is the mean across seven tasks. Darker/lighter blue mark the best/second-best results within each shot setting.

Model	OBQA	WinoGrande	ARC-c	ARC-e	HellaSwag	SIQA	PIQA	Avg.
Zero-shot								
Dense	25.40	54.70	31.06	66.33	37.69	39.61	68.44	46.18
Full Loop ($K = 2$)	**27.20**	53.43	31.91	66.20	**39.04**	**41.40**	68.55	46.82
AsymLoop ($K = 3$, ALRS)	26.80	**55.80**	**32.68**	**68.35**	38.54	40.02	**69.04**	**47.32**
Five-shot								
Dense	21.20	54.54	30.72	67.00	37.21	41.20	68.77	45.81
Full Loop ($K = 2$)	23.00	56.51	31.66	66.50	38.53	**41.50**	**68.99**	46.67
AsymLoop ($K = 3$, ALRS)	**25.40**	**56.67**	**32.08**	**68.56**	**38.70**	**41.50**	**68.99**	**47.41**

4.4 Effect of Recurrent Depth

We investigate how recurrent depth affects AsymLoop by varying the number of decoder iterations from $K = 1$ to $K = 3$ and $K = 5$ at the 60M, 135M, and 350M scales. To compare architectures at matched effective depth, these configurations are paired with Dense, Full Loop with $K = 2$, and Full Loop with $K = 3$, respectively, yielding depths of L, $2L$, and $3L$. The training token budget is held fixed within each model scale. Table 3 reports validation perplexity for these comparisons.

As shown in Table 3, AsymLoop achieves lower validation perplexity than the corresponding depth-matched baseline in every comparison. Its advantage is already evident at $K = 1$ and persists as effective depth increases to $2L$ and $3L$. Increasing the number of decoder iterations further reduces perplexity across the 60M, 135M, and 350M scales. For example, at 350M, perplexity decreases

Table 3: **Effect of recurrent depth.** Validation perplexity (↓) of Dense, Full Loop, and AsymLoop across model sizes at matched effective depths. Darker/lighter blue mark lower/higher perplexity within each depth-matched pair.

Scale	Loop / Total Layers	K	Effective Depth	Architecture	Budget	PPL
60M	0 / 6	1	6	Dense	1.68B	23.72
	3 / 6	1	6	**AsymLoop (Ours)**	1.68B	**21.79**
	6 / 6	2	12	Full Loop	1.68B	23.03
	3 / 6	3	12	**AsymLoop (Ours)**	1.68B	**21.45**
	6 / 6	3	18	Full Loop	1.68B	22.77
	3 / 6	5	18	**AsymLoop (Ours)**	1.68B	**21.30**
135M	0 / 12	1	12	Dense	3.00B	17.46
	6 / 12	1	12	**AsymLoop (Ours)**	3.00B	**16.54**
	12 / 12	2	24	Full Loop	3.00B	17.12
	6 / 12	3	24	**AsymLoop (Ours)**	3.00B	**16.14**
	12 / 12	3	36	Full Loop	3.00B	16.81
	6 / 12	5	36	**AsymLoop (Ours)**	3.00B	**16.09**
350M	0 / 24	1	24	Dense	7.00B	12.55
	12 / 24	1	24	**AsymLoop (Ours)**	7.00B	**12.20**
	24 / 24	2	48	Full Loop	7.00B	12.30
	12 / 24	3	48	**AsymLoop (Ours)**	7.00B	**12.00**
	24 / 24	3	72	Full Loop	7.00B	12.15
	12 / 24	5	72	**AsymLoop (Ours)**	7.00B	**11.92**
1B	24 / 24	2	48	Full Loop	20.00B	8.91
	12 / 24	3	48	**AsymLoop (Ours)**	20.00B	**8.86**

from 12.20 at $K = 1$ to 11.92 at $K = 5$. These results demonstrate that AsymLoop remains effective across different depths and benefits from additional recurrent computation.

4.5 Ablation Studies and Analysis

We examine the contributions of AsymLoop's architectural and optimization components. Further analyses of asymmetric learning-rate allocation, Global KV under recurrence, and alternative context-access designs are provided in Appendices B and C.

4.5.1 Component Ablations

We evaluate the contributions of AsymLoop's attention design, normalization, and learning-rate allocation through component ablations (Table 4). For the architectural components, we individually remove Global KV, Local KV, and QKNorm under uniform learning rates, and include a parameter-matched variant without Global KV. For optimization, we compare ALRS with uniform learning rates and a fixed decoder learning-rate boost, and examine sensitivity to the base learning rate.

Architectural components. Under uniform learning rates, removing either Global KV or Local KV increases validation perplexity at both $K = 3$ and $K = 5$ (Table 4a), indicating that encoder context and evolving decoder states provide complementary information. Retaining Global KV also yields lower perplexity than reallocating its parameters to a wider FFN, suggesting that access to encoder context is more effective than additional FFN capacity in these comparisons. Removing QKNorm increases perplexity at both recurrent depths.

ALRS and base learning rate. At the default base learning rate, ALRS lowers perplexity by approximately one point relative to uniform learning rates at both recurrent depths (Table 4b). It also achieves lower perplexity than a fixed decoder learning-rate boost, indicating the benefit of adapting

Table 4: **Ablations at the 60M scale.** Validation perplexity (↓). **(a)** Architectural components and **(b)** learning-rate ablations on AsymLoop ($K = 3, 5$). **(c)** Uniform LR versus ALRS on Full Loop ($K = 2$). Darker/lighter blue mark the best/second-best results within each panel of (a, b).

Model Variant	**PPL ↓ ($K = 3$)**	**PPL ↓ ($K = 5$)**
(a) Architectural components		
AsymLoop (ALRS)	**23.751**	**23.580**
AsymLoop (uniform LR)	<u>24.727</u>	<u>24.619</u>
w/o QKNorm	24.984	24.663
w/o Global KV	25.566	25.751
w/o Local KV	25.656	25.465
Baseline with parameter matching	25.339	25.323
(b) ALRS and base learning rate		
AsymLoop	**23.751**	**23.580**
w/o ALRS	24.727	24.619
w/ $0.5\times$ LR	<u>24.059</u>	<u>23.826</u>
w/ $2\times$ LR	24.865	24.278
w/ $4\times$ LR	26.181	25.941
w/ static decoder boost	24.785	24.708
(c) Full Loop ($K = 2$)	**Uniform LR**	**ALRS**
Validation PPL ↓	25.571	24.787

learning rates at the matrix level. Among the tested base learning rates, 0.001 gives the best results at both $K = 3$ and $K = 5$.

AsymLoop also retains its advantage over a widened Full Loop baseline at matched analytical per-step training FLOPs; detailed comparisons are provided in Appendix E.

