DUAL-SLOT: FROM SERIAL LOOPS TO PARALLEL SLOTS IN LANGUAGE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Looped language models increase computational depth by repeatedly applying shared layers, offering a way to improve model quality without adding parameters. However, sequential dependencies between loop iterations increase autoregressive decoding latency. We introduce **Dual-Slot**, an architecture that reorganizes serial looped computation into parallel latent and prediction slots. Rather than waiting for the current token's latent computation to finish, the prediction slot combines the previous token's deep latent state with the current representation, while the latent slot produces a state for the next decoding step. This shifts the latent dependency across adjacent tokens, replacing two sequential module invocations with a single paired invocation while preserving standard autoregressive generation. To preserve token-level parallelism during training, we use a small number of Jacobi iterations to refine latent states across the sequence in parallel. Experiments on Qwen-style dense and MoE models show consistent improvements in language modeling and downstream performance. On the 1B dense model, Dual-Slot reduces Pile validation perplexity from 7.25 to 7.00 and improves average five-shot accuracy across nine task configurations by 2.26 percentage points. After supervised fine-tuning, it improves zero-shot GSM8K accuracy by 5.83 and 3.49 percentage points for the 1B dense and A0.6B MoE models, respectively, while maintaining near-vanilla decoding latency.

1 INTRODUCTION

Increasing the computation applied to each token offers a way to improve language model quality without increasing model size. Looped language models implement this idea by repeatedly applying shared layers, increasing effective computational depth through parameter reuse (Dehghani et al., 2018; Giannou et al., 2023; Geiping et al., 2026).

However, parameter sharing does not eliminate the cost of repeated computation: successive loop iterations remain sequentially dependent and can make autoregressive decoding several times slower than a vanilla Transformer with the same parameter count. The challenge is therefore to benefit from recurrent computation without paying for multiple serial loop invocations at every decoding step.

To address this bottleneck, we introduce **Dual-Slot**, an architecture that reorganizes serial looped computation into parallel latent and prediction slots. Rather than waiting for the current token's latent computation to finish, the prediction slot combines the previous token's already-computed latent state with the current representation. Meanwhile, the latent slot processes the current representation to produce a state for the next decoding step. This shifts the latent dependency across adjacent tokens, replacing two sequential module invocations with a single paired invocation while preserving standard autoregressive generation.

While this design is natural during autoregressive decoding, it introduces a sequential dependency during teacher-forced training, where the latent states of preceding tokens are not yet available. We address this challenge using a small number of Jacobi iterations that refine latent states at all positions in parallel. This preserves token-level training parallelism while approximating the recurrent states used during decoding. We implement Dual-Slot in the middle third of a Transformer, leaving the prefix and suffix as single-pass components.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

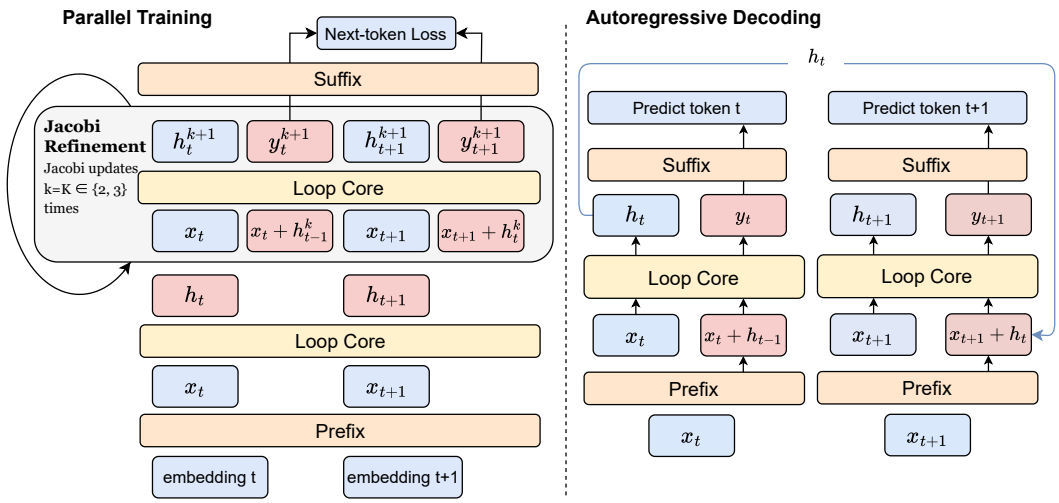


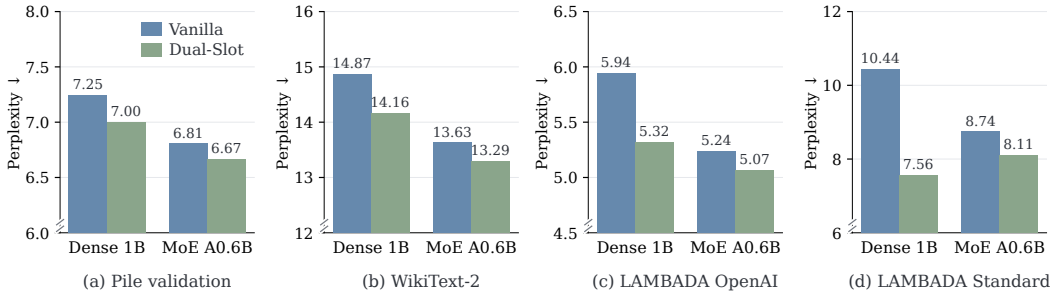
Figure 1: **Dual-Slot training and inference.** 1) **Parallel Training (left).** The prefix processes the full sequence once, and an initial warm pass through the core produces the initial latent slots. Jacobi updates then refine the slots at all positions in parallel, with next-token loss computed only from the final prediction states. 2) **Autoregressive Decoding (right).** At decoding step t, the shared Loop Core processes x_t and $x_t + h_{t-1}$ in one invocation. The prediction-slot output y_t predicts token $t + 1$, while the latent-slot output h_t is carried to the next step.

