T-LOOPFORMER: TOKEN-LEVEL ELASTIC-DEPTH LOOPED TRANSFORMERS WITH DYNAMIC ROUTING

Anonymous authors
Paper under double-blind review

ABSTRACT

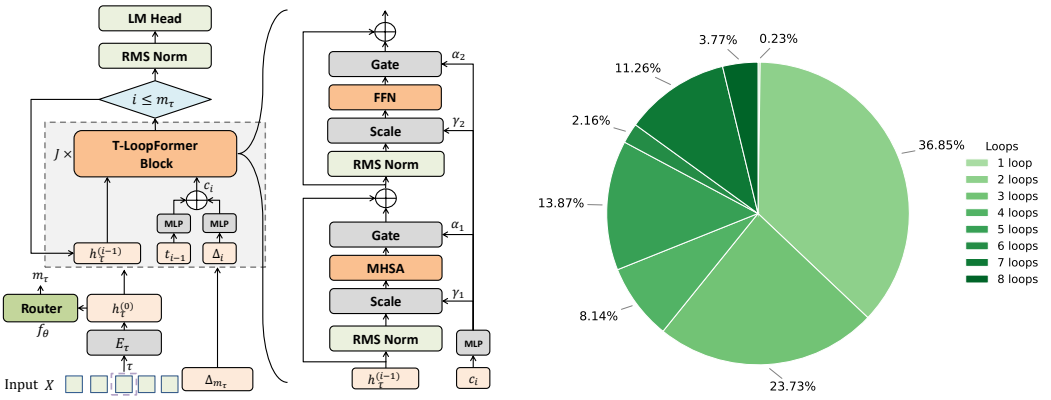
Looped Transformers have recently demonstrated strong performance in both reasoning and language tasks by reusing a shared set of parameters across multiple iterations, achieving parameter efficiency without sacrificing representational power. Besides, looped Transformers perform inference directly in the latent space (latent reasoning) to reduce the number of tokens consumed during inference, thereby achieving improved sample efficiency. However, these models typically apply a fixed recursion depth uniformly to every token, leading to suboptimal compute allocation and leaving significant efficiency gains on the table. In this work, we propose **t**oken-level elastic-depth **looped transformers**: T-LoopFormer. T-LoopFormer uses **dynamic token-choice routing**, which enables each token to adaptively determine its own number of loop iterations based on its hidden state, which could improve the token generation accuracy. Moreover, we further introduce recursion-wise KV cache, which maintains an independent key-value cache for each recursion loop of active tokens; this design ensures that tokens at different depths only attend to their corresponding cached states, effectively enabling faster autoregressive decoding. Extensive experiments show that T-LoopFormer achieves robust performance on a wide range of language modeling and zero-shot reasoning tasks, and our model could reach the lowest decoding latency, which validates the effectiveness of token-choice router and recursion-wise KV cache.

1 INTRODUCTION

Transformers with parameter sharing, often called looped or recurrent Transformers, have emerged as an efficient and capable alternative to deep non–shared stacks across vision and natural language (Jeddi et al., 2026; Dehghani et al., 2018; Lan et al., 2019; Geiping et al., 2026; Yu et al., 2026). Notably, looped Transformers exhibit strong performance on a broad range of algorithmic and reasoning tasks when applied to language modeling (Geiping et al., 2026; Jeddi et al., 2026; Saunshi et al., 2024). These models perform latent reasoning during inference, which allows elastic scaling of inference-time computational depth without increasing the parameter count, thereby improving computation efficiency and yielding better results on reasoning benchmarks and other downstream tasks and more computation in vanilla Transformers requires more layers, raising overhead (Geiping et al., 2026; Saunshi et al., 2024). However, a fixed number of loops is adopted by current approaches during training and inference.

Although there are methods that can achieve elastics-depth looped Transformers (Geiping et al., 2026; Jeddi et al., 2026), they do not provide **token-level elastic-depth** control over the number of loops. When these models perform reasoning recursively in latent space, they apply a uniform number of unrolling loops to all tokens, regardless of their individual hidden state (Jeddi et al., 2026; Geiping et al., 2026; Prairie et al., 2026; Xu & Sato, 2024). This monolithic strategy lacks token-level elasticity, meaning that different tokens recurse the same loop counts. As a result, the accuracy of the generated token will decrease, and the model incurs significant and often unnecessary memory overhead during training, which is particularly problematic when scaling to longer sequences or resource-constrained settings. Besides, because these models conduct recurrent reasoning in the latent space in decoding, they suffer from reduced inference throughput and consequently higher response latency (Wu et al., 2025; Dehghani et al., 2018). To overcome these questions, we propose **t**oken-level elastic-depth **looped transformers**: **T-LoopFormer**. T-LoopFormer can achieve token-level elastic-depth through a dynamic token-choice router, which can adaptively de-

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



(a) T-LoopFormer can achieve token-level elastic-depth through a dynamic token-choice router. The token-choice router determines the token's recurrent loops based on its hidden state, which can improve the token generation accuracy.

(b) The recurrent depth ratio of T-LoopFormer on FineWeb-Edu validation set. The average recurrent depth is 3.69 and exhibits long-tailed distribution, which validates that different tokens require different recurrent loops.

Figure 1: (a) The architecture of T-LoopFormer. (b) The loop ratio of T-LoopFormer for different tokens on the FineWeb-Edu validation set at 24x FLOPs.

termine one token should continue to recurse or exit early (Rahmath P et al., 2024; Yang et al., 2026). Specifically, a router assigns each token a fixed recursion depth based on its initial hidden state, the token then unrolls the shared block for that many loops before exiting. To accelerate the autoregressive decoding, T-LoopFormer employs recursion-wise KV cache (Shi et al., 2024; Li et al., 2024; Cai et al., 2024). Unlike approaches that retain key-value pairs for all tokens across depths, this strategy selectively caches KV entries only for tokens that remain active at each recursion loop. As tokens progressively exit in deeper recursions, the cache size naturally shrinks. Moreover, attention is restricted exclusively to these locally cached entries, which can accelerate the decoding.

The effectiveness of our T-Loopformer is thoroughly substantiated through extensive language modeling and zero-shot reasoning experiments. The dynamic token-choice routing mechanism grants the model the flexibility to modulate the number of recursive steps according to each token's hidden state, departing from the uniform-depth strategy that applies the same loop count indiscriminately, which decreases the accuracy of generated tokens. In addition, the recursion-wise KV cache effectively accelerates the autoregressive decoding by only retraining the active tokens in the loop. The key contributions of this paper are summarized as follows:

- We propose T-LoopFormer, which can realize token-level elastic-depth through a lightweight dynamic token-choice router. Unlike fixed-depth models, this dynamic token-choice router can adaptively determines whether the token should exit early or continue recursing based on its hidden state, different tokens require different recurrent depths, which can improve the token generation accuracy.
- We design a recursion-wise KV cache that selectively retains KV caches only for currently active tokens while discarding the KV caches of tokens that have exited the loop, thereby substantially accelerating autoregressive decoding of the model.
- Extensive experiments on a wide range of language modeling and reasoning benchmarks demonstrate that T-LoopFormer can achieve robust performance while delivering superior decoding efficiency compared with baselines. Moreover, ablation studies validate the effectiveness of the token-choice router and recursion-wise KV cache.

