Efficient Transformer Parameter Reuse via Zero-Token Mechanism

Guanghao Li[1,2*], **Wenhao Jiang**[3†], **Li Shen**[4], **Ming Tang**[2], **Chun Yuan**[1†]
[1]Tsinghua Shenzhen International Graduate School, Tsinghua University
[2]Southern University of Science and Technology
[3]Guangdong Laboratory of Artificial Intelligence and Digital Economy (SZ)
[4]Shenzhen Campus of Sun Yat-sen University

Abstract

Resource constraints often limit the parameter capacity of Large Language Models (LLMs), thereby hindering their performance. Although existing approaches leverage parameter sharing to reuse a fixed set of parameters within constrained budgets, they typically require each layer to fulfill multiple roles over a fixed number of iterations. This design compromises both efficiency and adaptability. In this work, we propose the *Zero Token Transformer (ZTT)*, which employs a *head-tail decoupled parameter cycling* strategy. Specifically, we decouple the first (head) and last (tail) layers from the parameter cycling process, enabling iterative refinement solely within the intermediate layers. Furthermore, we introduce a *Zero-Token Mechanism*, wherein a virtual token with a trainable key and a zero-valued vector functions as a standard token. The resulting attention scores not only reflect the computational significance of each layer but also facilitate dynamic early exiting, thereby preserving overall model accuracy. Our approach achieves superior performance under strict parameter constraints, substantially reduces computational overhead via early exits, and can be seamlessly integrated into the fine-tuning of existing pre-trained models, improving both efficiency and adaptability.

1 Introduction

In recent years, it has been widely recognized that the performance of Large Language Models (LLMs) improves as the number of parameters increases (Rae et al., 2021; Rosenfeld et al., 2019). As a result, scaling up parameter counts has become a prevalent strategy for boosting model performance (Leviathan et al., 2023; Xu et al., 2024; Pope et al., 2023). However, this approach is often impractical for users with limited computational resources. A fundamental challenge, therefore, is to achieve enhanced performance under a *fixed parameter budget* (Zhou et al., 2024).

To this end, various model compression techniques have been proposed, such as quantization (Lin et al., 2025; Liu et al., 2023), pruning (Ma et al., 2023; Sun et al., 2023), and distillation (Latif et al., 2023; Shum et al., 2024), to reduce large models to more compact versions. Concurrently, another line of research explores how to exploit additional computation within a fixed parameter budget to enable deeper or more iterative reasoning (Dehghani et al., 2018; Lan et al., 2019). A commonly adopted strategy is *parameter sharing*, where model layers reuse the same parameters across multiple computational steps, a practice sometimes referred to as "parameter cycling." Instead of assigning distinct parameters to each layer, a compact parameter set is recurrently applied, thereby reducing memory usage and potentially enhancing the model's reasoning depth.

Despite its promise, parameter cycling introduces three key challenges: (1) **Which** parameters should be shared across iterative cycles? (2) **How** can these shared parameters be effectively managed to prevent functional conflicts and performance degradation? (3) **When** should the model determine that further reasoning is unnecessary, thus conserving computational resources without prematurely halting essential inference?

Existing works address these questions only partially. For instance, Solar (Kim et al., 2023) reuses parameters from intermediate layers (*which* parameters), while the Relaxed Recursive Transformer (Bae et al., 2024) investigates *how* to manage recurring layers via LoRA (Hu et al., 2022). Palbert (Balagansky and Gavrilov, 2022), which integrates PonderNet (Banino et al., 2021) with ALBERT, explores *when* to terminate inference through a dynamic pondering mechanism. Nonetheless, none of these approaches offers a uni-

*E-mail: ligh24@mails.tsinghua.edu.cn
†Corresponding authors (cswhjiang@gmail.com, yuanc@sz.tsinghua.edu.cn)

Findings of the Association for Computational Linguistics: ACL 2026, pages 14498–14515
July 2-7, 2026 ©2026 Association for Computational Linguistics

fied framework that simultaneously addresses all three aspects—*which* parameters to cycle, *how* to apply them, and *when* to conclude reasoning.