We train and evaluate a series of Qwen-style dense and MoE models (Yang et al., 2025). Dual-Slot consistently improves language modeling and average downstream accuracy over the corresponding vanilla models. For the 1B dense model, Pile validation perplexity decreases from 7.25 to 7.00, while average five-shot accuracy across nine task configurations improves by 2.26 percentage points. After supervised fine-tuning, Dual-Slot improves zero-shot GSM8K accuracy by 5.83 and 3.49 percentage points for the 1B dense and A0.6B MoE models, respectively. These improvements are achieved while maintaining near-vanilla decoding latency.

2 Related Work

Looped language models. Parameter sharing allows a Transformer to reuse the same layers, increasing computation depth without adding parameters. Early work applies shared layers recurrently across depth (Dehghani et al., 2018). Later studies use repeated shared modules to improve language modeling and reasoning (Giannou et al., 2023; Yang et al., 2024), while recurrent-depth models increase computation within the latent space (Geiping et al., 2026). Some methods reuse intermediate latent representations by feeding continuous outputs back into the model (Zeng et al., 2026) or refining the current hidden state as a latent thought (Zeng et al., 2025). However, these computations remain sequential, so decoding latency grows with the number of loops.

Efficient looped inference. Several methods reduce this cost through adaptive computation. Some determine whether to continue from the current state (Banino et al., 2021), while token-level methods allocate different amounts of computation to different tokens (Li et al., 2026). Dynamic routing and separate KV caches can support variable loop depths (Yu et al., 2026), but they introduce irregular execution and complicate batching, scheduling, and cache management. Other methods stagger computation across adjacent decoding steps (Cutler et al., 2025), execute loop stages from different tokens in one model invocation (Wu et al., 2025), or use a parameter-shared parallel prediction stream (Langford et al., 2026). However, their information flow remains tied to a predefined stage schedule. During autoregressive decoding, the previous token's latent state is available before the current step (Fan et al., 2020; Tang & Lu, 2026; Cai et al., 2026; Huang et al., 2026) and can support prediction while the current latent state is computed. Dual-Slot uses this property to compute a latent slot and a prediction slot in one shared-module invocation, with Jacobi iteration preserving token-level parallelism during training.



(a) Pile validation (b) WikiText-2 (c) LAMBADA OpenAI (d) LAMBADA Standard

Figure 2: **Language modeling performance of the 1B dense and A0.6B MoE models.** From left to right: perplexity on Pile validation, WikiText-2 (word-level), LAMBADA-OpenAI, and LAMBADA-Standard. Lower is better. Dual-Slot improves over Vanilla across all four benchmarks for both model families.

3 METHOD

Looped language models allocate additional computation to each token by repeatedly applying a shared module. Consider a two-loop computation and let F denote the looped module. Omitting the causal context for notational simplicity, the computation can be written schematically as:

$$\text{Loop 1:} \quad h_t = F(x_t); \qquad \text{Loop 2:} \quad y_t = F(h_t). \tag{1}$$

Here, x_t is the representation of token t entering the looped module, h_t is an intermediate latent representation, and y_t is used to predict token $t+1$. In this conventional formulation, Loop 2 depends on the output of Loop 1 and therefore cannot begin until Loop 1 has finished. More generally, L loops require L sequential applications of F along the autoregressive decoding path.

To address the decoding latency introduced by serial looping, we propose Dual-Slot, which introduces a latent slot and a prediction slot for each token. Neither slot requires the other's final output at the current step, allowing the two slots to advance through the shared module in parallel.

We first describe this decoding procedure and its parallel training rule while treating F as an abstract looped module. We then instantiate F as a partial loop over the middle layers of a Transformer in subsection 3.3.

3.1 DUAL-SLOT PARALLEL DECODING

At decoding step t, we define the two slot inputs as:

$$\text{Latent slot:} \quad x_t; \qquad \text{Prediction slot:} \quad \widetilde{x}_t = \mathrm{Mix}(x_t, h_{t-1}) = x_t + h_{t-1}, \tag{2}$$

where x_t is the current representation and h_{t-1} is the preceding token's latent-slot output. The carried history is set to zero at sequence boundaries.

The two slots share token t's position ID and are ordered causally, with the latent slot first. Given the attention key–value cache $\mathrm{KV}_{<t}$ from preceding steps, we evaluate them together:

$$(h_t, y_t) = F([x_t, \widetilde{x}_t];\, \mathrm{KV}_{<t})\,. \tag{3}$$

Here, h_t is retained for the next decoding step, while y_t is used to predict token $t+1$. During autoregressive decoding, h_{t-1} is already available from the preceding step, so both slot inputs are known before invoking F and the two slots can be processed in parallel.

3.2 PARALLEL TRAINING VIA JACOBI ITERATION

Sequential decoding naturally provides h_{t-1} before token t is processed. Teacher-forced pretraining, however, processes all tokens in a sequence simultaneously, before their latent-slot outputs are available. Naively reproducing the decoding procedure would create a left-to-right chain $h_1 \to h_2 \to \cdots \to h_T$ and remove the token-level parallelism of Transformer training. To preserve token-level parallelism while approximating the latent states induced by sequential inference, we use Jacobi iteration to refine all positions in parallel for a small number of rounds (Saad, 2003).

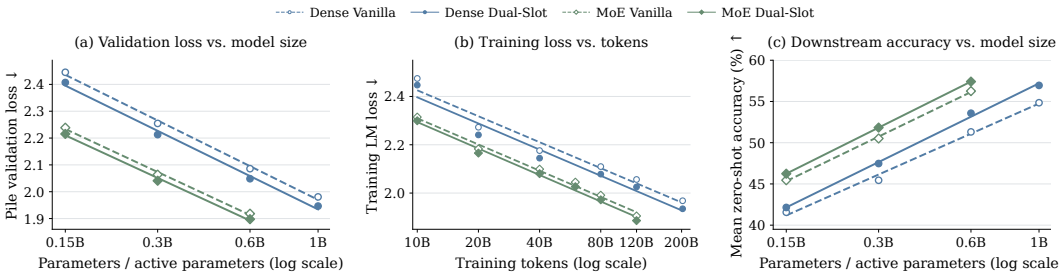


Figure 3: **Scaling behavior of Dual-Slot.** (a) Pile validation loss versus model size. (b) Loss versus training tokens for the 1B dense and A0.6B MoE models. (c) Average zero-shot accuracy over nine downstream task configurations versus model size. Model size denotes the nominal dense parameter count or the MoE active parameter count. Lower is better in (a) and (b); higher is better in (c).