5 Conclusion

We presented AsymLoop, an asymmetric looped Transformer that combines single-pass context encoding with recurrent state refinement through joint attention to Global and Local KV. We train this architecture using the spectral layerwise learning-rate method of He et al. (2026) and evaluate its effect through optimization ablations. Across four model scales from 60M to 1B, AsymLoop achieves lower validation perplexity and higher average downstream accuracy than the depth-matched Full Loop baseline. Its language-modeling gains extend to an OLMo-style backbone and sparse MoE feed-forward layers, while supervised fine-tuning on GSM8K also yields higher mathematical reasoning accuracy. Although AsymLoop requires more training FLOPs than the original Full Loop, it achieves lower validation loss than a widened Full Loop baseline under matched training FLOPs.

AI USE STATEMENT

Generative AI tools assisted with polishing the language of this manuscript. The authors reviewed the AI-assisted text and take responsibility for all content and claims in this paper.

REPRODUCIBILITY STATEMENT

The model architecture is described in Section 3, and Appendix A provides the training configurations, hyperparameters, and computational resources. All source datasets used in our experiments are publicly available; Section 4.1 and the relevant appendices specify our FineWeb-Edu shard selection and other custom training and validation splits. For the seven downstream benchmarks, we use the standard prompts and evaluation protocols of the LM Evaluation Harness (Gao et al., 2024); the separate GSM8K protocol is documented in Appendix H. The H100 pre-training runs used up to 32 NVIDIA H100 GPUs per run, while hardware for other experiments is identified in their respective protocols. We plan to publicly release the training and evaluation code and the model checkpoints used in this work upon acceptance.

REFERENCES

Yonatan Bisk, Rowan Zellers, Jianfeng Gao, Yejin Choi, et al. Piqa: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pp. 7432–7439, 2020.

Hugh Blayney, Álvaro Arroyo, Johan Obando-Ceron, Pablo Samuel Castro, Aaron Courville, Michael M. Bronstein, and Xiaowen Dong. A mechanistic analysis of looped reasoning language models, 2026. URL https://arxiv.org/abs/2604.11791.

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.

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

DeepSeek-AI, :, Anyi Xu, B. Li, Bangcai Lin, Bing Xue, BingCheng Xian, Bingzheng Xu, Bochao Wu, Bowei Zhang, Boyi Deng, C. C. Yu, Chao Jin, Chaofan Lin, Chen Dong, Chenbing Wang, Chenfan Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chengyuan Zhang, Chenhao Xu, Chenqi Zhao, Chenze Shao, Chuhao Wang, Chuqi Zhang, Damai Dai, Dejian Yang, Deli Chen, Di Huang, Di Wu, Donghao Li, Erhang Li, Eric Fu, F. Zhou, Fangwei Zhou, Fangyun Lin, Fangzhou Yuan, Feiyu Xia, Fucong Dai, Guangbo Hao, Guanglin Li, Guanting Chen, Guoai Cao, Guofan Fan, Guolai Meng, Guowei Li, Haichuan Zhang, Haiyang Ma, Haiyang Shen, Han Li, Han Yu, Han Zhang, Hangyuan Deng, Hanwei Xu, Hanxiang Xu, Hanxun Zhong, Hao Guo, Hao Jiang, Hao Li, Hao Qin, Haodong Wen, Haofen Liang, Haofeng Huang, Haohua Liu, Haoling Zhang, Haoming Luo, Haoran Yang, Haotian Xu, Haotian Yuan, Haoting Huang, Haowen Luo, Haoyang Cai, Haoyu Chen, Haozhe Ji, Hengran Zhang, Hengrui Wang, Hengxu Wu, Honghui Ding, Hongxuan Tang, Huadong Wang, Huanqi Cao, Huazuo Gao, Hui Qu, Hui Zeng, J. Yang, J. H. Jin, J. H. Zhang, J. X. Zou, Jia Yu, Jiahui Zhou, Jiajun Chen, Jialiang Huang, Jialin Zhao, Jiamin Tang, Jian Zhou, Jianan Tong, Jianwen Li, Jiaqi Zhu, Jiarui Wang, Jiasheng Ye, Jiashi Li, Jiaxin Xu, Jiaying Ding, Jibai Lu, Jiewen Hu, Jin Yan, Jincheng Zhai, Jingchang Chen, Jingcheng Hu, Jingli Zhou, Jingsheng Xu, Jingting Xiang, Jingyan Yun, Jingyang Yuan, Jingyuan Cheng, Jinhua Zhu, Jinpeng Wang, Jinyi Chen, Jinyi Hu, Jiping Yu, Jueliang Guo, Junbo Pei, Junbo Sun, Junguang Jiang, Junjie Qiu, Junkang Zhou, Junqi Liu, Junren Li, Junxian Li, Junxiao Song, Junyi Guo, Kai Dong, Kaifeng Chen, Kaige Gao, Kang Guan, Kangdong Yuan, Ke Hong, Ke Xu, Kefan Zhao, Kexin Ji, Kexin Zhang, Kexing Zhou, Kuai Yu, Lan Zhang, Lean Wang, Lecong Zhang, Lei Wang, Letian Gao, Liang Zhao, Liansheng Xu, Lihua Guo, Lingxiao Luo, Lingyue Fu, Litao Deng, Litong Wang, Liyue Zhang, Longhao Chen, Lu Chen, Luotian Huang, Luyao Ma, Luyao Wang, M. S. Di, Max Mei, Menghao Ye, Miao Cui, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingjing Zhang, Mingqi Wei, Mingshu Chen, Mingxing Liu, Mingxu