2 Related Work

Looped Transformers. Parameter sharing provides an orthogonal route to efficiency and effective depth (Jeddi et al., 2026). Through the introduction of adaptive computation time, the Universal Transformer establishes that repeatedly reusing a single block can achieve representational perfor-

mance comparable to that of deep stacks without weight sharing (Dehghani et al., 2018). During pretraining, ALBERT further reveals that extensive cross-layer weight tying achieves considerable parameter efficiency, while downstream performance remains uncompromised (Lan et al., 2019). Building upon this paradigm, Deep Equilibrium Models (DEQ) define an implicitly deep transformation with tied weights, solved iteratively via fixed-point methods and implicit differentiation (Bai et al., 2019). Besides, there are some works that explore looped transformers as programmable computers (Giannou et al., 2023) and mechanisms for algorithmic length generalization (Fan et al., 2025; Jeddi et al., 2026). What's more, Time-step-conditioned looping approaches include TMLT (Time-Modulated Looped Transformers) and LoopFormer, TMLT examine the representational capacity of looped Transformers in language modeling and demonstrate that conditioning on the timestep yields both improved scaling and lower perplexity (Xu & Sato, 2024). LoopFormer incorporates timestep and step-size conditioning into looped Transformers, via shortcut-consistency training over variable-length trajectories, it enables elastic-depth inference, where performance scales gracefully with the chosen compute budget (Jeddi et al., 2026). However, these methods treat all tokens identically, forcing each of them to undergo the same fixed number of recursion steps regardless of their own hidden states, which may lead to a decrease in accuracy when generating tokens.

Latent Reasoning. looped Transformers possess an inductive bias for reasoning that strengthens with increasing effective computational depth and such abilities are framed as latent reasoning (Saunshi et al., 2024; 2025). Unlike explicit chain-of-thought (CoT) prompting (Goyal et al., 2024; Cheng & Van Durme, 2024; Pfau et al., 2024; Kaissis et al., 2026; Chen et al., 2026; Zhu et al., 2025; Jolicoeur-Martineau, 2025), latent reasoning methods like looped transformers circumvent the need for verbalized reasoning steps, which significantly reduces the required context length and token consumption (Hao et al., 2024; Saunshi et al., 2025; Jeddi et al., 2026). Simultaneously, both theoretical and empirical researches link reasoning ability with network depth and algorithmic generalization (Merrill & Sabharwal, 2024). Our method leverages the inductive bias inherent in looped Transformers and extends it to the token-level, enabling dynamic loop unrolling at the token-level latent reasoning. This allows the autoregressive decoding process to operate directly on the hidden states of each individual token.

Dynamic Routing. Dynamic routing enhances model efficiency by adapting computation to input complexity. Capsule Networks introduced routing-by-agreement for dynamic part-whole assignment without static pooling (Sabour et al., 2017). In Transformers, ITT applies adaptive token-level routing with iterative hidden-state refinement, allowing critical tokens to undergo deeper computation while skipping redundant processing for simple tokens (Chen et al., 2025). In MoE, difficulty-aware routing activates more experts for complex inputs and fewer for easy ones, overcoming fixed top-k selection (Huang et al., 2024), while AdaMoE further enables token-adaptive allocation with null experts and differentiable end-to-end training (Zeng et al., 2024). MoD (Raposo et al., 2024) enables token-level dynamic compute in standard Transformers via top-k routing but its non-causal routing and discontinuous KV cache hinder parameter-tied looped architectures. In contrast to prior dynamic routing in standard Transformers, which selects among independent layers or experts, we extend dynamic routing to token-level loop unrolling within a parameter-tied looped Transformer, where each token's hidden state governs how many times the shared recurrent block is applied.

KV Cache. The KV cache has become a memory and bandwidth bottleneck for LLM inference as context lengths grow. **Eviction-based** methods selectively discard less critical KV pairs: CAKE (Qin et al., 2025) frames eviction as a cake-slicing problem that allocates per-layer cache sizes via spatial-temporal attention dynamics, achieving 10$\times$ speedup with 3.2% cache; CriticalKV (Feng et al., 2025) provides a perturbation-constrained selection algorithm that minimizes worst-case output perturbation, halving compression loss across benchmarks; ForesightKV (Dong et al., 2026) learns to predict eviction via supervised training and RL, outperforming prior methods under half the cache budget. **Quantization-based** methods reduce bit-width: PolarQuant (Wu et al., 2026) uses polar transformation to handle key cache outliers, converting query-key inner products into table lookups for decoding acceleration; CommVQ (Li et al., 2025) introduces commutative vector quantization with a RoPE-commutative codebook, achieving 87.5% cache reduction with 2-bit quantization. In this work, we propose a recursion-wise KV cache that retains KV caches only for active tokens in the loop, which is an exact, loop-indexed cache management method tailored to re-

current unrolling, rather than an eviction heuristic or a lossy-approximation KV cache method such as those mentioned above.

3 T-LOOPFORMER

We introduce T-LoopFormer (see Figure 1a), which is a looped transformer that utilizes a light weight dynamic token-choice router to achieve the token-level elastic-depth based on the hidden state of a token and use the recursion-wise KV cache during inference to accelerate autoregressive decoding. We use $X = (x_1, ..., x_K)$ to denote a sequence of K tokens which are drawn from a vocabulary V. Following LoopFormer (Jeddi et al., 2026), the positional embeddings are added in a one-shot manner for simplicity we use the simplest cycle design, where a stack of J Transformer layers denoted by $\Phi_J(\cdot)$.

3.1 LONG AND SHORT TRAJECTORY ALIGNMENT

We align the long and short trajectories produced during autoregressive decoding, ensuring that the model can flexibly adjust its recursion depth in the *sequence-level*. The establishment of a sentence-level global inference loops provides a computational upper bound for token-level elastic depth recursion and the number of loops per token is determined by a **dynamic token-choice router**. Following the work (Jeddi et al., 2026), T-LoopFormer uses a user-defined training loops L, the user specifies a step schedule Δ_L such that $\sum_{i=1}^{L} \Delta_i = 1$. T-LoopFormer then applies $\Phi_J(\cdot)$ for L iterations, conditioning each loop i on the cumulative time c_{i-1} and step size Δ_i, where $0 = c_0 < \cdots < c_L = 1$ and $\Delta_i = c_i - c_{i-1}$. The sequence $\Delta_L = (\Delta_1, \ldots, \Delta_L)$ is referred to as a *trajectory*. The full trajectory corresponds to the maximum L loops, each with $\Delta_i = 1/L$. Besides, T-LoopFormer is a decoder-only looped Transformer in which a single shared stack is applied iteratively. At iteration l, the model conditions on the pair (c_{i-1}, Δ_i), where $c_{i-1} \in [0, 1]$ is the cumulative normalized time and $\Delta_i \in [0, 1]$ is the step size. Encoded with sine–cosine frequency embeddings and projected via small MLPs, the two scalars yield e_c and e_Δ, and summing them produces $e_i = e_c + e_\Delta$. Modulation of the *T-LoopFormer Block* is performed via this signal: an MLP maps e_i to scaling (γ_1, γ_2) for the two RMSNorm layers and to gating (α_1, α_2) applied immediately before the residual connections of MHSA and FFN. T-LoopFormer uses shortcut conditioning together with an alignment loss $\mathcal{L}_{align}$ that drives trajectories of different lengths to align with the complete trajectory of length L. To sample the shortcut trajectory during training, we start from a T-LoopFormer with L (unrolled L times) and a maximum trajectory Δ_L. In each batch, we first draw a shortcut length $S \sim \mathcal{U}\{1, \ldots, L-1\}$, and subsequently sample the step schedule Δ_S uniformly over $[0, 1]$ subject to $\sum_{i=1}^{S} \Delta_i = 1$. The loss objective is:

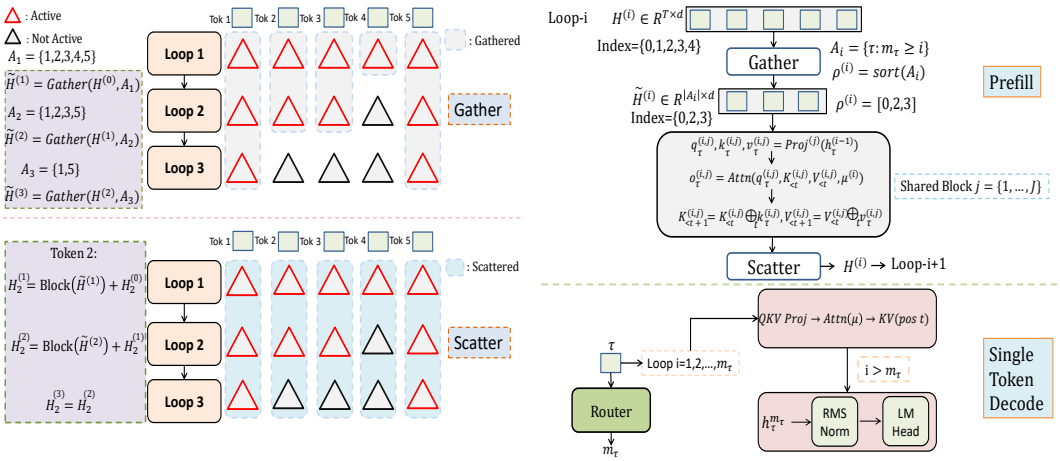
$$\mathcal{L} = \mathcal{L}_L + \lambda_1 \mathcal{L}_S + \lambda_2 \mathcal{L}_{align}, \tag{1}$$

where $\mathcal{L}_L$ and $\mathcal{L}_S$ correspond to the next-token prediction losses for the longest and sampled shortcut trajectories, respectively, and $\mathcal{L}_{align}$ is a stop-gradient loss aligning per-token logits of shorter trajectories to those of the longest trajectory. We set $\lambda_1 = \lambda_2 = 0.1$ in all experiments.

3.2 DYNAMIC TOKEN-CHOICE ROUTING

The implementation of token-level elastic-depth unroll is based on sequence-level elastic-depth expansion. We use a two-layer MLP to serve as the lightweight dynamic token-choice router. Loops are indexed by $i \in \{1, \ldots, M\}$, where M is the sequence-level maximum recursion depth shared by all tokens (during training, M equals the sampled shortcut length S, or L for the full trajectory; during inference, M is set by the user). Let $h_\tau^{(i)} \in \mathbb{R}^d$ denote the hidden state of token τ after i loops, with the initial embedding $h_\tau^{(0)}$ and ($h_\tau^{(0)} = E_\tau + E_{pos(\tau)} \in \mathbb{R}^d$); loop i maps $h_\tau^{(i-1)}$ to $h_\tau^{(i)}$. The router $f_\theta : \mathbb{R}^d \to \mathbb{R}^M$ maps the initial hidden state to a distribution over recursion depths via a softmax layer:

$$\pi_\tau = \text{softmax}\big(f_\theta(h_\tau^{(0)})\big) \in \mathbb{R}^M, \tag{2}$$



(a) Gather-scatter during training.

(b) Prefill and decode during inference.

Figure 2: T-LoopFormer training and inference pipeline. (a): the gather-scatter during training; (b): prefill and decode during inference.

where $\pi_{\tau,\ell}$ is the probability that token τ requires depth ℓ, $\ell \in \{1, \ldots, M\}$. For each loop i, the router computes the cumulative probability that token τ requires a recursion depth of at least i:

$$p_\tau^{(i)} = \sum_{\ell=i}^{M} \pi_{\tau,\ell}, \tag{3}$$

and token τ executes loop i (i.e., remains active) only if $p_\tau^{(i)} > 0.5$:

$$\mathbb{1}\left[\tau \in \mathcal{A}_i\right] = \mathbb{1}\left[p_\tau^{(i)} > 0.5\right], \tag{4}$$

where $\mathcal{A}_i \subseteq \{1, \ldots, T\}$ denotes the set of tokens still active at loop i. Since $p_\tau^{(i)}$ is non-increasing in i, the active sets are automatically nested, $\mathcal{A}_1 \supseteq \mathcal{A}_2 \supseteq \cdots \supseteq \mathcal{A}_M$, and $\mathcal{A}_1 = \{1, \ldots, T\}$ holds by construction as $p_\tau^{(1)} = 1$. A token is thus permanently deactivated once dropped (the routing decision is monotone over loops), and each token adaptively determines its own token-level recursion depth:

$$m_\tau = \max\{i \in \{1, \ldots, M\} : \tau \in \mathcal{A}_i\}, \qquad m_\tau \in \{1, \ldots, M\}, \tag{5}$$

where $m_\tau = 1$ means the token traverses only a single pass of the shared block, whereas $m_\tau = M$ indicates the token propagates through all M iterations. The trajectory is $\Delta_{m_\tau} = (\Delta_{m_1}, ..., \Delta_{m_\tau})$ and $\sum_{i=1}^{m_\tau} \Delta_i = 1$. Equivalently, $\tau \in \mathcal{A}_i$ if and only if $m_\tau \geq i$: the token's depth directly bounds the loops it participates in, with no index offset. As the routing distribution is computed once from the initial hidden state $h_\tau^{(0)}$, the per-token depth allocation is decided in a single forward evaluation of the lightweight router. Since the forward pass yields discrete values while backpropagation requires gradients, we employ the STE (Bengio et al., 2013) to address this issue. See Appendix C for more routing discussions.

Gather–Scatter computation for training efficiency. Because routing decisions vary per token, a naive implementation still executes the shared block over the full sequence at every loop, forfeiting any computational savings. We therefore couple the router with a gather-scatter mechanism that performs the lossless sequence compression during training. Let $H^{(i-1)} \in \mathbb{R}^{T\times d}$ collect the loop-i input hidden states of all T tokens. At each loop i, we first *gather* the hidden states of active tokens into a compacted tensor:

$$\tilde{H}^{(i)} = \text{Gather}\left(H^{(i-1)}, \mathcal{A}_i\right) \in \mathbb{R}^{|\mathcal{A}_i|\times d}, \tag{6}$$

so that the shared block operates only on $\tilde{H}^{(i)}$, and the computation cost of loop i scales with the number of surviving tokens $|\mathcal{A}_i|$ rather than the sequence length T. The updated states are

then *scattered* back to their original positions (this restores H as a full-length tensor in the original sequence order, so that (i) the next loop can gather its own active set $\mathcal{A}_{i+1}$, which differs from $\mathcal{A}_i$; (ii) token τ in each position retains its latest state at a fixed coordinate for loss alignment; and (iii) gradients can flow back through the indexed assignment to the corresponding rows in the compact tensor):

$$H_\tau^{(i)} = \begin{cases} \mathrm{Block}\big(\tilde{H}^{(i)}\big) + H_\tau^{(i-1)}, & \tau \in \mathcal{A}_i, \\ H_\tau^{(i-1)}, & \tau \notin \mathcal{A}_i. \end{cases} \tag{7}$$

Tokens that exit the recursion simply retain the hidden state from their last executed loop, $h_\tau^{(m_\tau)}$, which is passed to the final normalization and language-modeling head.