Initial state estimation. Given the unexpanded input sequence $X = [x_1, \ldots, x_T]$, we first apply the looped module once without history injection:

$$H^{(0)} = [h_1^{(0)}, \ldots, h_T^{(0)}] = F(X). \tag{4}$$

Using $H^{(0)}$, we construct the initial prediction-slot inputs in parallel and insert them after their corresponding latent-slot inputs:

$$\widetilde{x}_t^{(0)} = x_t + h_{t-1}^{(0)}, \qquad \widetilde{X}^{(0)} = [x_1, \widetilde{x}_1^{(0)}, \ldots, x_T, \widetilde{x}_T^{(0)}]. \tag{5}$$

At sequence boundaries, the shifted history state is set to zero. We refer to Equation 4 as the *warm pass*; it provides the initial estimates used to construct $\widetilde{X}^{(0)}$ and is not counted as a Jacobi update.

Parallel state updates. At Jacobi round k, beginning with $k = 0$, we process the current interleaved sequence in one application of F:

$$\left(H^{(k+1)}, Y^{(k+1)}\right) = F\left(\widetilde{X}^{(k)}\right), \tag{6}$$

where $H^{(k+1)}$ and $Y^{(k+1)}$ collect the outputs at the latent and prediction slots, respectively. We then use the new latent-slot outputs to update all prediction-slot inputs in parallel and construct the sequence for the next round:

$$\widetilde{x}_t^{(k+1)} = x_t + h_{t-1}^{(k+1)}, \qquad \widetilde{X}^{(k+1)} = [x_1, \widetilde{x}_1^{(k+1)}, \ldots, x_T, \widetilde{x}_T^{(k+1)}]. \tag{7}$$

Training objective. After K updates, we apply the standard next-token cross-entropy objective only to the final prediction states $Y^{(K)}$:

$$\mathcal{L}_{\mathrm{LM}} = -\sum_{t=1}^{T-1} \log p_\theta\left(\mathrm{token}_{t+1} \mid y_t^{(K)}\right). \tag{8}$$

The latent-slot outputs receive no auxiliary language-modeling loss. To expose the model to variable refinement depths, we synchronously sample K from $\{2, 3\}$ for each training batch (Zeng et al., 2025). We report the empirical convergence behavior in section 4.

3.3 Partial-Loop Implementation

In practice, to balance model performance and computational cost, we apply Dual-Slot in a partial-loop form rather than looping the entire model. Only a contiguous set of middle Transformer layers is looped. The complete model can therefore be viewed as three components: a Prefix, a Loop Core, and a Suffix.

For each token, the Prefix is evaluated once to produce the loop input:

$$x_t = \mathrm{Prefix}(\mathrm{Embed}(\mathrm{token}_t)). \tag{9}$$

Table 1: **Downstream performance of the 1B dense and A0.6B MoE models.** Zero-shot and five-shot accuracy (%). Avg. is the unweighted mean over the nine task configurations. Bold indicates the best score within each model and shot setting, including ties.

Model	Method	LAMBADA		PIQA	Wino Grande	SciQ	Hella Swag	ARC		RACE high	Avg.
		OpenAI	Standard					Easy	Challenge		
Zero-shot											
1B Dense	Vanilla	62.64	52.42	71.93	**60.38**	89.30	42.09	60.82	**27.56**	35.31	55.83
	Dual-Slot	**64.95**	**56.69**	**72.58**	59.75	**90.20**	**43.74**	**61.95**	27.47	**36.56**	**57.10**
A0.6B MoE	Vanilla	64.43	52.61	72.85	**59.04**	88.60	44.12	63.05	26.79	**34.83**	56.26
	Dual-Slot	**64.70**	**55.62**	**73.23**	58.33	**89.00**	**45.33**	**65.07**	**31.06**	34.35	**57.41**
Five-shot											
1B Dense	Vanilla	57.62	49.18	72.58	60.06	**93.40**	41.68	63.38	29.10	34.55	55.73
	Dual-Slot	**58.86**	**56.14**	**73.23**	**62.59**	**93.40**	**43.75**	**65.70**	**30.97**	**37.22**	**57.98**
A0.6B MoE	Vanilla	57.97	49.47	73.78	60.22	92.20	43.88	65.78	30.46	**35.02**	56.53
	Dual-Slot	**58.78**	**52.30**	**74.10**	**61.09**	**94.60**	**45.65**	**67.97**	**32.94**	34.64	**58.01**

The Loop Core serves as F in Equation 3–Equation 6: only these middle layers operate on the paired latent and prediction slots. We then discard the latent stream and pass only the prediction output through the Suffix and the language modeling head:

$$\text{logits}_t = \text{LMHead}(\text{Suffix}(y_t)) \,. \tag{10}$$

Thus, token encoding in the Prefix and prediction decoding in the Suffix are performed once, while recurrent latent computation is confined to an intermediate representation space.

4 Experiments

We evaluate Dual-Slot from three perspectives: language modeling and downstream performance, scaling across model sizes and training budgets, and the convergence of its Jacobi refinement procedure. Our main results focus on a qwen-style 1B dense model and an A0.6B mixture-of-experts (MoE) model. At each scale, we compare Dual-Slot with the corresponding vanilla Transformer baseline.

4.1 Main Results

4.1.1 Language Modeling Ability

We evaluate the 1B dense and A0.6B MoE models on Pile validation (Gao et al., 2020), WikiText-2 (Merity et al., 2016), LAMBADA-OpenAI, and LAMBADA-Standard (Paperno et al., 2016), using word-level perplexity for WikiText-2. As shown in Figure 2, Dual-Slot consistently improves language modeling performance over Vanilla across all four benchmarks, achieving lower perplexity for both dense and MoE models.