Zhou, Mingyu Xu, Mingyu Yang, Mingze Wang, Muyang Chen, Ni Shentu, Ning Wang, Niufang Ning, Panpan Huang, Peixin Cong, Peiyi Wang, Peiyuan Xin, Pengfei Ren, Pengfei Yan, Pengle Zhang, Qi Kang, Qi Tang, Qiancheng Wang, Qiang Li, Qihao Zhu, Qingyang Li, Qinyu Chen, Qiushi Du, Qizhou Guo, Rongxian Xu, Rui Ding, Rui Hu, Rui Tian, Rui Yu, Ruidong Zhu, Ruifan Xu, Ruihan Yang, Ruihang Xia, Ruijie Lu, Ruilin Geng, Ruipeng Hong, Ruiqi Ge, Ruisong Zhang, Ruize Sun, Ruizhe Pan, Runji Wang, Runqian Chen, Runxin Xu, Ruohong Tian, Ruomeng Shen, Ruoyu Zhang, Ryan X., S. H. Liu, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shaofei Cai, Shaoheng Nie, Shaoyuan Chen, Shengding Hu, Shengkai Lin, Shengwen Ran, Shengyu Liu, Shengyuan Jia, Shi Bai, Shi Feng, Shicheng Xu, Shichun Liu, Shiqiang Hu, Shirong Ma, Shiyu Wang, Shiyuan Feng, Shufan Gong, Shuhan Lin, Shuiping Yu, Shunfeng Zhou, Shuo Yang, Shuomeng Wang, Shuting Guo, Shuting Pan, Shuying Yu, Sinuo Cao, Siyi Lin, Sizhe Chen, Songyang Chen, Songyang Zhou, Tao Ni, Tao Yun, Tian Jin, Tian Pei, Tian Ye, Tianle Lin, Tianran Ji, Tianyi Cui, Tianyuan Yue, Tingting Yu, Tongrui Xiong, Wangding Zeng, Wei Liu, Wei Zhang, Weibin Xu, Weihao Zeng, Weilin Zhao, Wen Liu, Wenfeng Liang, Wenjie Pang, Wenjing Luo, Wenjing Yao, Wenjun Gao, Wenkai Shao, Wenkai Yang, Wenli Zhang, Wenlu Wang, Wenlve Huang, Wenqian Yan, Wentao Zhang, Xi Gao, Xiang He, Xiang Li, Xiangli Li, Xiangwen Wang, Xiangying Zhang, Xiankui Wei, Xiao Bi, Xiaodong Liu, Xiaohan Wang, Xiaojian Qu, Xiaokang Chen, Xiaokang Zhang, Xiaotao Nie, Xiaoyao Zou, Xiaoyuan Li, Xicheng Guo, Xieting Chu, Xin Cheng, Xin Liu, Xin Xie, Xinbo Xu, Xingchao Liu, Xingchen Liu, Xingkai Yu, Xingyou Li, Xintong Yao, Xinyang Chen, Xinyong Jiang, Xinyu Yang, Xinyu Yang, Xu Chen, Xuanyu Wang, Xubei Zhong, Xuecheng Su, Xuejie Liu, Xuheng Lin, Xujie Fan, Xuncheng Zhao, Xuwei Fu, Y. C. Yan, Y. H. Jiang, Y. T. Wu, Y. W. M., Y. Z. Wang, Yafei Gao, Yang Yang, Yang Zhang, Yanru Ma, Yanwen Huang, Yao Li, Yao Li, Yao Meng, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yaoyang Ye, Yehang Yin, Yexinrui Wu, Yi Qian, Yi Tao, Yi Yu, Yichao Zhang, Yichen Jiang, Yicheng Wang, Yifan Ding, Yifan Shi, Yifeng Peng, Yifeng Zhai, Yijia Wu, Yiliang Xiong, Yilun Wang, Ying He, Ying Zhou, Yingjia Luo, Yinmin Zhong, Yiping Wang, Yisong Wang, Yixiang Zhang, Yixiao Chen, Yixuan Tan, Yixuan Wei, Yiyang Ma, Yiyao Yang, Yiyuan Liu, Yizai Cai, Yizhen Wei, Yizhi Wang, Yonglun Yang, Yongqi Zhuo, Yongqiang Guo, Yongtong Wu, Yu Wu, Yu Zhang, Yuan Bian, Yuan Cheng, Yuan Ou, Yuan Sun, Yuanfan Xu, Yuanhang Sun, Yuanhao Li, Yuchen Liu, Yuchen Yao, Yudong Han, Yuduan Wang, Yuhan Wu, Yuhao Meng, Yuheng Zou, YuKun Li, Yunchuan Wang, Yunfan Xiao, Yunfan Xiong, Yupeng Chen, Yuqian Cao, Yuqian Wang, Yuqing Chen, Yushun Zhang, Yutong Lin, Yuwei Xiao, Yuxian Gu, Yuxiang Chen, Yuxiang Huang, Yuxiang Luo, Yuxiang You, Yuxin Chen, Yuxin Xiang, Yuxuan Liu, Yuxuan Zhou, Yuyang Zhou, Yuzhe Guo, Yuzhen Huang, Yuzhuo Bai, Z. Y. Z., Zanlin Ni, Zehao Wang, Zehua Zhao, Zehui Ren, Zejun Zhao, Zhangli Sha, Zhanying Wang, Zhaochen Zhang, Zhaoshuai Du, Zhe Fu, Zhean Xu, Zhenda Xie, Zheng Liu, Zhengyan Zhang, Zhenhua Dong, Zhewen Hao, Zhibang Wang, Zhibin Gou, Zhicheng Ma, Zhihao Li, Zhihong Shao, Zhihuan Huang, Zhijie Li, Zhirui Lu, Zhixian Huang, Zhixuan Chen, Zhixuan Chen, Zhixuan Pan, Zhiyu Wu, Zhizhou Ren, Zhu He, Zhuoshu Li, Zhuping Zhang, Zian Xu, Zihao Wang, Zihui Gu, Zijia Zhu, Zili Zhang, Zilin Li, Zilong Hou, Zilong Lyu, Ziqiao Wang, Ziwei Xie, Ziya Zhang, Ziyi Gao, Zizheng Pan, Zonglin Li, Zongqing Yao, Zui Chen, Zuofan Wu, Chenchen Ling, Chengyu Hou, Chong Chen, D. Li, Di Qi, Dongjie Ji, Fang Wei, Fanyi Xia, Fei Xie, Feiyi Tan, Hailong Guo, Haiyan Zhai, Hui Zhou, Huihui Tan, Huijie Li, Jia Luo, Jia Song, Jialu Cai, Jian Liang, Jiangting Zhou, Jiaqi Gao, Jiayi Shao, Jie Chen, Jieyu Yang, Jin Chen, Jingde Zhang, Jingzi Zhou, Jinqian Wang, Jinyang Liu, JinZhao Sun, Junhua Ling, Junmin Zheng, Kaicheng Yang, Ke Xu, Le Su, Leyi Xia, Liangfeng Ding, Lin Zhuo, Linwang Ma, Linyan Zhu, Liyu Cai, Luqi Yao, M. K. Zhang, Meng Li, Miao Lin, Miaojun Wang, Min Zhang, Mingming Li, Mingming Wang, Mingze Yin, Minmin Han, Nan Cao, Ning Wang, Ningxin Ma, Panpan Wang, Peihan Lin, Peng Sun, Peng Zhang, Qian Ying, Qiang Xiang, Qiao Wang, Qingmiao Mao, Qiwei Jiang, Rongli Jin, Ruyi Chen, Sha Tao, Shangmian Sun, Shaoqing Wu, Shichao Zou, Si Lei, Tianyang Zhang, Tianyu Sun, Tingting Yin, W. L. Xiao, Wei An, Wei Li, Wei Wang, Weiwei Lin, Wenqing Hou, X. Lin, Xiangfei Meng, Xianzhu Huang, Xiao Peng, Xiaoqian Li, Xiaoting Zhang, Xiaowen Sun, Xiaoxiang Wang, Xiaoyu Ye, Xinrou Zhang, Xinyu Zhang, Xue Cao, Xueyin Chen, Yanan Zhou, Yanhong Xu, Yao Xia, Yao Xu, Yi Shao, Yihong Zhang, Yiling Ma, Ying Tang, Yining Lou, Yiru Chen, Yishi Piao, Yixuan Chen, Yong Xiong, Yuchen Xuan, Yuehan Yang, Yuer Xu, Yukun Zha, Yunxian Ma, Yuping Lin, Yuting Yan, Yutong Xie, Yuwen Sheng, Yuxuan Zhu, Zekai Zhang, Zhe Ju, Zhenzhen Lin, Zheren Gao, Zheyang Sun, Zhigang Yan, Zhongyu Wu, Zi Wang, Zihua Qu, Ziling Yan, and Ziyi Wan. Deepseek-v4.1-flash: Pushing the limits of kv cache compression, 2026. URL https://arxiv.org/abs/2609.19969.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Lukasz Kaiser. Universal transformers. In *International Conference on Learning Representations*, 2019.