In this paper, we propose a *Zero Token Transformer (ZTT)* that systematically addresses these three challenges. Specifically:

- **Head-Tail Decoupled Cyclic Architecture.** To address *which* parameters to share, we decouple the first (head) and last (tail) layers from parameter sharing, as their specialized roles—encoding raw inputs and mapping representations to outputs—differ substantially from those of the intermediate layers. Only the intermediate layers are reused recurrently in a cyclic manner, thereby enhancing efficiency while maintaining the essential input and output transformations.
- **Zero-Token Mechanism.** To tackle *how* to manage shared parameters effectively, we introduce a novel Zero-Token Mechanism. Each intermediate layer retrieves a Zero-Token (comprising a trainable key and a zero-valued representation) from a Zero-Token Pool and processes it together with the regular tokens. The attention scores assigned to the Zero-Token guide layer-specific computations, enabling the model to decide the extent of "reuse vs. new reasoning" at each cyclic iteration.
- **Adaptive Exit Mechanism.** To determine *when* to terminate the cyclic process, we adopt an early-exit mechanism guided by the attention scores to the Zero-Token. When the attention toward the Zero-Token exceeds a predefined threshold, the model infers that further computations are unlikely to yield additional benefits and exits accordingly—achieving a balance between computational efficiency and accuracy preservation.

We demonstrate that our approach improves performance in both training from scratch and fine-tuning existing large language models across a range of reasoning tasks, underscoring its practical applicability in real-world deployments.

2 Related Work

Parameter sharing has been extensively explored in early deep learning architectures, such as Convolutional Neural Networks (CNNs) (Eigen et al., 2013; Savarese and Maire, 2019) and Recurrent Neural Networks (RNNs) (Graves, 2016; Sherstinsky, 2020), as an effective means to reduce model complexity while maintaining performance. The Universal Transformer (Dehghani et al., 2018) extended this concept to the Transformer architecture, demonstrating that cyclic parameter reuse across layers can significantly enhance computational efficiency. Subsequently, various studies have investigated *which* components of the Transformer should be shared. Some explore parameter reuse within individual layers (Dabre and Fujita, 2019), the coupling of encoder and decoder components (Milbauer et al., 2023; Xia et al., 2019), or the integration of partial expert networks (Liu et al., 2024). Others aim to optimize *how* parameter sharing is organized, such as by stacking parameters in specific sequences (Takase and Kiyono, 2021) or employing factorized embeddings (Lan et al., 2019) to improve model performance. A crucial dimension of parameter cycling involves determining *when* to repeat computations. Techniques such as ACT (Chowdhury and Caragea, 2024; Graves, 2016; Csordás et al., 2024; Tan et al., 2023; Storaï et al., 2025) and PonderNet (Banino et al., 2021; Balagansky and Gavrilov, 2022) enable adaptive recursion by dynamically selecting the number of computation cycles, while equilibrium formulations (e.g., DEQ-style models) relate depth to iterative fixed-point computation (Bai et al., 2019, 2020). However, most of these approaches focus on training from scratch rather than fine-tuning large pre-trained models, limiting practical deployment.

Recent work has explored parameter cycling in pre-trained large language models. For example, Solar (Kim et al., 2023) reuses middle layers of Llama (Touvron et al., 2023), addressing *which* layers to cycle. Relaxed Recursive Transformers (Bae et al., 2024) use LoRA (Hu et al., 2022) to alleviate degradation caused by repeated reuse. However, *when* to halt recurrent computation remains underexplored. Although Relaxed Recursive Transformers support variable cycle counts, they still rely on fixed computation paths rather than truly dynamic mechanisms. Meanwhile, early-exit methods (Chen et al., 2023; Pan et al., 2024; Schuster et al., 2022) mainly focus on non-recurrent models, leaving open how many cycles a recurrent model should perform.

In addition, our Zero Token differs from prior virtual or soft tokens (Lester et al., 2021): instead of being appended to the input sequence as learnable content, it is introduced inside each attention layer as a control entry with a trainable key and a fixed zero-valued value for cyclic reuse and adaptive halting. More broadly, our approach addresses *what* to cycle, *how* to manage recurrent parameters, and *when* to terminate reasoning. It is applicable

to both training from scratch and fine-tuning pretrained LLMs, offering a practical and efficient solution for improving model performance under stringent parameter constraints.