4.1.2 Downstream Performance

We evaluate the 1B dense and A0.6B MoE models on nine downstream task configurations: LAMBADA-OpenAI, LAMBADA-Standard (Paperno et al., 2016), PIQA (Bisk et al., 2020), WinoGrande (Sakaguchi et al., 2021), SciQ (Welbl et al., 2017), HellaSwag (Zellers et al., 2019), ARC-Easy, ARC-Challenge (Clark et al., 2018), and RACE-high (Lai et al., 2017). Table 1 reports zero-shot and five-shot accuracy, together with the unweighted average over all nine configurations.

In zero-shot evaluation, Dual-Slot improves average accuracy from 55.83% to 57.10% for the 1B dense model and from 56.26% to 57.41% for the A0.6B MoE model, corresponding to gains of 1.27 and 1.15 percentage points, respectively. The gains also hold in the five-shot setting, where

Table 2: **Mathematical reasoning after supervised fine-tuning.** GSM8K test final-answer EM (%) under zero-shot and eight-shot prompting for the 1B dense and A0.6B MoE models. Dual-Slot consistently improves accuracy across both model families and evaluation settings.

Model	Method	Zero-shot		Eight-shot	
		EM (%) ↑	Gain (pp.)	EM (%) ↑	Gain (pp.)
1B Dense	Vanilla	13.12	–	10.84	–
1B Dense	Dual-Slot	**18.95**	+5.83	**14.94**	+4.10
A0.6B MoE	Vanilla	11.60	–	7.96	–
A0.6B MoE	Dual-Slot	**15.09**	+3.49	**11.83**	+3.87

average accuracy increases 2.26 and 1.48 percentage points. Although individual task gains vary, Dual-Slot consistently improves aggregate downstream performance across both model families and shot settings.

4.1.3 Mathematical Reasoning after Fine-Tuning

We evaluate mathematical reasoning on GSM8K (Cobbe et al., 2021) after supervised fine-tuning of the pretrained models. Table 2 reports final-answer exact-match accuracy under both zero-shot and eight-shot evaluation.

Dual-Slot consistently improves GSM8K performance across both model families and prompting settings. For the 1B dense model, zero-shot and eight-shot accuracy improve by 5.83 and 4.10 percentage points, respectively. The A0.6B MoE model shows similar gains of 3.49 and 3.87 points. These results indicate that the improvements from Dual-Slot persist after supervised fine-tuning and extend to mathematical reasoning.

4.2 Scaling Analysis

We study the scaling behavior of Dual-Slot using dense models with 150M, 300M, 600M, and 1B parameters, and MoE models with A0.15B, A0.3B, and A0.6B active parameters, with tpp200 (token-per-parameter). Figure 3 summarizes validation loss across model sizes, loss curves over training tokens, and average downstream accuracy across model sizes.

Scaling with model size. Figure 3(a) reports final Pile validation loss. Dual-Slot outperforms Vanilla at every evaluated scale, reducing loss by about 0.035 for dense models and 0.020 for MoE models. These improvements persist as model size increases.

Scaling with training tokens. Figure 3(b) plots loss against training tokens for the 1B dense and A0.6B MoE models. Dual-Slot maintains lower loss at every reported checkpoint for both model families, with the advantage persisting through approximately 202B and 120B training tokens.

Scaling of downstream performance. Figure 3(c) reports the unweighted average zero-shot raw accuracy over the nine task configurations in subsubsection 4.1.2. Dual-Slot improves average accuracy at every evaluated scale, with gains ranging from 0.58 to 2.27 percentage points for dense models and from 0.79 to 1.32 points for MoE models. Together with the validation-loss results, these findings demonstrate improvements in both language modeling and downstream performance across the evaluated scales.

4.3 Comparison with Parallel Loop Transformer

We further compare Dual-Slot with Parallel Loop Transformer (Wu et al., 2025), another approach for parallelizing looped computation. Following the dense scaling setup in subsection 4.2, we train PLT2 models at 150M, 300M, 600M, and 1B scales using the same settings.

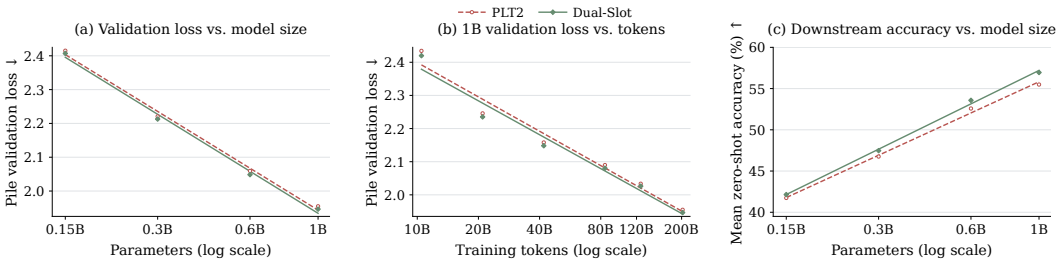


Figure 4: **Comparison with Parallel Loop Transformer (PLT2).** (a) Final Pile validation loss across dense model scales. (b) Pile validation loss over training tokens for the 1B models. (c) Average zero-shot raw accuracy over nine downstream task configurations across model scales. Dual-Slot consistently achieves lower validation loss and higher downstream accuracy than PLT2 across the evaluated scales.

As shown in Figure 4(a), Dual-Slot consistently achieves lower Pile validation loss across all four model sizes, with improvements of approximately 0.008–0.011 over PLT2. Figure 4(b) further shows that Dual-Slot maintains lower validation loss throughout the entire training trajectory.

The advantage also transfers to downstream evaluation. Figure 4(c) reports the unweighted average zero-shot accuracy over the same nine task configurations used in subsubsection 4.1.2. Dual-Slot consistently outperforms PLT2 at every evaluated scale, with the gain increasing from 0.41 percentage points at 150M to 1.46 points at 1B. Overall, these results show that Dual-Slot provides consistently stronger modeling quality than PLT2 under matched model scales and training-token budgets. More detailed results are shown in Appendix A

4.4 Decoding Efficiency

We evaluate the autoregressive decoding efficiency of Dual-Slot on a single GPU using BF16, SDPA, and the standard KV cache. Table 3 reports decoding throughput for the 1B dense and A0.6B MoE models across representative batch sizes.