3.3 Recursion-wise KV Cache

During inference, T-LoopFormer employs a recursion-wise KV cache mechanism to retain the KV cache for tokens in active loops while discarding the KV cache for inactive loops, thereby accelerating token generation. For each token, we allocate M independent KV caches, one for each recursion loop $i \in \{1, \ldots, M\}$. Let $j \in \{1, \ldots, J\}$ denote the layer index within the shared block, where J is the total number of layers, $t^{'}$ are the historical keys positions. We define the cache state at recursion loop i, layer j, up to sequence position $t-1$ as

$$K_{<t}^{(i,j)} = \big(k_{t'}^{(i,j)}\big)_{t'<t}, \qquad V_{<t}^{(i,j)} = \big(v_{t'}^{(i,j)}\big)_{t'<t}. \tag{8}$$

Crucially, inference is also executed with the $gather-scatter$ scheme (see Appendix B). The active rows are first gathered into a compact tensor $\tilde{H}^{(i)}$ (equation 6), where the gather indices are sorted so that the causal order is preserved within the compact sequence. Only $\tilde{H}^{(i)}$ is passed through the shared block. Importantly, the original sequence positions $\rho^{(i)} = \mathrm{sort}(\mathcal{A}_i)$ are captured at gather time and propagated through the block to the cache-writing routine: within each attention layer, the newly computed keys and values are written into the loop-i cache at their original positions $\rho^{(i)}$ rather than at their compact indices. Positional bookkeeping maintains causal alignment via original-position cache references. After the block returns the updated compact states, they are scattered back to their original positions and get $H^{(i)}$ (equation 7), where $\mathrm{scatter}$ overwrites only the rows indexed by $\mathcal{A}_i$; the hidden states of exited tokens are left untouched. In the single-token decoding case, the scheme degenerates to an early exit: the recursion loop terminates as soon as the current token leaves the active set.

During autoregressive decoding of token τ (arriving at sequence position t, so that $K_{<t}$ holds exactly the tokens preceding it), its routing depth m_τ is determined by the router. For each recursion loop i, the token computes its query, key, and value projections at layer j from its incoming state:

$$q_\tau^{(i,j)},\, k_\tau^{(i,j)},\, v_\tau^{(i,j)} = \mathrm{Proj}^{(j)}\big(h_\tau^{(i-1)}\big). \tag{9}$$

The attention mechanism queries only the historical keys and values belonging to the same loop i, restricted to the valid entries of that cache:

$$o_\tau^{(i,j)} = \mathrm{Attn}\big(q_\tau^{(i,j)},\, K_{<t}^{(i,j)},\, V_{<t}^{(i,j)};\, \mu^{(i)}\big), \tag{10}$$

where the validity indicator $\mu_{t'}^{(i)} = \mathbb{1}[\, m_{t'} \geq i \,]$ excludes the cache slots of tokens that exited before loop i (in batched prefill, a block-causal mask is additionally applied among the currently gathered tokens). Masking is required to avoid softmax assigning non-negligible mass to zero slots of exited tokens.

Whenever token τ executes loop i (i.e., $m_\tau \geq i$), its current key and value must be stored, since future tokens that also reach loop i will attend to them. We update the loop-specific cache by writing the new states at the original sequence position:

$$K_{<t+1}^{(i,j)} = K_{<t}^{(i,j)} \oplus_t k_\tau^{(i,j)}, \qquad V_{<t+1}^{(i,j)} = V_{<t}^{(i,j)} \oplus_t v_\tau^{(i,j)}, \tag{11}$$

where $\oplus_t$ denotes writing the new state at cache position t. For single-token decoding, this reduces to ordinary sequence concatenation at the tail of the cache; for batched prefill, the writes are indexed by the gathered positions $\rho^{(i)}$ of all active tokens. If the token exits the recursion after loop m_τ (i.e., for loops $i > m_\tau$), it neither computes new projections nor updates any cache. Its final hidden state $h_\tau^{(m_\tau)}$ is directly passed to the final normalization and LM head, bypassing the remaining shared block iterations.

Table 1: The perplexity comparison on the FineWeb-Edu and OpenWebText validation sets and the accuracy comparison on ten zero-shot reasoning tasks between our model and all baseline models under various FLOPs. The best performance is marked in **bold**, the second-best is underlined, and our model's results are highlighted.

	Params	Perplexity ↓		Language Tasks (Accuracy) ↑										Avg Acc
		FineWeb-Edu	OpenWebText	COPA	HS	LB	OBQA	PIQA	Race	SciQ	ARC-C	SIQA	ARC-E	
FLOPs: 24x														
Base (24 ⊗ 1)	24x	20.17	19.22	62	**34.79**	**41.98**	27.57	66.29	29.78	**70.22**	27.79	38.83	38.76	43.8
Base-Loop (3 ⊗ 8)	3x	23.23	22.78	61	31.65	35.83	27.11	64.45	28.93	63.79	25.11	**38.94**	36.12	41.29
TMLT (3 ⊗ 8)	3x	22.87	21.33	66	33.22	40.06	27.92	64.11	30.14	70.11	25.55	37.32	36.58	43.1
Naive-Loop-EE (3 ⊗ 8)	3x	25.54	24.23	67	30.78	31.89	27.03	63.42	29.56	66.32	25.88	36.97	36.77	41.56
Base-Loop-EE-Align (3 ⊗ 8)	3x	24.21	23.25	67	31.46	32.92	26.9	63.23	29.43	65.29	26.77	37.81	37.83	41.86
TMLT-EE (3 ⊗ 8)	3x	23.47	21.98	67	32.44	36.92	**28.4**	64.14	28.9	68.1	26.68	37.73	37.66	42.80
LoopFormer(3 ⊗ 8)	3x	21.96	20.09	**68**	32.93	39.32	26.7	64.95	31.19	68.44	27.56	38.19	37.49	43.48
T-LoopFormer(3 ⊗ 8)	3x	**20.13**	**19.14**	**68**	33.67	40.18	27.33	**66.32**	**31.88**	68.76	**28.04**	38.68	**39.17**	**44.20**
FLOPs: 12x														
Base (12 ⊗ 1)	12x	**21.69**	**21.11**	68	**32.82**	**37.78**	**26.28**	64.69	**29.88**	**69.21**	26.31	38.41	38.17	**43.16**
Naive-Loop-EE (3 ⊗ 4)	3x	24.79	25.12	66	29.67	31.69	26.19	61.95	28.53	64.56	25.23	37.12	36.24	40.72
Base-Loop-EE-Align (3 ⊗ 4)	3x	26.01	25.83	63	29.98	26.07	27.43	61.63	28.26	58.89	25.25	36.35	36.74	39.36
TMLT-EE (3 ⊗ 4)	3x	26.36	26.77	62	30.6	27.94	26.79	62.31	28.85	61.74	25.72	36.78	36.98	39.97
LoopFormer(3 ⊗ 4)	3x	24.23	23.69	68	31.55	32.54	25.68	63.99	28.51	66.73	26.28	37.82	37.14	41.82
T-LoopFormer(3 ⊗ 4)	3x	23.6	23.14	**69**	32.33	33.26	26.19	**64.82**	28.79	67.11	**26.96**	**38.47**	**38.85**	42.58
FLOPs: 6x														
Base (6 ⊗ 1)	6x	**24.84**	**24.04**	**64**	**30.58**	**33.78**	25.78	**62.58**	**28.31**	**67.61**	**31.16**	36.21	37.66	**41.77**
Naive-Loop-EE (3 ⊗ 2)	3x	29.03	28.12	63	28.86	27.51	24.82	61.78	26.49	62.38	30.02	35.53	36.13	39.65
Base-Loop-EE-Align (3 ⊗ 2)	3x	35.07	34.32	60	28.33	18.88	25.67	59.88	25.24	54.02	29.05	34.79	36.23	37.21
TMLT-EE (3 ⊗ 2)	3x	37.31	37.22	59	28.34	17.45	25.37	59.11	26.51	50.41	28.54	35.16	36.53	36.64
LoopFormer (3 ⊗ 2)	3x	32.89	32.05	63	28.81	26.68	26.87	60.67	26.03	59.02	29.67	35.77	37.02	39.35
T-LoopFormer (3 ⊗ 2)	3x	32.03	31.76	**64**	29.17	27.03	**27.35**	61.13	26.5	59.64	30.44	**36.46**	**38.18**	39.99

Table 2: TPOT of different models on FineWeb-Edu validation set at 24× FLOPs.