Rao Fu, Zixuan Yang, Jiankun Zhang, Jing Ma, Hechang Chen, Yu Li, and Yi Chang. Simply stabilizing the loop via fully looped transformer. *arXiv preprint arXiv:2605.18797*, 2026.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. The language model evaluation harness, 07 2024. URL https://zenodo.org/records/12608602.

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. *arXiv preprint arXiv:2502.05171*, 2025.

Angeliki Giannou, Shashank Rajput, Jy-yong Sohn, Kangwook Lee, Jason D. Lee, and Dimitris Papailiopoulos. Looped transformers as programmable computers. In *International Conference on Machine Learning*, 2023.

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

Di He, Songjun Tu, Keyu Wang, Lu Yin, and Shiwei Liu. One lr doesn't fit all: Heavy-tail guided layerwise learning rates for llms. *arXiv preprint arXiv:2605.22297*, 2026.

Tom Henighan, Jared Kaplan, Mor Katz, Mark Chen, Christopher Hesse, Jacob Jackson, Heewoo Jun, Tom B. Brown, Prafulla Dhariwal, Scott Gray, et al. Scaling laws for autoregressive generative modeling. *arXiv preprint arXiv:2010.14701*, 2020.

Alex Henry, Prudhvi Raj Dachapally, Shubham Shantaram Pawar, and Yuxuan Chen. Query-key normalization for transformers. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pp. 4246–4253, 2020.

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

Benhao Huang, Chufan Shi, Junlin Chen, Shicheng Wen, Zhengzhong Liu, Eric Xing, and Xuezhe Ma. Towards looped models done right—part I: Topology, input injection, recurrent-state design, 2026. URL https://ifm-research.notion.site/Towards-Loop-Models-Done-Right-3ade511912ec8128987dfeb7a5580043. Research blog. Accessed September 20, 2026.

Vedang Lad, Jin Hwa Lee, Wes Gurnee, and Max Tegmark. The remarkable robustness of llms: Stages of inference?, 2025. 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.

Charles H. Martin and Michael W. Mahoney. Traditional and heavy-tailed self regularization in neural network models, 2019. URL https://arxiv.org/abs/1901.08276.

Charles H. Martin and Michael W. Mahoney. Traditional and heavy-tailed self-regularization in deep neural networks. *Journal of Machine Learning Research*, 22(1):1–71, 2021.

Charles H. Martin, Tong Sheng Peng, and Michael W. Mahoney. Predicting trends in the quality of state-of-the-art neural networks without access to training or testing data. *Nature Communications*, 12(1):4122, 2021.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. Can a suit of armor conduct electricity? a new dataset for open book question answering. *arXiv preprint arXiv:1809.02789*, 2018.

Niklas Muennighoff, Luca Soldaini, Dirk Groeneveld, Kyle Lo, Jacob Morrison, Sewon Min, Weijia Shi, Pete Walsh, Oyvind Tafjord, Nathan Lambert, et al. OLMoE: Open mixture-of-experts language models. In *International Conference on Learning Representations*, volume 2025, pp. 62061–62121, 2025.

Guilherme Penedo, Hynek Kydlíček, Anton Lozhkov, Margaret Mitchell, Colin A. Raffel, Leandro von Werra, Thomas Wolf, et al. The fineweb datasets: Decanting the web for the finest text data at scale. *Advances in Neural Information Processing Systems*, 37, 2024.

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.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: an adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106, 2021.

Maarten Sap, Hannah Rashkin, Derek Chen, Ronan LeBras, and Yejin Choi. Socialiqa: Commonsense reasoning about social interactions. *arXiv preprint arXiv:1904.09728*, 2019.

Nikunj Saunshi, Nishanth Dikkala, Zhiyuan Li, Sanjiv Kumar, and Sashank J. Reddi. Reasoning with latent thoughts: On the power of looped transformers. *arXiv preprint arXiv:2502.17416*, 2025.

Yutao Sun, Li Dong, Yi Zhu, Shaohan Huang, Wenhui Wang, Shuming Ma, Quanlu Zhang, Jianyong Wang, and Furu Wei. You only cache once: Decoder-decoder architectures for language models. *Advances in Neural Information Processing Systems*, 37:7339–7361, 2024.

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.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830*, 2019.

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, 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 Eshraghian. Scaling latent reasoning via looped language models. *arXiv preprint arXiv:2510.25741*, 2025.

A Architecture, Training, and Evaluation Details

Table 5 summarizes the architectures and pre-training hyperparameters for the four model configurations. We denote these configurations by 60M, 135M, 350M, and 1B, with the actual parameter counts reported separately for each architecture.

All main pre-training experiments use cleaned and deduplicated FineWeb-Edu shards 00000–00039 for training, with shard 00233 reserved for validation. We use the t5-base tokenizer with a vocabulary size of 32,128 and a sequence length of 1024. Input embeddings and the output head are untied.

Training budgets are matched across architectures within each model configuration. The 60M models are trained for 3,200 optimizer steps, corresponding to approximately 1.68B input token positions including padding. The 135M, 350M, and 1B models are trained on 3B, 7B, and 20B tokens, respectively.

Table 5: **Model architectures and pre-training hyperparameters.** Backbone configurations follow He et al. (2026). Parameter counts include the untied input embeddings and output head.