3 Zero Token Transformer

In this section, we introduce our *Zero-Token Transformer (ZTT)*, with the overall framework depicted in Fig. 1. The ZTT employs a *Head-Tail Decoupled Cyclic Architecture* and incorporates a novel *Zero-Token Mechanism* in conjunction with a gating strategy. In the subsequent subsections, we detail how the proposed *head-tail decoupling* addresses the aforementioned challenges, and how the integration of the *Zero Token* and the *gating mechanism* enhances performance and enables dynamic early exiting.

3.1 Head-Tail Decoupled Cyclic Architecture

The Transformer is a sequence-to-sequence architecture based entirely on attention mechanisms. Given an input sequence $X = [x_1, x_2, \ldots, x_n]$, each token is embedded and combined with a positional encoding to form Z^0. The layer-wise computation is defined as

$$Z^l = f^{(l)}\left(Z^{l-1}\right), \tag{1}$$

where $f^{(l)}(\cdot)$ denotes the operation of the l-th layer, and l ranges from 1 to L. Each layer consists of a *multi-head self-attention* module followed by a position-wise feedforward network, with residual connections and layer normalization applied to both sub-layers.

Recent analyses (Sun et al., 2025) suggest that *intermediate* Transformer layers often exhibit functionally similar representations, whereas the *head* (first) and *tail* (last) layers specialize in tasks such as raw feature encoding and output mapping. We observe a consistent pattern in a pretrained GPT-2 model via layer-wise representation similarity (Appendix D.8, Fig. 3). In contrast, modifying these boundary layers often has an outsized impact on overall performance. Hence, we preserve the original first and last layers as *fixed* (non-cyclic) and let only the middle layers reuse parameters across N cycles, as shown in Figure 1(right).

Concretely, if the Transformer has L layers, the forward propagation within each cycle is formulated as

$$Z^{l,n} = f^{(l,n)}\left(Z^{l-1,n}\right), \tag{2}$$

where $l \in \{2, 3, \ldots, L-1\}$ and $n \in \{1, \ldots, N\}$. After completing the $(L-1)$-th layer of cycle n, its output is passed to the next cycle as

$$Z^{2,n+1} = Z^{L-1,n}, \quad n < N. \tag{3}$$

The first and last layers ($l = 1$ and $l = L$) remain fixed and are excluded from parameter cycling. This preserves the distinct roles of the head and tail layers, mitigating potential conflicts.

3.2 Zero-Token Mechanism

Motivation and design. Even with head–tail decoupling, intermediate layers in a cyclic architecture may still process representations redundantly. When the current representation is already of high quality, repeated refinement may overwrite useful features or introduce inconsistencies. To mitigate this issue, we introduce a virtual "Zero Token" into each attention layer, serving as a control signal that indicates whether the model should further refine the current representation or preserve it with minimal update. This design is necessary because head–tail decoupling alone determines which layers are reused, but does not provide an explicit mechanism for controlling how reused layers behave across cycles. For the l-th layer in cycle n, the virtual token consists of a **trainable key**, an **all-zero value**, and **no query** component. In implementation, the Zero Token is placed at the front so that all other tokens can attend to it.

Role of the Zero Token. Each cycle retrieves its corresponding Zero Token, whose trainable key can elicit either strong or weak attention from the queries Q^l. When the model attends heavily to this zero-valued token, the resulting output remains close to the previous representation, since the zero-valued branch contributes little additional update. In this way, the Zero Token provides an explicit no-op option under cyclic reuse, allowing each layer to softly choose between further refinement and preserving the current representation. This mechanism is particularly important in repeated parameter reuse, where redundant refinement may otherwise accumulate and introduce functional interference across cycles. We further provide a preliminary theoretical discussion of this near-identity behavior from a fixed-point viewpoint in Appendix C.

3.3 Token-wise Gating Mechanism

While the Zero Token's attention score serves as the primary indicator for determining whether to terminate additional cycles, we introduce a lightweight