Dual-Slot retains 93.1–96.5% of Vanilla throughput for the 1B dense model and 98.1–99.6% for the A0.6B MoE model. These results indicate that Dual-Slot introduces only modest decoding overhead. Throughput is measured over the decoding phase after a 128-token prefill, with 1,024 generated tokens.

Table 3: **Autoregressive decoding throughput.** Throughput (tokens/s) on a single GPU using BF16, SDPA, and the standard KV cache. DS/V denotes the throughput ratio between Dual-Slot and Vanilla.

1B Dense				**A0.6B MoE**			
BS	Vanilla	Dual-Slot	DS/V	BS	Vanilla	Dual-Slot	DS/V
1	80.51	74.92	0.9306×	1	17.25	16.92	0.9809×
4	307.53	292.29	0.9504×	4	68.29	67.97	0.9953×
32	2378.11	2294.87	0.9650×	32	549.12	546.74	0.9957×
128	9527.89	9056.98	0.9506×	128	2191.15	2162.76	0.9870×

4.5 Convergence of Jacobi Refinement

We examine the iterative behavior of Dual-Slot through evaluation loss and hidden-state changes as the number of Jacobi updates increases. Figure 5 reports both measurements for the 1B dense and A0.6B MoE models.

Evaluation loss. We evaluate both models with $K = 0, \ldots, 5$ Jacobi updates on 4096 samples. Evaluation loss is high at $K = 0$ and decreases rapidly as refinement proceeds, reaching low values

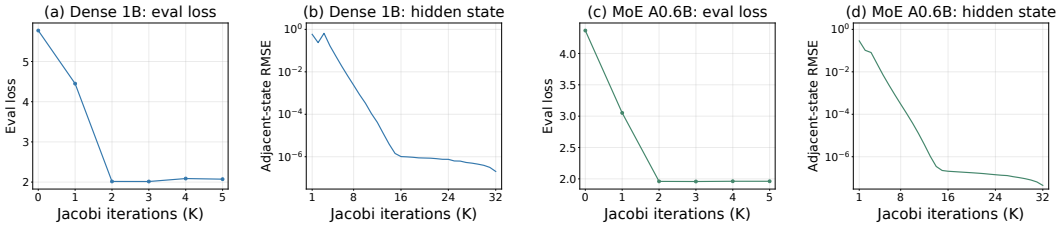


Figure 5: **Convergence of Jacobi refinement.** From left to right: evaluation loss and inter-iteration hidden-state RMSE for the 1B dense model, followed by the corresponding measurements for the A0.6B MoE model. Evaluation loss decreases rapidly up to the training depths ($K = 2, 3$) and changes comparatively little with further updates, while the RMSE between consecutive hidden states diminishes under continued refinement.

at $K = 2$ and $K = 3$, the depths used during training. Further iterations introduce only small changes, with both models remaining near the loss levels reached at their training depths.

Hidden-state convergence. We measure the change between consecutive iterations using the root-mean-square error (RMSE):

$$\mathrm{RMSE}^{(k)} = \frac{\left\| H^{(k)} - H^{(k-1)} \right\|_F}{\sqrt{N}}, \tag{11}$$

where $H^{(k)}$ denotes the hidden states at iteration k and N is the number of scalar elements included in the comparison. We continue increasing the number of Jacobi updates and observe that the changes between consecutive hidden states become progressively smaller for both models. At $K = 32$, the RMSE reaches 2.03×10^{-7} for the dense model and 4.47×10^{-8} for the MoE model, indicating empirical stabilization under continued refinement.

Together, these observations provide empirical evidence that training Dual-Slot with only two or three Jacobi updates is sufficient to learn stable and convergent iterative dynamics for models.

5 Ablation Studies

We conduct all ablations on the 600M dense model trained for 120B tokens. We examine three design choices: how to construct the prediction-slot input, where to place the loop core, and whether to use fixed or random K during training.

5.1 Prediction-Slot Fusion

We compare four prediction-slot fusion methods: direct addition (Raw Add), postmerge RMS matching, gated addition, and gated addition with postmerge RMS matching. RMS matching rescales the merged representation to match the RMS of x_t, while the learnable gate controls the contribution of h_{t-1} before addition.

As shown in Table 4, RMS matching increases validation loss both with (w/) and without (w/o) gating. Gated addition improves over Raw Add by only 0.001. We therefore retain Raw Add as our default for its simplicity and parameter-free formulation.

5.2 Loop-Core Location

With the loop-core depth fixed to one third of the model, we investigate whether the core should be placed in the first, middle, or final third.

As shown in Table 5, all three placements improve over Vanilla, but the middle-third configuration achieves the lowest validation loss. It reduces loss from 2.0859 to 2.0485, outperforming the first- and final-third configurations by 0.0105 and 0.0328, respectively. These results support our choice of the middle third as the default loop-core location.

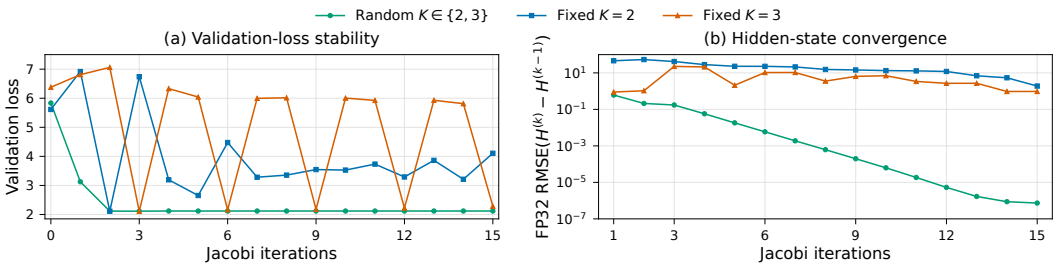


Figure 6: **Fixed versus random Jacobi iterations.** Validation loss (left) and inter-iteration hidden-state RMSE (right) for models trained with fixed $K = 2$, fixed $K = 3$, or random $K \in \{2, 3\}$. With random K, loss changes only slightly beyond the training iteration counts, and hidden states stabilize faster within the evaluated range.