Configuration	**60M**	**135M**	**350M**	**1B**
Dense parameters	91.8M	134.3M	368.2M	1.34B
AsymLoop parameters	95.4M	141.4M	393.4M	1.44B
Layers L $(E + D)$	6 (3+3)	12 (6+6)	24 (12+12)	24 (12+12)
d_{model}	768	768	1024	2048
d_{ff}	2048	2048	2736	5461
Heads	6	12	16	32
Training budget	1.68B	3.00B	7.00B	20.00B
Peak LR η_0	1.0×10^{-3}	1.0×10^{-3}	1.0×10^{-3}	5.0×10^{-4}
Weight decay	0.1	0.1	0.1	0.1
Effective depth $(L/2L/3L)$	6 / 12 / 18	12 / 24 / 36	24 / 48 / 72	24 / 48 / 72

We use AdamW with $(\beta_1, \beta_2) = (0.9, 0.95)$, $\epsilon = 10^{-8}$, and weight decay 0.1. All main pre-training experiments use a global batch size of 512, a global gradient clipping threshold of 1.0 in L2 norm, and random seed 6198. The base learning rate follows linear warmup over the first 10% of training, followed by cosine decay to 10% of its peak. Schedule progress is measured in optimizer steps at 60M and training tokens at larger scales. ALRS scheduling details are provided in Appendix B.1.

Large-scale pre-training uses up to 32 NVIDIA H100 GPUs, each with 80 GB of memory.

We report validation cross-entropy (CE) and perplexity (PPL) to assess pre-training performance. For downstream commonsense reasoning, we report zero-shot and five-shot accuracy using the standard prompts provided by the LM Evaluation Harness (Gao et al., 2024).

B ALRS Allocation and Weight-Spectrum Diagnostics

B.1 ALRS Scheduling Details

Following the allocation and scheduling procedure of He et al. (2026), ALRS assigns matrix-wise learning-rate multipliers in $[1, 5]$. Allocations are updated at the first optimizer step and every 100 steps thereafter during the first 20% of training, with a 50-step linear transition to each new learning-rate target. After this initial phase, the multipliers remain fixed while the base learning rate continues its cosine decay.

B.2 Observed Encoder–Decoder Learning-Rate Differences

We analyze how ALRS allocates learning rates between the encoder and decoder in the 135M and 350M models. Figure 3 shows their mean learning-rate trajectories and decoder-to-encoder ratios separately for Attention and FFN modules.

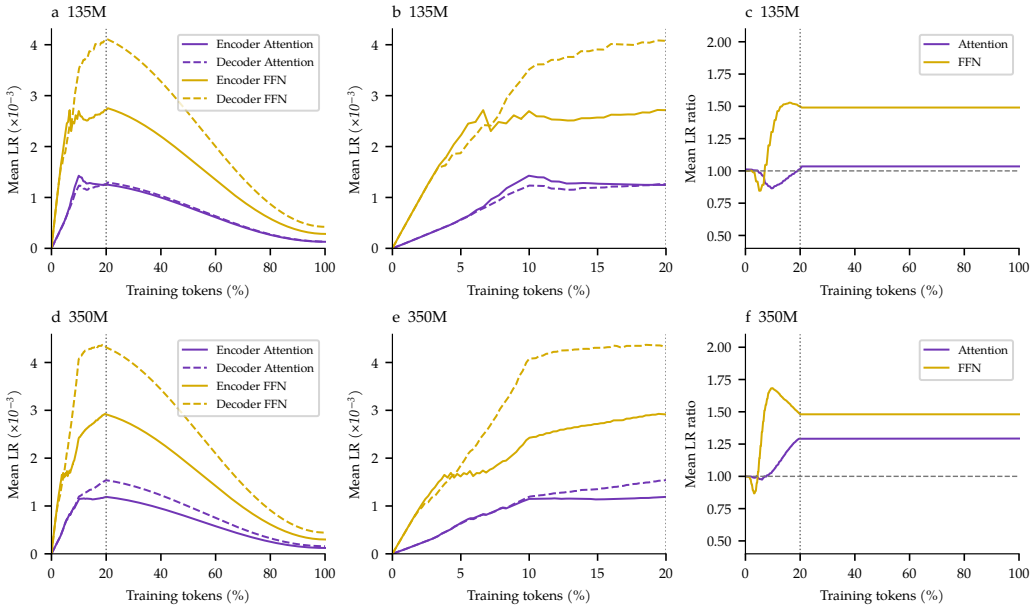

Figure 3: **Learning-rate dynamics under ALRS at 135M and 350M. Left:** Full training trajectories. **Middle:** The first 20% of training. **Right:** Decoder-to-encoder mean learning-rate ratios. Vertical dotted lines mark the end of allocation updates.

Figure 3 shows a consistent FFN asymmetry across model scales: after the allocation updates end, the decoder's mean FFN learning rate is approximately 1.5 times the encoder's. For Attention, the encoder–decoder difference is more pronounced at 350M than at 135M. These differences arise from applying the same spectral allocation rule to all matrices; the approximately 1.5-fold FFN ratio is an observed outcome rather than a prescribed decoder multiplier.

B.3 Weight-Spectrum Analysis

To examine the weight structure learned under ALRS, we compare normalized stable rank between uniform learning rates and ALRS in the 135M and 350M AsymLoop models ($K = 3$). Statistics are averaged equally over the local Q/K/V/O matrices for Attention and the gate/up/down matrices for FFN, separately for the encoder and decoder.

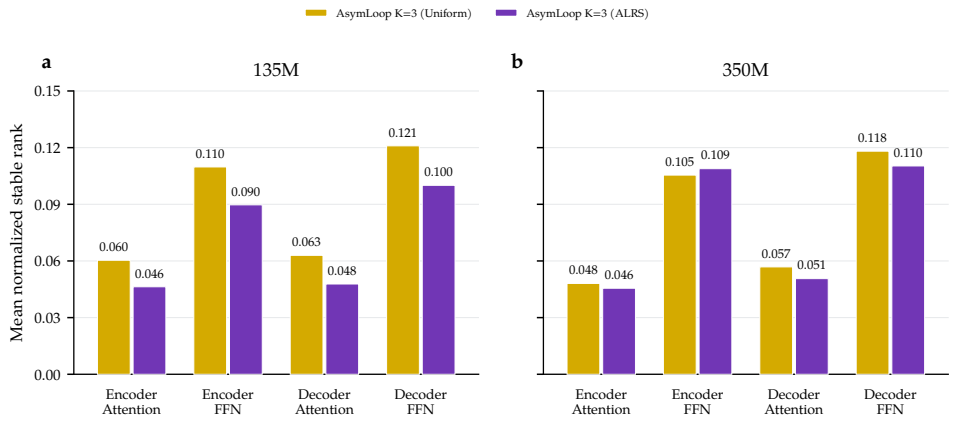

Figure 4: **Normalized stable rank under uniform learning rates and ALRS.** Results for 135M and 350M AsymLoop models ($K = 3$). For each matrix W, we compute $\|W\|_F^2/[\|W\|_2^2 \min(\text{shape}(W))]$ and average equally within each module group.