Model	Latency (ms/token)
Base (24 ⊗ 1)	0.202
Base-Loop (3 ⊗ 8)	0.497
TMLT (3 ⊗ 8)	0.389
Naive-Loop-EE (3 ⊗ 8)	0.433
Base-Loop-EE-Align (3 ⊗ 8)	0.425
TMLT-EE (3 ⊗ 8)	0.312
LoopFormer (3 ⊗ 8)	0.441
T-LoopFormer (3 ⊗ 8)	**0.197**

Table 3: Training memory of different models on FineWeb-Edu training set at 24× FLOPs

Model	Memory (MB)
Base (24 ⊗ 1)	634,225
Base-Loop (3 ⊗ 8)	93,442
TMLT (3 ⊗ 8)	96,594
Naive-Loop-EE (3 ⊗ 8)	87,997
Base-Loop-EE-Align (3 ⊗ 8)	91,256
TMLT-EE (3 ⊗ 8)	98,387
LoopFormer (3 ⊗ 8)	97,739
T-LoopFormer (3 ⊗ 8)	**74,889**

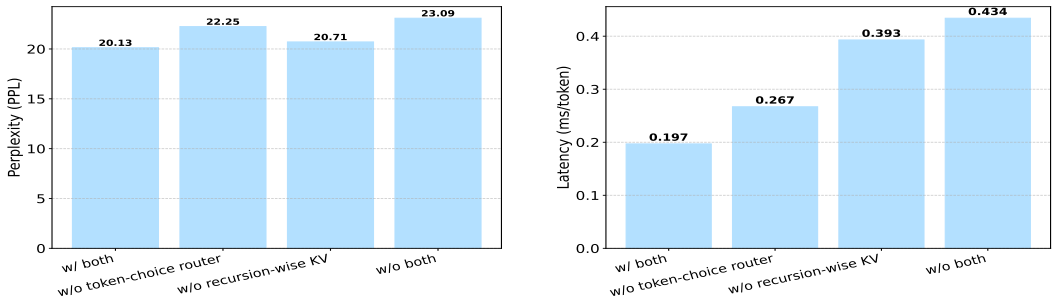
4 EXPERIMENTS

Following (Jeddi et al., 2026), we compare a 24-layer, ∼1B-parameter non-looped Transformer with FLOP-matched looped variants. All models use a GPT-style decoder (Radford et al., 2019) with NanoGPT configurations. Training is performed on FineWeb-Edu (Penedo et al., 2024) for 25B tokens in accordance with Chinchilla scaling (Hoffmann et al., 2022). See Appendix A for details. Following (Jeddi et al., 2026), in terms of model parameters, 24× indicates that the model contains 24 layers. In terms of FLOPS, 24× denotes the product of the number of model layers J and the user-specified inference depth M, $FLOPs \propto J \otimes M$, ignoring embedding/unembedding costs. The token-choice router threshold we set is 0.5.

Evaluation Metrics and Benchmarks. Following (Saunshi et al., 2025; Geiping et al., 2026; Jeddi et al., 2026), we report perplexity and downstream zero-shot accuracy. Perplexity is measured on FineWeb-Edu (Penedo et al., 2024) and OpenWebText (Gokaslan & Cohen, 2019). For latent reasoning, we report zero-shot accuracy on ten established benchmarks spanning a range of reasoning difficulty: COPA (Roemmele et al., 2011), HellaSwag (HS) (Zellers et al., 2019), LAM-BADA (LB) (Paperno et al., 2016), OpenBookQA (OBQA) (Mihaylov et al., 2018), RACE (Lai et al., 2017), Social IQA (SIQA) (Sap et al., 2019), ARC-Easy (ARC-E), ARC-Challenge (ARC-C) (Clark et al., 2018), and SciQ (Welbl et al., 2017).

Baselines. We compare T-LoopFormer against two groups of baselines: fixed-depth models and depth-elastic models. Fixed-depth models include Base: a non-looped Transformer; Base-Loop: a standard looped model; TMLT: a looped model with timestep conditioning. Depth-elastic models in-

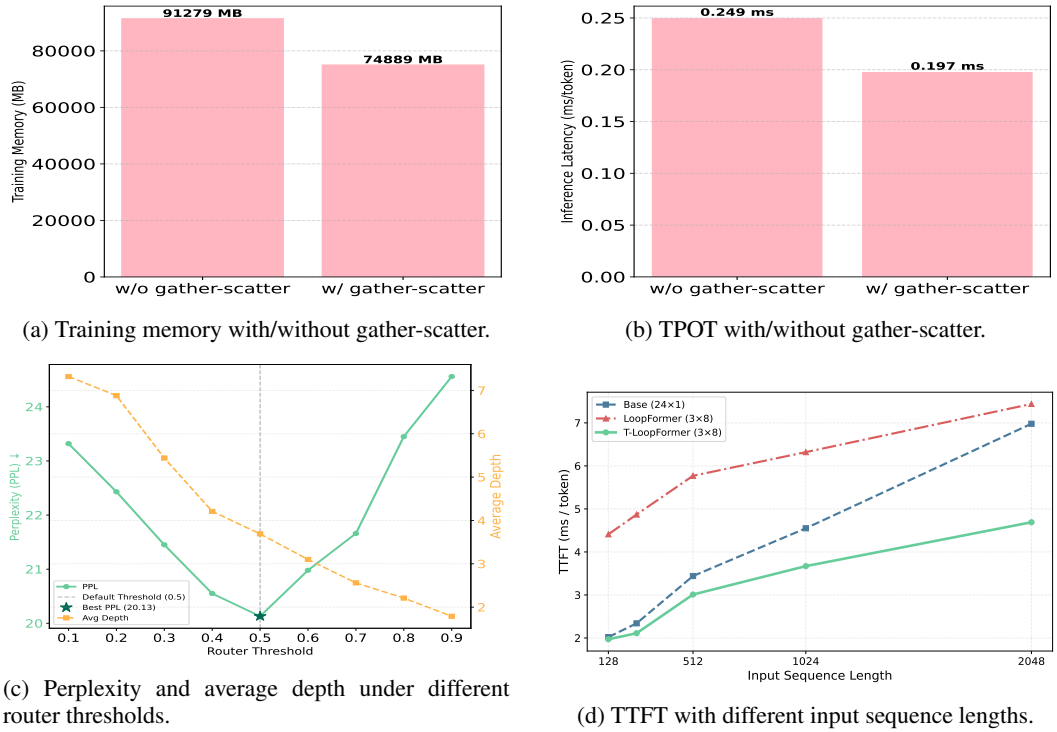
clude Base-Loop-EE: naive early exiting applied to the basic looped model; TMLT-EE: early exiting and long-short trajectory alignment training applied to TMLT to enable depth elasticity; Base-Loop-EE-Align: augmented with the long-short trajectory alignment during training. LoopFormer: an elastic-depth looped transformers with long-short trajectory alignment.



(a) Perplexity (PPL)

(b) TPOT

Figure 3: The influence of token-choice router and recursion-wise KV cache. Tested on FineWeb-Edu validation set at 24× FLOPs. (a) PPL, (b) TPOT.



(a) Training memory with/without gather-scatter.

(b) TPOT with/without gather-scatter.

(c) Perplexity and average depth under different router thresholds.

(d) TTFT with different input sequence lengths.