Table 4: **Prediction-slot fusion.** Validation loss with (w/) and without (w/o) learnable gating on the 600M dense model trained for 120B tokens. Loss gaps are measured relative to Raw Add without a gate; negative values indicate improvement. The best loss in each group is bold.

Method	Val. loss ↓	Loss gap
Raw Add w/o gate	**2.0485**	0.0000
RMS matching w/o gate	2.0497	+0.0012
Raw Add w/ gate	**2.0474**	−0.0011
RMS matching w/ gate	2.0490	+0.0006

Table 5: **Loop-core location.** Validation loss for different placements of a seven-layer Dual-Slot core in a 21-layer Transformer. Loss gaps are measured relative to Vanilla; negative values indicate improvement. The best validation loss is bold.

Method	Val. loss ↓	Δ vs. Vanilla
Vanilla	2.0859	0.0000
Front7	2.0590	−0.0269
Middle7 (default)	**2.0485**	**−0.0374**
Last7	2.0813	−0.0046

5.3 Fixed versus Random Jacobi Iterations

We compare training with fixed $K = 2$, fixed $K = 3$, and random $K \in \{2, 3\}$ to examine the effect of randomizing the iteration count. Following subsection 4.5, we evaluate each trained model with increasing numbers of Jacobi iterations and examine both validation loss and inter-iteration hidden-state changes. As shown in Figure 6, models trained with a fixed K are more sensitive to the number of Jacobi iterations used at evaluation time. In contrast, the model trained with random $K \in \{2, 3\}$ shows only minor changes in validation loss as the number of iterations increases beyond those used during training.

The hidden-state results show the same trend. As additional Jacobi iterations are applied, the model trained with random K exhibits substantially faster decay in the change between consecutive hidden states, reaching an inter-iteration RMSE of 7.36×10^{-7} at $K = 15$, compared with 1.90 and 0.96 for fixed $K = 2$ and fixed $K = 3$, respectively.

Together, these results suggest that randomizing K during training reduces dependence on a particular iteration count and encourages more stable iterative dynamics. We therefore use random $K \in \{2, 3\}$ as the default training strategy.

6 Conclusion

We introduced Dual-Slot, an architecture that reorganizes serial looped computation into parallel latent and prediction slots. By carrying latent states across adjacent tokens, the two slots advance through shared layers in a single paired invocation while preserving standard autoregressive generation. A small number of Jacobi iterations preserves token-level parallelism during training, while randomizing $K \in \{2, 3\}$ encourages more stable predictions and hidden-state refinement beyond the iteration counts used during training. The experiment results highlight cross-token latent-state reuse as a practical way to improve language model quality while avoiding the within-token serial loop dependency of conventional looped models.

AI USE STATEMENT

Generative AI tools were used to assist with manuscript drafting, language editing and translation, LaTeX formatting, and the summarization and presentation of experimental results. The authors reviewed all AI-assisted content and take full responsibility for the final manuscript, including its methods, experimental results, claims, and associated artifacts.

ETHICS STATEMENT

This work studies architectural and training modifications to language models using existing datasets and benchmarks, without collecting new human-subject data. The proposed method does not directly address the broader safety risks of language models, which may generate biased, harmful, or privacy-sensitive content. Models using Dual-Slot should therefore undergo appropriate safety evaluation and employ suitable safeguards before deployment.

REPRODUCIBILITY STATEMENT

subsection 3.1, subsection 3.2, and subsection 3.3 specify the Dual-Slot decoding procedure, boundary handling, Jacobi training schedule, training objective, and partial-loop implementation. Appendix B provides complete dense and MoE model configurations and pretraining hyperparameters, while Appendix C documents the supervised fine-tuning and evaluation settings. Experimental protocols, scaling results, decoding efficiency, and convergence analyses are described in section 4, and section 5 reports controlled ablations of the main architectural and training choices.

REFERENCES

Joshua Ainslie, James Lee-Thorp, Michiel De Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. Gqa: Training generalized multi-query transformer models from multi-head checkpoints. In *Proceedings of the 2023 conference on empirical methods in natural language processing*, pp. 4895–4901, 2023.

Andrea Banino, Jan Balaguer, and Charles Blundell. Pondernet: Learning to ponder. *arXiv preprint arXiv:2107.05407*, 2021.

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.

Ziyang Cai, Xingyu Zhu, Yihe Dong, Yinghui He, and Sanjeev Arora. Tˆ 2mlr: Transformer with temporal middle-layer recurrence. *arXiv preprint arXiv:2607.15178*, 2026.

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, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Dylan Cutler, Arun Kandoor, Nishanth Dikkala, Nikunj Saunshi, Xin Wang, and Rina Panigrahy. Stagformer: Time staggering transformer decoding for runninglayers in parallel. *arXiv preprint arXiv:2501.15665*, 2025.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. Universal transformers. *arXiv preprint arXiv:1807.03819*, 2018.

Angela Fan, Thibaut Lavril, Edouard Grave, Armand Joulin, and Sainbayar Sukhbaatar. Addressing some limitations of transformers with feedback memory. *arXiv preprint arXiv:2002.09402*, 2020.

Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, et al. The pile: An 800gb dataset of diverse text for language modeling. *arXiv preprint arXiv:2101.00027*, 2020.

Jonas Geiping, Sean McLeish, Neel Jain, John Kirchenbauer, Siddharth Singh, Brian Bartoldson, Bhavya Kailkhura, Abhinav Bhatele, and Tom Goldstein. Scaling up test-time compute with latent reasoning: A recurrent depth approach. *Advances in Neural Information Processing Systems*, 38: 41340–41391, 2026.

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*, pp. 11398–11442. PMLR, 2023.

Zeyi Huang, Xuehai He, LiLiang Ren, Yiping Wang, Baolin Peng, Hao Cheng, Shuohang Wang, Pengcheng He, Jianfeng Gao, Yong Jae Lee, et al. Latent recurrent transformer: Architecture exploration, training strategies, and scaling behavior. *arXiv preprint arXiv:2605.26797*, 2026.