Figure 4 shows that ALRS generally produces lower normalized stable rank, with consistent reductions in Attention and decoder FFN modules at both model scales. This pattern reflects greater relative concentration of spectral energy in the leading singular direction, indicating that differentiated learning-rate allocation is accompanied by changes in the learned weight structure.

C Complete Ablation Results and Evaluation Details

C.1 Global KV under Recurrence

We compare AsymLoop with and without Global KV at $K = 1, 3, 5$ to examine how its contribution changes with recurrent depth. Both variants retain QKNorm and use uniform learning rates.

Table 6: **Global KV and recurrent depth.** Validation CE (↓) with and without Global KV. $G(K)$ is the CE reduction from Global KV; the final row reports the additional gain $G(K) - G(1)$.

Model Variant	**CE ↓ ($K = 1$)**	**CE ↓ ($K = 3$)**	**CE ↓ ($K = 5$)**
AsymLoop	**3.238735**	**3.207915**	**3.203515**
w/o Global KV	3.260346	3.241257	3.248483
Global KV gain $G(K)$	0.021611	0.033343	0.044968
Gain beyond $K = 1$	0.000000	0.011732	0.023357

Global KV reduces validation cross-entropy at every tested depth, with its gain increasing from 0.022 at $K = 1$ to 0.045 at $K = 5$ (Table 6). This trend supports a greater benefit from encoder context as recurrence deepens.

C.2 Alternative Context-Access Designs

We compare Global KV with two alternatives: removing this access path or replacing it with additive injection of the encoder output before each decoder iteration. The additive variant computes $z_0 = 0$ and $z_t = \mathcal{D}(z_{t-1} + E)$, where E is the encoder output. Full Loop provides an additional baseline. All models use QKNorm and uniform learning rates.

Table 7: **Context-access variants and a Full Loop baseline.** Validation perplexity (↓). Full Loop uses $K = 2$; all other variants use $K = 3$. Darker/lighter blue mark the best/second-best results.

Model Variant	PPL ↓
AsymLoop	**24.727**
w/o Global KV	25.566
w/ additive injection	25.116
Full Loop	25.053

As shown in Table 7, AsymLoop achieves the lowest validation perplexity. Additive injection improves over removing Global KV, while retaining Global KV provides a further gain. These results highlight the benefit of accessing encoder context through attention during recurrent computation.

C.3 Additional Validation Results

Table 8 reports validation cross-entropy and perplexity for Dense, Full Loop, and AsymLoop without QKNorm under a common evaluation protocol. The AsymLoop results also provide the reference values for the QKNorm ablation in Table 4.

Table 8: **Additional validation results.** Cross-entropy and perplexity (↓) under a common evaluation protocol. Darker/lighter blue mark the best/second-best results in each metric column.

Reference	K	Effective depth	CE	PPL
Dense, Uniform	1	6	3.271820	26.359274
Dense, layerwise LR	1	6	3.209192	24.759083
Full Loop, Uniform	2	12	3.241455	25.570904
Full Loop, Uniform	3	18	3.229346	25.263120
AsymLoop w/o QKNorm, Uniform	3	12	3.218239	24.984094
AsymLoop w/o QKNorm, Uniform	5	18	**3.205305**	**24.663015**

D Computational Cost and Downstream Performance

We analyze AsymLoop's computational cost and its relationship with downstream performance. Compute-matched comparisons with a widened Full Loop baseline are reported in Appendix E.

D.1 Cost of Global KV and Joint Attention

Let N, d, m, and V denote the sequence length, hidden dimension, SwiGLU intermediate dimension, and vocabulary size, respectively. We count two FLOPs per multiply–accumulate and use full attention matrices. A standard Transformer block has forward cost

$$C(m) = 8Nd^2 + 6Ndm + 4N^2d. \tag{6}$$

At matched effective depth $J = E + KD = RL$, the Full Loop cost is $F_{\text{Full}} = F_{\text{blk}} + F_{\text{head}}$, where $F_{\text{blk}} = JC(m)$ and $F_{\text{head}} = 2NdV$. Here, $E = D = L/2$ and $R = (K + 1)/2$.

AsymLoop adds Global KV projections computed once per decoder layer and additional attention products at each decoder iteration:

$$\begin{aligned} F_{\text{proj}} &= 4DNd^2, \qquad F_{\text{glob}} = 4KDN^2d, \\ F_{\text{Asym}} &= F_{\text{Full}} + F_{\text{proj}} + F_{\text{glob}}. \end{aligned} \tag{7}$$

Table 9: **Analytical forward FLOPs at matched effective depth.** Full Loop uses $R = 2$ and AsymLoop uses $K = 3$, with $N = 1024$ and $V = 32128$. Costs are reported per input position; overhead is relative to Full Loop.

	Forward GFLOPs/token		Additional cost (%)		
Model size	Full Loop	AsymLoop	Global KV proj.	Global attn.	Total
60M	0.256967	0.292356	2.75	11.02	13.77
135M	0.464585	0.535364	3.05	12.19	15.23
350M	1.476657	1.677984	3.41	10.23	13.63
1B	5.365891	5.869208	3.75	5.63	9.38

Table 9 summarizes the resulting forward costs. Across the evaluated model sizes, these components add 9.38%–15.23% to the Full Loop baseline.

D.2 Compute–Performance Trade-offs Across Recurrent Depths

We investigate the trade-off between training compute and downstream performance across model scales and recurrent depths. Figure 5 compares average zero-shot accuracy against analytical training FLOPs, pairing AsymLoop with Dense or Full Loop at matched effective depths of L, $2L$, and $3L$.