Figure 4: Ablation studies on key components of T-LoopFormer. (a) and (b): effect of the gather–scatter mechanism on training memory and inference latency; (c): sensitivity analysis of the router threshold; (d): scaling of TTFT with input sequence length.

4.1 Main Results

Table 1 shows that, under different FLOPs, T-LoopFormer achieves the best reasoning performance among models with the same parameter. Moreover, at 24× FLOPs, our model outperforms the non-looped base model with 24× parameter. This demonstrates that the dynamic token-choice routing mechanism designed in our model enables each token to adaptively unroll different number of loops

based on its hidden state, and that either excessive or insufficient elastic-depth unrolling may lead to incorrect token outputs. Under 12x and 6x FLOPs, because the maximum number of loops is reduced, some tokens require more loop to unroll, which causes those tokens to be insufficiently reasoned. As a result, T-LoopFormer underperforms the non-looped base model on perplexity and zero-shot reasoning tasks. However, compared with other looped models, it narrows the performance gap with the non-looped base model (see Appendix D for more discussions). In addition to the comparison of model performance, Table 2 shows that our model achieves the lowest decoding latency among all baselines, even lower than the non-looped base model, demonstrating that our model can improve single token decoding efficiency. Moreover, compared with LoopFormer, our model achieves 2.24× decoding speed faster than LoopFormer, which confirms that the dynamic token-choice routing mechanism and the recursion-wise KV cache are effective in improving inference speed. Table 3 validates that T-LoopFormer can achieve the lowest memory usage during training, indicating that it can improve the training efficiency. Figure 1b shows the distribution of loop counts per token on the FineWeb-Edu validation set and most tokens require at most 4 loops (average depth is 3.69), and the overall distribution exhibits a long-tail pattern. This indicates that different tokens need different numbers of loops, validating the effectiveness of token-choice router.

4.2 Ablation Study

Influence of Token-Choice Router and Recursion-wise KV Cache. Figure 3a exhibits that the token-choice router contributes more to generating correct tokens (without router, PPL increases 2.12, without recursion-wise KV cache, PPL increases 0.58) and figure 3b validates that the recursion-wise KV cache contributes more to increasing the decoding speed (without recursion-wise KV cache, TPOT increases 0.196 ms and without router, TPOT increases 0.70 ms).

Influence of Gather-Scatter during Training and Inference. Figure 4a shows that without gather-scatter during training, memory increases 21.9%, which validates that the gather-scatter mechanism can improve the training efficiency and figure 4b shows that without gather-scatter during inference, the inference latency increases 26.4% (tested on FineWeb-Edu validation set). These validate that gather-scatter mechanism can improve the autoregressive decoding efficiency.

Influence of Different Router Threshold. Figure 4c shows the influence of different router threshold (tested on FineWeb-Edu validation set at 24x FLOPs), 0.5 is the best threshold and the perplexity (PPL) of the model equipped with the token-choice router on the FineWeb-Edu validation set exhibits a U-shaped curve as the threshold varies.

TTFT with Different Input Sequence Lengths. Figure 4d shows the TTFT (time to first token) with different input sequence lengths (128, 512, 1024, and 2048), we randomly sample 1,000 documents from FineWeb-Edu validation set, each with a raw length exceeding 2,048 tokens. We then truncate each to target lengths 128, 512, 1024, 2048 as the input prompt. The results show that, as input length grows, T-LoopFormer shows a smaller increase in TTFT than the base model and achieves the lowest latency across all sequence lengths, demonstrating its efficient scaling with input length.

5 Conclusion

We build **T-LoopFormer**, a looped Transformer with token-level elastic-depth for latent reasoning. A dynamic token-choice router adaptively assigns each token a variable number of recursion loops, allowing different tokens receive different computations. For efficiency, we incorporate a gather–scatter mechanism during training and inference, which restricts each loop to active tokens, reducing training memory and decoding latency, and a recursion-wise KV cache that discards caches of exited tokens and keeps KV caches of active tokens in each loop to further accelerate autoregressive decoding. Together, these designs yield an efficient and flexible architecture. Experiments show that T-LoopFormer reaches the robust performance under the same parameters on language modeling and zero-shot reasoning tasks (even surpassing base Transformer at 24× FLOPs) and can reach the lowest decoding latency, which validates the effectiveness of our model. Future work will extend T-LoopFormer to larger-scale pretraining and broader reasoning tasks.

6 AI USE STATEMENT

During the preparation of this manuscript, the authors used DeepSeek solely for language polishing and for assisting with data preprocessing, including drafting or refactoring preprocessing code and checking data-cleaning scripts. The tool was not used to generate scientific ideas, formulate methods, conduct experiments, interpret results, or write the core scientific claims. All AI-assisted outputs were carefully reviewed, verified, and edited by the authors, who take full responsibility for the content, integrity, and reproducibility of this work. The final data preprocessing was carried out by deterministic scripts.

7 REPRODUCIBILITY STATEMENT

To support reproducibility, we provide a complete description of the proposed method, experimental setup, and evaluation protocol in Section 4 and Appendix A. The source code, hyperparameters, random seeds, and step-by-step instructions for reproducing the main results are available in the anonymous supplementary repository at `https://anonymous.4open.science/r/T-LoopFormer-53BE`. We believe these resources are sufficient for reproducing the results reported in this paper.

REFERENCES

Shaojie Bai, J Zico Kolter, and Vladlen Koltun. Deep equilibrium models. *Advances in neural information processing systems*, 32, 2019.

Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

Zefan Cai, Yichi Zhang, Bofei Gao, Yuliang Liu, Yucheng Li, Tianyu Liu, Keming Lu, Wayne Xiong, Yue Dong, Junjie Hu, et al. Pyramidkv: Dynamic kv cache compression based on pyramidal information funneling. *arXiv preprint arXiv:2406.02069*, 2024.

Guanxu Chen, Dongrui Liu, and Jing Shao. Loop as a bridge: Can looped transformers truly link representation space and natural language outputs? *arXiv preprint arXiv:2601.10242*, 2026.

Yilong Chen, Junyuan Shang, Zhenyu Zhang, Yanxi Xie, Jiawei Sheng, Tingwen Liu, Shuohuan Wang, Yu Sun, Hua Wu, and Haifeng Wang. Inner thinking transformer: Leveraging dynamic depth scaling to foster adaptive internal thinking. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 28241–28259, 2025.

Jeffrey Cheng and Benjamin Van Durme. Compressed chain of thought: Efficient reasoning through dense representations. *arXiv preprint arXiv:2412.13171*, 2024.

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.

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

Zican Dong, Peiyu Liu, Junyi Li, Zhipeng Chen, Han Peng, Shuo Wang, and Wayne Xin Zhao. Foresightkv: Optimizing kv cache eviction for reasoning models by learning long-term contribution. *arXiv preprint arXiv:2602.03203*, 2026.

Ying Fan, Yilun Du, Kannan Ramchandran, and Kangwook Lee. Looped transformers for length generalization. In *International Conference on Learning Representations*, volume 2025, pp. 14502–14520, 2025.

Yuan Feng, Junlin Lv, Haoyu Guo, Yukun Cao, S Kevin Zhou, and Xike Xie. Criticalkv: Optimizing kv cache eviction from an output perturbation perspective. *arXiv preprint arXiv:2502.03805*, 2025.

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.

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

Sachin Goyal, Ziwei Ji, Ankit Singh Rawat, Aditya Krishna Menon, Sanjiv Kumar, and Vaishnavh Nagarajan. Think before you speak: Training language models with pause tokens. In *International Conference on Learning Representations*, volume 2024, pp. 27896–27923, 2024.