Guokun Lai, Qizhe Xie, Hanxiao Liu, Yiming Yang, and Eduard Hovy. Race: Large-scale reading comprehension dataset from examinations. In *Proceedings of the 2017 conference on empirical methods in natural language processing*, pp. 785–794, 2017.

John Langford, Nathan Godey, Giovanni Monea, Yoav Artzi, Harry Dong, Ying Fan, Gustavo de Rosa, and Zheng Zhan. Free pause tokens. *arXiv preprint arXiv:2609.03807*, 2026.

He Li, Feichen Song, Boyi Zeng, Shixiang Song, Zhiqin John Xu, Ziwei He, and Zhouhan Lin. Ponderlm-3: Adaptive token-wise pondering with differentiable masking. *arXiv preprint arXiv:2603.02023*, 2026.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*, 2016.

Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, Julie Bernauer, Bryan Catanzaro, et al. Efficient large-scale language model training on gpu clusters using megatron-lm. In *Proceedings of the international conference for high performance computing, networking, storage and analysis*, pp. 1–15, 2021.

Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Ngoc-Quan Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. The lambada dataset: Word prediction requiring a broad discourse context. In *Proceedings of the 54th annual meeting of the association for computational linguistics (volume 1: Long papers)*, pp. 1525–1534, 2016.

Yousef Saad. *Iterative methods for sparse linear systems*. SIAM, 2003.

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.

Noam Shazeer. Glu variants improve transformer. *arXiv preprint arXiv:2002.05202*, 2020.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.

Mohan Tang and Sidi Lu. Turbo connection: Reasoning as information flow from higher to lower layers. *arXiv preprint arXiv:2602.17993*, 2026.

Johannes Welbl, Nelson F Liu, and Matt Gardner. Crowdsourcing multiple choice science questions. In *Proceedings of the 3rd Workshop on Noisy User-generated Text*, pp. 94–106, 2017.

Bohong Wu, Mengzhao Chen, Xiang Luo, Shen Yan, Qifan Yu, Fan Xia, Tianqi Zhang, Hongrui Zhan, Zheng Zhong, Xun Zhou, et al. Parallel loop transformer for efficient test-time computation scaling. *arXiv preprint arXiv:2510.24824*, 2025.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Liu Yang, Kangwook Lee, Robert Nowak, and Dimitris Papailiopoulos. Looped transformers are better at learning learning algorithms. In *International conference on learning representations*, volume 2024, pp. 42195–42214, 2024.

Mingqian Yu, Wenpeng Zhang, and Peilin Zhao. T-loopformer: Token-level elastic-depth looped transformers for latent reasoning with dynamic routing. *arXiv preprint arXiv:2609.15160*, 2026.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? In *Proceedings of the 57th annual meeting of the association for computational linguistics*, pp. 4791–4800, 2019.

Boyi Zeng, He Li, Shixiang Song, Yixuan Wang, Zitong Wang, Ziwei He, Xinbing Wang, and Zhouhan Lin. Ponderlm-2: Pretraining llm with latent thoughts in continuous space. *arXiv preprint arXiv:2509.23184*, 2025.

Boyi Zeng, Shixiang Song, Siyuan Huang, Yixuan Wang, He Li, Ziwei He, Xinbing Wang, Zhouhan Lin, et al. Ponderlm: Pretraining language models to ponder in continuous space. In *International Conference on Learning Representations*, volume 2026, pp. 91874–91893, 2026.

Biao Zhang and Rico Sennrich. Root mean square layer normalization. *Advances in neural information processing systems*, 32, 2019.

A Detailed Comparison with Parallel Loop Transformer

For completeness, we report detailed zero-shot results for Vanilla, PLT2, and Dual-Slot across all four dense model scales. Table 6 includes Pile validation perplexity and raw accuracy on the same nine downstream task configurations used in subsubsection 4.1.2. Avg. denotes the unweighted mean over the nine downstream tasks.

Table 6: **Detailed comparison with Parallel Loop Transformer.** Pile validation perplexity (↓) and zero-shot raw accuracy (%, ↑) across four dense model scales. Avg. is the unweighted mean over the nine downstream task configurations. Bold indicates the best result within each model scale.

Method	Pile PPL	LAMBADA		PIQA	Wino Grande	SciQ	Hella Swag	ARC		RACE high	Avg.
		OpenAI	Standard					Easy	Challenge		
150M											
Vanilla	11.53	37.34	24.26	62.24	51.07	**75.80**	29.32	45.33	**19.37**	**29.38**	41.57
PLT2	11.19	**38.44**	25.00	**64.25**	50.20	74.80	29.28	46.04	19.11	28.52	41.74
Dual-Slot	**11.10**	38.11	**27.05**	63.93	**51.85**	75.70	**29.89**	**46.63**	18.77	28.90	**42.32**
300M											
Vanilla	9.53	45.62	35.82	66.38	49.96	78.40	32.33	49.16	21.08	30.24	45.44
PLT2	9.21	48.61	38.29	**66.70**	51.78	81.10	33.32	50.21	20.56	30.24	46.76
Dual-Slot	**9.14**	**49.29**	**39.86**	66.16	**51.93**	**81.50**	**33.33**	**51.18**	**21.33**	**31.58**	**47.35**
600M											
Vanilla	8.05	56.53	45.72	69.10	54.06	86.00	37.69	56.65	23.98	32.06	51.31
PLT2	7.84	58.47	47.84	**69.97**	53.67	86.20	38.82	**57.28**	**27.13**	33.88	52.58
Dual-Slot	**7.76**	**59.48**	**50.75**	68.93	**57.22**	**88.60**	**39.15**	57.03	24.74	**34.64**	**53.39**
1B											
Vanilla	7.25	62.64	52.42	71.93	**60.38**	89.30	42.09	60.82	**27.56**	35.31	55.83
PLT2	7.06	63.30	52.65	72.25	58.80	87.80	42.95	60.94	26.45	34.26	55.49
Dual-Slot	**7.00**	**64.95**	**56.69**	**72.58**	59.75	**90.20**	**43.74**	**61.95**	27.47	**36.56**	**57.10**

GSM8K after supervised fine-tuning. We further compare the 1B models on GSM8K after supervised fine-tuning. Under zero-shot evaluation, accuracy increases from 13.12% for Vanilla to 15.85% for PLT2, while Dual-Slot achieves 18.95%. Dual-Slot therefore improves over PLT2 by an additional 3.10 percentage points and over Vanilla by 5.83 points. PLT2 additionally achieves 13.34% under eight-shot prompting, while Dual-Slot gains an additional 1.6 percentage points.