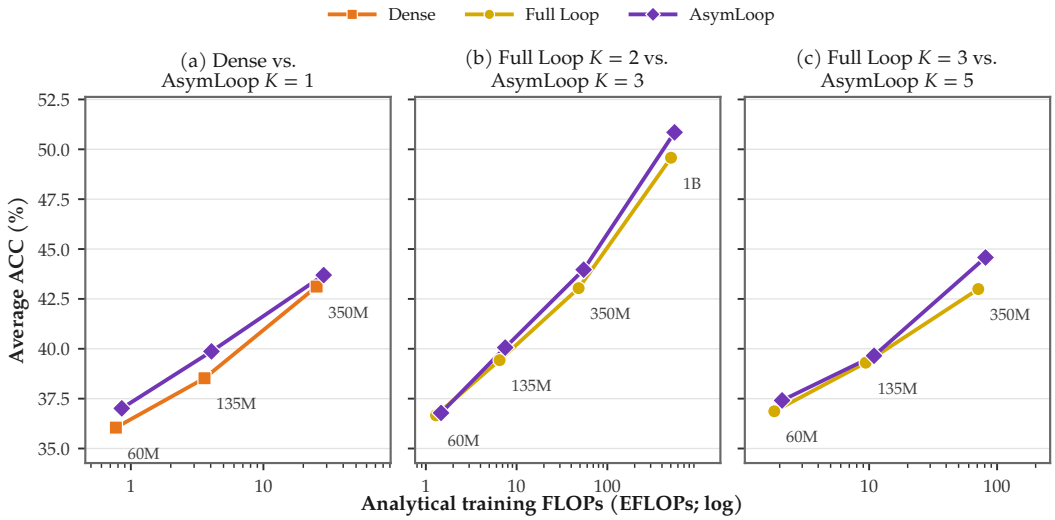

Figure 5: **Training compute–performance trade-offs across model scales and recurrent depths.** Average zero-shot accuracy over seven downstream tasks versus analytical training FLOPs. The horizontal axis uses a logarithmic scale in EFLOPs (10^{18} FLOPs). Panels pair AsymLoop with the baseline of matched effective depth: (a) Dense and AsymLoop $K = 1$ (L), (b) Full Loop $K = 2$ and AsymLoop $K = 3$ ($2L$), and (c) Full Loop $K = 3$ and AsymLoop $K = 5$ ($3L$). Purple, yellow, and orange denote AsymLoop, Full Loop, and Dense, respectively.

Across all three depth settings, AsymLoop consistently achieves higher accuracy than its depth-matched baseline with a modest increase in training compute. Comparing across depths further reveals a favorable compute–performance trade-off: at 60M, 135M, and 350M, AsymLoop with $K = 1$ achieves higher accuracy than Full Loop with either $K = 2$ or $K = 3$, while requiring fewer training FLOPs. These results show that AsymLoop can offer both better downstream performance and lower compute than deeper Full Loop configurations. Complementary comparisons with a widened Full Loop baseline at matched analytical per-step training FLOPs are provided in Appendix E.

E PERFORMANCE AT MATCHED PER-STEP TRAINING FLOPS

To investigate whether AsymLoop's performance gains arise solely from additional training FLOPs, we compare it with a widened Full Loop baseline. We increase the baseline's SwiGLU intermediate dimension while preserving its recurrent architecture, matching AsymLoop's analytical per-step training FLOPs under a full-block recomputation assumption.

E.1 EXPERIMENTAL SETUP

We evaluate the original Full Loop, widened Full Loop, and AsymLoop at the 60M and 135M scales. At both scales, the widened baseline increases the FFN dimension from 2048 to 2656. Full Loop uses two iterations, while AsymLoop uses three decoder iterations with ALRS. Within each scale, all models share the training data, token budget, batch size, and distributed training configuration. At 60M, we report validation cross-entropy from one run per configuration; at 135M, we report the mean and sample standard deviation over three training seeds per configuration.

E.2 RESULTS

Table 10 shows that widening Full Loop improves validation cross-entropy, closing approximately 40% of its gap to AsymLoop at both model scales. Nevertheless, AsymLoop achieves the lowest cross-entropy in both comparisons. Its advantage also persists across all three random seeds in the 135M experiments. These results suggest that additional training FLOPs alone do not explain AsymLoop's performance gains.

At 135M, each configuration is evaluated over three training seeds. The widened Full Loop and AsymLoop achieve validation cross-entropies of 2.78983 ± 0.00197 and 2.74947 ± 0.00299, respectively (mean $\pm$ sample standard deviation). The mean difference is 0.04036, and all three AsymLoop runs achieve lower cross-entropy than all three widened Full Loop runs. These observations support the consistency of the improvement within the evaluated training configuration.

Table 10: **Validation cross-entropy and training cost with a widened Full Loop baseline.** The widened baseline matches AsymLoop's analytical per-step training FLOPs. The 60M results are single runs; the 135M results are means $\pm$ sample standard deviations over three training seeds. Step time and peak memory are measured on eight H100 GPUs. Darker/lighter blue marks the best/second-best cross-entropy at each model scale.

Model size	Model	FFN	Params	s/step	Peak mem.	Validation CE
60M	Full Loop $K=2$	2048	91.83M	0.232	20.2 GiB	3.0434
	+ FLOPs matching	2656	100.23M	0.230	21.6 GiB	3.0139
	AsymLoop $K=3$	2048	95.37M	0.216	21.7 GiB	**2.9684**
135M	Full Loop $K=2$	2048	134.30M	0.286	30.9 GiB	2.81550 ± 0.00221
	+ FLOPs matching	2656	151.11M	0.296	33.6 GiB	2.78983 ± 0.00197
	AsymLoop $K=3$	2048	141.38M	0.367	33.8 GiB	2.74947 ± 0.00299

E.3 TRAINING FLOPS VERIFICATION

We measure per-step training matrix FLOPs on H100 GPUs, including forward, backward, and activation recomputation. Table 11 shows that the widened Full Loop uses 1.23%–2.29% fewer FLOPs than AsymLoop, indicating closely matched training compute.

Table 11: **Measured matrix FLOPs per training step.** Counts include forward, backward, and activation recomputation, with sequence length 1024 and batch size 1.

	Measured GFLOPs/step		Widened Full Loop	
Model size	AsymLoop	Full Loop (wide)	FFN Width	Difference vs. AsymLoop
60M	1134.878	1108.907	2656	-2.29%
135M	2118.157	2092.198	2656	-1.23%

E.4 Discussion

AsymLoop achieves lower validation cross-entropy than the widened Full Loop at both model scales while using fewer parameters. Table 10 also shows comparable peak memory usage and per-step training times of the same order of magnitude. These results highlight AsymLoop's performance advantage at closely matched training FLOPs.

F Downstream Training Dynamics at 1B

We track the zero-shot performance of 1B Full Loop and AsymLoop throughout pre-training. Figure 6 presents the accuracy trajectories on seven downstream benchmarks and their average.

Both models improve with training, while AsymLoop develops a clearer advantage in average accuracy during the later stages. This trend is particularly evident on OpenBookQA, ARC-Easy, and HellaSwag, where AsymLoop maintains higher accuracy across successive evaluations toward the end of training.