Shibo Hao, Sainbayar Sukhbaatar, DiJia Su, Xian Li, Zhiting Hu, Jason Weston, and Yuandong Tian. Training large language models to reason in a continuous latent space. *arXiv preprint arXiv:2412.06769*, 2024.

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.

Quzhe Huang, Zhenwei An, Nan Zhuang, Mingxu Tao, Chen Zhang, Yang Jin, Kun Xu, Liwei Chen, Songfang Huang, and Yansong Feng. Harder task needs more experts: Dynamic routing in moe models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12883–12895, 2024.

Ahmadreza Jeddi, Marco Ciccone, and Babak Taati. Loopformer: Elastic-depth looped transformers for latent reasoning via shortcut modulation. *arXiv preprint arXiv:2602.11451*, 2026.

Alexia Jolicoeur-Martineau. Less is more: Recursive reasoning with tiny networks. *arXiv preprint arXiv:2510.04871*, 2025.

Georgios Kaissis, David Mildenberger, Juan Felipe Gomez, Martin J Menten, and Eleni Triantafillou. Step-resolved data attribution for looped transformers. *arXiv preprint arXiv:2602.10097*, 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.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*, 2019.

Haoyang Li, Yiming Li, Anxin Tian, Tianhao Tang, Zhanchao Xu, Xuejia Chen, Nicole Hu, Wei Dong, Qing Li, and Lei Chen. A survey on large language model acceleration based on kv cache management. *arXiv preprint arXiv:2412.19442*, 2024.

Junyan Li, Yang Zhang, Muhammad Yusuf Hassan, Talha Chafekar, Tianle Cai, Zhile Ren, Pengsheng Guo, Foroozan Karimzadeh, Chong Wang, and Chuang Gan. Commvq: Commutative vector quantization for kv cache compression. *arXiv preprint arXiv:2506.18879*, 2025.

William Merrill and Ashish Sabharwal. The expressive power of transformers with chain of thought. In *International Conference on Learning Representations*, volume 2024, pp. 7690–7706, 2024.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. Can a suit of armor conduct electricity? a new dataset for open book question answering. In *Proceedings of the 2018 conference on empirical methods in natural language processing*, pp. 2381–2391, 2018.

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.

Guilherme Penedo, Hynek Kydlíček, Anton Lozhkov, Margaret Mitchell, Colin 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*, 37:30811–30849, 2024.

Jacob Pfau, William Merrill, and Samuel R Bowman. Let's think dot by dot: Hidden computation in transformer language models. *arXiv preprint arXiv:2404.15758*, 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.

Ziran Qin, Yuchen Cao, Mingbao Lin, Wen Hu, Shixuan Fan, Ke Cheng, Weiyao Lin, and Jianguo Li. Cake: Cascading and adaptive kv cache eviction with layer preferences. In *International Conference on Learning Representations*, volume 2025, pp. 89945–89977, 2025.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Haseena Rahmath P, Vishal Srivastava, Kuldeep Chaurasia, Roberto G Pacheco, and Rodrigo S Couto. Early-exit deep neural network-a comprehensive survey. *ACM Computing Surveys*, 57(3): 1–37, 2024.

David Raposo, Sam Ritter, Blake Richards, Timothy Lillicrap, Peter Conway Humphreys, and Adam Santoro. Mixture-of-depths: Dynamically allocating compute in transformer-based language models. *arXiv preprint arXiv:2404.02258*, 2024.

Melissa Roemmele, Cosmin Adrian Bejan, and Andrew S Gordon. Choice of plausible alternatives: An evaluation of commonsense causal reasoning. In *AAAI spring symposium: logical formalizations of commonsense reasoning*, pp. 90–95, 2011.

Sara Sabour, Nicholas Frosst, and Geoffrey E Hinton. Dynamic routing between capsules. *Advances in neural information processing systems*, 30, 2017.

Maarten Sap, Hannah Rashkin, Derek Chen, Ronan Le Bras, and Yejin Choi. Social iqa: Commonsense reasoning about social interactions. In *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (EMNLP-IJCNLP)*, pp. 4463–4473, 2019.

Nikunj Saunshi, Stefani Karp, Shankar Krishnan, Sobhan Miryoosefi, Sashank J Reddi, and Sanjiv Kumar. On the inductive bias of stacking towards improving reasoning. *Advances in Neural Information Processing Systems*, 37:71437–71464, 2024.

Nikunj Saunshi, Nishanth Dikkala, Zhiyuan Li, Sanjiv Kumar, and Sashank J Reddi. Reasoning with latent thoughts: On the power of looped transformers. In *International Conference on Learning Representations*, volume 2025, pp. 14855–14881, 2025.

Luohe Shi, Hongyi Zhang, Yao Yao, Zuchao Li, and Hai Zhao. Keep the cost down: A review on methods to optimize llm's kv-cache consumption. *arXiv preprint arXiv:2407.18003*, 2024.

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.

Songhao Wu, Ang Lv, Xun Zhang, Guojun Yin, Wei Lin, Rui Yan, et al. Polarquant: Leveraging polar transformation for key cache quantization and decoding acceleration. *Advances in Neural Information Processing Systems*, 38:50584–50607, 2026.

Kevin Xu and Issei Sato. On expressive power of looped transformers: Theoretical analysis and enhancement via timestep encoding. *arXiv preprint arXiv:2410.01405*, 2024.

Chenxu Yang, Qingyi Si, Yongjie Duan, Zheliang Zhu, Chenyu Zhu, Qiaowei Li, Minghui Chen, Zheng Lin, and Weipinng Wang. Dynamic early exit in reasoning models. In *International Conference on Learning Representations*, volume 2026, pp. 88170–88210, 2026.

Xinlei Yu, Zhangquan Chen, Yongbo He, Tianyu Fu, Cheng Yang, Chengming Xu, Yue Ma, Xiaobin Hu, Zhe Cao, Jie Xu, et al. The latent space: Foundation, evolution, mechanism, ability, and outlook. *arXiv preprint arXiv:2604.02029*, 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.

Zihao Zeng, Yibo Miao, Hongcheng Gao, Hao Zhang, and Zhijie Deng. Adamoe: Token-adaptive routing with null experts for mixture-of-experts language models. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 6223–6235, 2024.

Rui-Jie Zhu, Zixuan Wang, Kai Hua, Tianyu Zhang, Ziniu Li, Haoran Que, Boyi Wei, Zixin Wen, Fan Yin, He Xing, et al. Scaling latent reasoning via looped language models. *arXiv preprint arXiv:2510.25741*, 2025.

A Implementation Details

Hardware and framework. All models are trained on 8×A100 (80 GB) GPUs using the open-source NanoGPT training stack as a reference implementation.

Data and tokens. Unless otherwise specified, we run each experiment for 250,000 optimizer steps with a global batch size of 48 sequences and block size 2048 (context length). This corresponds to approximately 25B training tokens in total.

Optimization. We use AdamW with weight decay 2×10^{-1} , cosine learning-rate decay (per NanoGPT), peak learning rate $lr = 6 \times 10^{-4}$, minimum learning rate $min_lr = 6 \times 10^{-5}$, and 4,000 warmup steps, which we found important for stability. Similar to observations in (Geiping et al., 2026) , we occasionally observe training instabilities at depth; warmup and cosine decay mitigate these in practice. Unless noted, other optimizer and training defaults follow NanoGPT.

Model hyperparameters. Following (Saunshi et al., 2025; Jeddi et al., 2026), we use hidden size d = 2048 and nheads = 32 for all configurations. The feed-forward dimension is d_{ff} = 5120 with a standard two-layer GELU MLP. All normalizations are RMSNorm. We use learned positional embeddings added to token embeddings (NanoGPT default).