B Model Configurations and Training Details

B.1 Model Architecture

Common backbone. All models are trained from random initialization and use a Qwen3-style decoder-only Transformer. The backbone combines grouped-query attention (GQA) (Ainslie et al., 2023), Q/K normalization, pre-RMSNorm (Zhang & Sennrich, 2019), SwiGLU with SiLU activations (Shazeer, 2020), and rotary position embeddings (RoPE) (Su et al., 2024). Attention projections have no bias terms. The vocabulary contains 50,304 tokens, and the input embedding and output language-model head share their weights. Unless stated otherwise, we set `rms_norm_eps`$= 10^{-6}$, `rope_theta`$= 10^6$, `attention_dropout`$= 0$, and `initializer_range`$= 0.02$. The attention head dimension is 64 at every model size.

Table 7: Dense model configurations. Vanilla and Dual-Slot have identical physical architectures and parameter counts. Dual-Slot core layers use zero-based indices.

Model	Parameters	Layers	d_{model}	d_{ff}	Q heads	KV heads	Dual-Slot core	Prefix/Core/Suffix
150M	149,541,120	12	768	3,328	12	4	L4–L7	4/4/4
300M	303,203,200	15	1,024	4,608	16	4	L5–L9	5/5/5
600M	604,627,328	21	1,280	5,632	20	5	L7–L13	7/7/7
1B	1,011,626,496	24	1,536	7,168	24	6	L8–L15	8/8/8

Table 8: MoE model configurations. Active parameters follow the conventional count for one slot with Top-8 routing. Each model has 64 routed experts, and core layers use zero-based indices.

Model	Active params.	Total params.	Layers	d_{model}	$d_{\text{ff}}^{\text{expert}}$	Q/KV heads	Dual-Slot core	Prefix/Core/Suffix
A0.15B	150,130,944	794,218,752	12	768	416	12/4	L4–L7	4/4/4
A0.3B	304,186,240	1,790,542,720	15	1,024	576	16/4	L5–L9	5/5/5
A0.6B	606,347,648	3,785,499,008	21	1,280	704	20/5	L7–L13	7/7/7

At each scale, Vanilla and Dual-Slot have identical parameter counts and physical Transformer layers. Dual-Slot introduces no additional parameters. Layer indices in this paper start at zero, whereas the implementation indexes the core's starting layer from one.

B.1.1 Dense Models

Table 7 lists the dense configurations. Depth, hidden dimension, and FFN width increase with model size, while the Dual-Slot core occupies the middle third of the layers.

B.1.2 Mixture-of-Experts Models

As shown in Table 8, MoE models are based on the dense configurations. Each Transformer layer has 64 routed experts, selects eight experts for each token, and has no shared expert. Each expert has 1/8 of the corresponding dense model's FFN width. Thus, the eight selected experts together have the same FFN width as the dense model. The two models have similar active parameter counts; the small difference mainly comes from the router. The MoE models use the same attention, normalization, positional encoding, vocabulary, tied embeddings, and Dual-Slot core placement as the dense models. Their output projections use Qwen-style unscaled initialization.

B.2 Training Setup

All models are pretrained with the causal next-token prediction objective on the Pile dataset. Training sequences contain 2,048 tokens. The global batch comprises 1,024 sequences, or 2,097,152 tokens per optimizer step, and the per-device micro-batch contains eight sequences.

Dense models use AdamW (Loshchilov & Hutter, 2017). MoE models use Megatron Adam with the distributed optimizer (Narayanan et al., 2021). Both use $\beta_1 = 0.9$, $\beta_2 = 0.95$, $\epsilon = 10^{-8}$, weight decay of 0.1, and global gradient-norm clipping at 1.0. The learning rate warms up linearly over the first 5% of training, then follows a cosine schedule to 10% of its peak value. The peak is 1.5×10^{-3} for the 150M, 300M, and 600M dense models and their MoE counterparts; 1.0×10^{-3} for the 1B dense. For MoE models, the Megatron optimizer maintains the main parameters and accumulated gradients in FP32. All runs use random seed 42, with activation recomputation disabled. At a given scale, Vanilla and Dual-Slot share the physical architecture, batch size, optimizer hyperparameters, learning-rate schedule, and training protocol unless an ablation specifies otherwise.

C Supervised Fine-Tuning Details

We perform full-parameter supervised fine-tuning (SFT) for mathematical reasoning on GSM8K. Vanilla and Dual-Slot start from their respective pretrained checkpoints and use identical fine-tuning data and hyperparameters. Dual-Slot retains its pretraining recurrence, with the number of Jacobi updates K sampled from $\{2, 3\}$ for each training batch.

GSM8K SFT uses three epochs, a global batch size of 128, a maximum sequence length of 2,048, and no sequence packing. Optimization uses AdamW with a peak learning rate of 2×10^{-5}, $\beta_1 = 0.9$, $\beta_2 = 0.95$, $\epsilon = 10^{-8}$, zero weight decay, and global gradient-norm clipping at 1.0. The learning rate follows a cosine schedule with a warmup ratio of 0.03. Training uses BF16 and random seed 42. Only response tokens contribute to the loss; prompt tokens are masked.

We use the official GSM8K training split of 7,473 examples. The official test split of 1,319 examples is used exclusively for evaluation. Each training input follows the format `Question: {question}\nAnswer:`, with the original GSM8K response as its target. The target includes the intermediate reasoning trace and the final answer marked by `####`.