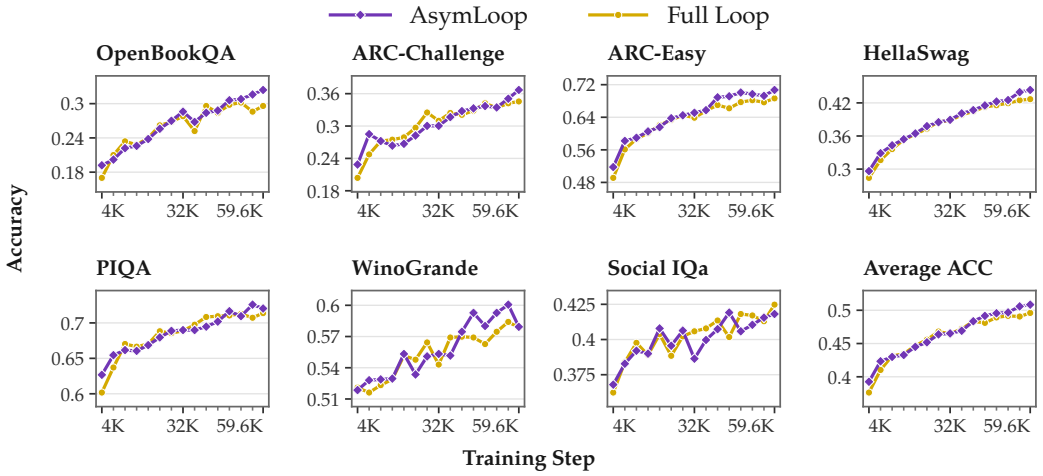

Figure 6: **Downstream performance during pre-training at 1B.** Zero-shot accuracy on seven benchmarks and their average. Purple and yellow denote AsymLoop and Full Loop, respectively.

G Transfer to an OLMo-Style Backbone

To assess generalization across backbone architectures, we evaluate AsymLoop on an OLMo-style backbone (Groeneveld et al., 2024). We compare Dense, Full Loop, and AsymLoop trained from scratch.

G.1 Experimental Setup

Architecture. All models use eight physical Transformer blocks with hidden dimension 512, eight attention heads, and SwiGLU intermediate dimension 2048, together with RMSNorm and RoPE. Full Loop repeats the backbone twice, while AsymLoop combines a four-layer encoder with a four-layer decoder repeated three times. The two looped models therefore have the same effective depth of 16.

Data and evaluation. We train all models on approximately 1.68B tokens from FineWeb-Edu-dedup using the SmolLM tokenizer, with sequence length 1024 and global batch size 512. Validation cross-entropy and perplexity are evaluated on a shared held-out set.

Optimization. We use AdamW with a cosine learning-rate schedule and 10% warmup. AsymLoop applies ALRS multipliers in $[1, 3]$, with allocation updates restricted to the first 20% of training.

G.2 Validation Results

Table 12: **Validation performance on an OLMo-style backbone.** Cross-entropy and perplexity (↓). Depth denotes effective depth. Darker/lighter blue marks the best/second-best results.

Model	Depth	CE ↓	PPL ↓
Dense	8	3.508664	33.403615
Full Loop ($K = 2$)	16	3.501389	33.161466
AsymLoop ($K = 3$)	16	**3.341073**	**28.249416**

Results. Table 12 shows that AsymLoop achieves the lowest validation cross-entropy and perplexity among all three models. Compared with the depth-matched Full Loop, AsymLoop reduces perplexity by 14.8%. These results demonstrate that its language-modeling benefits extend to an OLMo-style backbone, supporting its applicability across backbone architectures.

H Supervised Fine-Tuning on GSM8K

To evaluate mathematical reasoning after supervised fine-tuning, we fine-tune the 1B pretrained models on GSM8K (Cobbe et al., 2021). We compare Dense, Full Loop, and AsymLoop after three epochs of full-parameter adaptation.

H.1 Results after Three SFT Epochs

Table 13: **GSM8K accuracy of 1B models after three epochs of full-parameter supervised fine-tuning.** The best result is highlighted in blue.

Model	GSM8K Accuracy (%) ↑	Relative gain over Dense (%) ↑
Dense	8.94	–
Full Loop ($K = 2$)	10.24	+14.54
AsymLoop ($K = 3$)	**11.75**	**+31.43**

Table 13 shows that AsymLoop achieves the highest GSM8K accuracy among the three models. Under matched fine-tuning schedules, AsymLoop reaches 11.75% accuracy, improving over Full Loop by 1.52 percentage points. These results suggest that AsymLoop's pre-training benefits also extend to mathematical reasoning after supervised fine-tuning.

H.2 Training and Evaluation Protocols

Models and optimization. We fine-tune the 1B pretrained models using AdamW with a global batch size of 32, a peak learning rate of 10^{-4}, and cosine decay with 10% warmup. We report results after three epochs using a single random seed.

Data and supervision. We use 6,973 GSM8K training examples for supervised fine-tuning and reserve 500 for development, with a context length of 1024. Training supervises the worked solutions and final answers, excluding question and padding tokens from the loss.

Evaluation. We evaluate all 1,319 test questions using greedy decoding without demonstrations, allowing up to 512 generated tokens. Accuracy is computed by comparing the normalized final numerical answer with the reference.

I SPARSE MIXTURE-OF-EXPERTS MODELS

We investigate whether AsymLoop's benefits extend to sparse mixture-of-experts models (Muennighoff et al., 2025). Using the 60M and 135M backbone configurations, we compare MoE variants of Dense, Full Loop, and AsymLoop, where Dense denotes the non-recurrent architecture.

I.1 EXPERIMENTAL SETUP

We replace each feed-forward layer with eight SwiGLU experts and top-2 routing, without token dropping. Each expert has intermediate dimension 1024, preserving the active expert-matrix parameter count of the original dense FFN. Full Loop repeats the backbone twice, while AsymLoop divides it equally into an encoder and a decoder and repeats the decoder three times, matching the effective depth of Full Loop.

Experts and routers are shared across recurrences, with routing decisions recomputed at each iteration. Training combines cross-entropy with load-balancing and router z-loss terms weighted by 0.01 and 0.001, respectively. Validation cross-entropy excludes these auxiliary losses.

We train all models on FineWeb-Edu-dedup using AdamW with a cosine learning-rate schedule, sequence length 1024, and global batch size 512. The 60M and 135M configurations use 1.6B and 3B training tokens, respectively. We report validation cross-entropy and perplexity on a shared held-out set.

I.2 VALIDATION RESULTS

Table 14: **Validation performance of sparse MoE models.** Cross-entropy and perplexity (↓) with eight experts and top-2 routing. Darker/lighter blue marks the best/second-best results at each model scale.

Model	Effective depth	CE ↓	PPL ↓
60M configuration: 1.6B training tokens			
Dense + MoE	6	2.9856	19.798
Full Loop K2 + MoE	12	2.9363	18.845
AsymLoop K3 + MoE	12	**2.8986**	**18.149**
135M configuration: 3B training tokens			
Dense + MoE	12	2.7489	15.625
Full Loop K2 + MoE	24	2.7154	15.111
AsymLoop K3 + MoE	24	**2.6690**	**14.426**

Table 14 shows that AsymLoop achieves the lowest validation cross-entropy and perplexity at both model scales. At 135M, it reduces perplexity by 4.5% relative to the depth-matched Full Loop. These results show that AsymLoop remains effective with sparse experts, extending its applicability beyond dense feed-forward layers.