T-LoopFormer Conditioning. We use two embedding modules, one for normalized time $t \in [0, 1]$ and one for step size $\Delta t \in (0, 1]$. Each maps a scalar input to a d-dimensional conditioning vector via (i) fixed sinusoidal Fourier features (width $D_f = 256$, max period 10000), followed by (ii) a 2-layer MLP with hidden size d and SiLU activation:

$$\phi(\tau) = \mathrm{MLP}\left(\left[\cos(\tau\omega_1), \sin(\tau\omega_1), \ldots, \cos(\tau\omega_{D_f/2}), \sin(\tau\omega_{D_f/2})\right]\right) \in \mathbb{R}^d$$

where $\omega_k = \exp\left(-\frac{k-1}{D_f/2} \log 10{,}000\right)$ for $k = 1, \ldots, D_f/2$. Given per-token cumulative time $t_{\tau,i}$ and step size $\Delta t_{\tau,i}$ for each active token τ at loop i, we compute $e_{t_{\tau,i}} = \phi(t_{\tau,i})$ and $e_{\Delta t_{\tau,i}} = \phi(\Delta t_{\tau,i})$, and sum them to obtain a token-specific conditioning signal $c_{\tau,i} = e_{t_{\tau,i}} + e_{\Delta t_{\tau,i}} \in \mathbb{R}^d$. Conditioning is applied inside each T-LoopFormer block via an AdaLN-style modulator: a small MLP takes $c_{\tau,i}$ and outputs $4d$ parameters per token, which we split into $(\alpha_{\mathrm{msa},\tau,i}, \alpha_{\mathrm{mlp},\tau,i}, \gamma_{\mathrm{msa},\tau,i}, \gamma_{\mathrm{mlp},\tau,i})$. We use RMSNorm (with no learned affinity) before MHSA and FFN, and apply token-wise multiplicative scaling and residual gating as

$$x \leftarrow x + \alpha_{\mathrm{msa},\tau,i} \odot \mathrm{MHSA}\big(\mathrm{RMSNorm}(x) \odot (1 + \gamma_{\mathrm{msa},\tau,i})\big),$$

$$x \leftarrow x + \alpha_{\mathrm{mlp},\tau,i} \odot \mathrm{FFN}\big(\mathrm{RMSNorm}(x) \odot (1 + \gamma_{\mathrm{mlp},\tau,i})\big).$$

The modulator—a SiLU followed by a linear layer of output size $4d$—shares weights across tokens, is applied token-wise rather than broadcast over the sequence, and is zero-initialized (weights and bias). This guarantees that the initial behavior equals that of the unmodulated backbone and that conditioning is learned stably.

B The difference between gather−scatter during training and inference.

While the gather–scatter scheme is employed in both training and inference, its role and mechanics differ between the two phases. These differences arise from the distinct objectives of each phase: training must maintain a differentiable computation graph, whereas inference must maintain a consistent KV cache for decoding.

Purpose of the Scatter. Training: The scatter serves as an integral part of the differentiable computation graph. After the shared block processes the compact tensor $\tilde{H}^{(i)}$, scattering the results back to their original positions in H accomplishes three things simultaneously: (i) the next loop can gather its own active set $\mathcal{A}_{i+1}$, which generally differs from $\mathcal{A}_i$, from a full-length tensor whose row indices remain the original sequence positions; (ii) each token's state resides at a fixed coordinate indexed by its position, enabling correct alignment with the loss; and (iii) gradients flow back

precisely along the indexed assignment to the corresponding rows of the compact tensor. In short, the scatter in training exists primarily to keep the computation graph consistent and the gradients well-defined.

Inference: No gradients are involved. The scatter instead serves cache management: it updates H with the latest states and, crucially, carries the newly computed keys and values to their designated slots in the loop-wise KV cache. Writing at the correct original positions is essential because future tokens will attend to these entries, and the causal alignment of the cache depends on each entry residing at its true position.

Address Translation via $\rho^{(i)}$. **Training:** The gather indices $\mathcal{A}_i$ themselves serve as the addresses for the scatter, and no additional bookkeeping is required: a row at compact index j is written back to position $\mathcal{A}_i[j]$.

Inference: Since the KV caches are indexed by original sequence positions rather than compact indices, the gather-time mapping $\rho^{(i)} = \mathrm{sort}(\mathcal{A}_i)$ must be captured and propagated through the shared block to the cache-writing routine. The newly computed K/V entries are written into the loop-i cache at their original positions $\rho^{(i)}$, not at their compact row indices. This positional bookkeeping is an inference-specific burden that training does not incur.

Reading the Cache: the Validity Mask $\mu^{(i)}$. **Training:** No cache reads occur, and no masking against stale entries is needed — inactive tokens are simply not present in the gathered tensor.

Inference: At loop i, attention queries only the loop-i cache, whose slots for tokens with $m_\tau < i$ are empty (zero-valued). Without masking, the softmax will assign non-negligible probability mass to these zero slots. The validity indicator $\mu^{(i)}_{t'} = \mathbb{1}[\, m_{t'} \geq i \,]$ (plus a block-causal mask among currently gathered tokens during batched prefill) is therefore mandatory at inference and has no training-time counterpart.

Degeneration during Single-Token Decoding. **Training:** Every forward pass processes the full sequence, so gather–scatter always operates on a multi-token tensor and never degenerates.

Inference: During autoregressive decoding, only one token is processed at a time. Gathering a single row is an identity operation, and scattering it back is equivalent to writing in place; the scheme therefore degenerates to a pure early exit: the recursion simply terminates once the current token leaves the active set. Cache updates reduce to ordinary tail concatenation ($\oplus_t$), and the address translation $\rho^{(i)}$ collapses to the identity.

In essence, the training-time scatter exists to preserve the integrity of the computation graph — coordinates, alignment, and gradients — while the inference-time scatter exists to preserve the integrity of the KV cache. The former guarantees that learning is correct; the latter guarantees that generation is correct.

C Routing Design Considerations

A natural concern regarding the proposed routing mechanism is the : *non-differentiability* — the hard thresholding and discrete index selection involved in gather–scatter seem to provide no gradient path to the router. We address the concern below.

Non-Differentiability. Although the resulting computation graph is discrete (selected tokens are gathered via non-differentiable indexing), the router receives gradients through complementary paths. *First*, under token-choice routing the selection rule $m_\tau = \min\{i : p^{(i)}_\tau \geq \frac{1}{2}\}$ is derived from the continue-probability $\bar{p}^{(i)}_\tau = \sum_{d>i} p^{(d)}_\tau$, a *differentiable* function of the router logits: gradient signal flows into the softmax distribution at every loop, even though the binary mask itself is discrete. *Second*, for tokens that *are* selected, the task loss $\mathcal{L}$ backpropagates through the shared block into the gathered representation and, via the gather indices, reaches the hidden states on which the router conditions and the router weight matrix itself can additionally be updated with a straight-through estimator $\mathrm{STE}[m_\tau] = m_\tau + \sigma(s_\tau) - \mathrm{sg}(\sigma(s_\tau))$ when task-loss gradients through the routing decision are desired, where s_τ is the "continue probs" at loop-i of token τ.

D Discussion For Performance Degeneration

Since the number of loops for each token is determined by its initial hidden state (for operational and hardware efficiency—specifically, to enable static tensor compaction via gather–scatter and avoid sequential halting checks at each loop iteration), when the loop count dictated by the token's initial hidden state exceeds the model's maximum loop limit, the decoding results will degenerate. Moreover, because the loop count for each token is not recomputed during the looping stage, the token's understanding of the *context* may be constrained, leading to degraded reasoning performance (a potential limitation of T-LoopFormer).