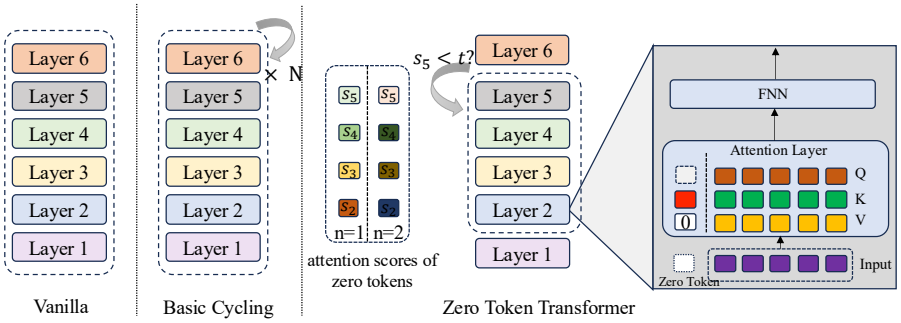


Figure 1: **Left:** A 6-layer vanilla transformer without cycling. **Center:** A transformer equipped with a basic two-cycle mechanism. **Right:** A Zero Token Transformer ($N = 2$), where the first and last layers are excluded from the cycling process. Each participating layer introduces an additional virtual Zero Token. The rightmost illustration demonstrates the incorporation of the Zero Token. Taking the second layer as an example: the virtual Zero Token is prepended to the sequence by aligning its key with those of the original tokens at the beginning, and an all-zero value is inserted at the front of the value sequence.

gating mechanism around the feed-forward network (FFN) to enable more fine-grained computational control. Even when an early exit is not triggered, certain cycles may only require partial FFN computation. To this end, we modify the FFN by incorporating a gating mechanism as follows:

$$Z^l = Z_h^l + \text{FFN}\left(\text{LN}\left(Z_h^l\right)\right) \cdot \text{gate}\left(\text{LN}(Z_h^l)\right), \tag{4}$$

where Z_h^l denotes the output from the self-attention sublayer, and $\text{gate}(\cdot)$ is a gating function applied to each input token embedding, implemented as an MLP with a sigmoid activation. When the gate value approaches 1, the full FFN transformation is applied; when it approaches 0, the FFN is largely bypassed. We observed that the gating value often correlates with the attention assigned to the Zero Token: high attention to the Zero Token indicates that further refinement is unnecessary, leading to a corresponding reduction in the gate value.

3.4 Adaptive Exit Mechanism

Exit criterion. The attention score of the Zero Token (§3.2) remains the primary criterion for initiating early exit. Once this score surpasses a predefined threshold (e.g., 0.9), the iterative cycles are terminated, and the final representation is produced. The gating function smooths the transition across cycles by reducing redundant FFN computations before the final exit trigger.

Discussion. We do not apply gating to the *attention module* itself, as attention is responsible for integrating token representations, including that of the Zero Token. By gating only the FFN, we enable partial omission of the more computationally intensive transformations, without disrupting the signaling function of the Zero Token. Thus, the attention score of the Zero Token determines *when* to exit entirely, while the gating mechanism modulates *how much* computation is applied prior to that decision point.

3.5 Training and Inference Strategy

Training with early exit. To enable adaptive exiting during inference, we supervise the model at every cycle. Concretely, after each iteration n, we apply a lightweight prediction head (shared across cycles; $< 1\%$ extra parameters) to the intermediate state and compute a task-specific loss (e.g., cross-entropy for classification or span loss for QA). This encourages meaningful predictions at each step and yields informative signals for early exit at test time. While it is possible to train with a fully dynamic number of cycles per example, optimizing step counts can easily lead to degenerate halting without careful regularization (Geiping et al., 2025). For robustness and simplicity, we therefore train with a fixed maximum number of cycles and supervise all intermediate states, following common early-exit practice (Zhou et al., 2020; Kaya et al., 2019).

Inference. At inference time, we run the cyclic block iteratively and monitor the Zero-Token attention; once the threshold is met, we stop and return the current prediction. We discard intermediate heads and evaluate the prediction head only once at the selected exit cycle; hence, intermediate supervision incurs negligible inference-time overhead.

4 Experiments and Analysis

4.1 Experimental Results on Reasoning Tasks

We present our experimental setup and results to demonstrate the effectiveness of the proposed *Zero-*

Token Transformer approach under a fixed parameter budget on reasoning tasks. We evaluate both the *training from scratch* and *fine-tuning* scenarios, employing a decoder-only Transformer architecture.

4.1.1 Experimental Setup

Datasets and Metrics. We consider seven representative datasets covering reasoning skills, including PIQA (Bisk et al., 2020), ARC Challenge and ARC Easy (Clark et al., 2018), LAMBADA (Paperno et al., 2016), HellaSwag (Zellers et al., 2019) , SQuAD (Rajpurkar et al., 2016) and CoQA (Reddy et al., 2019). We report accuracy for PIQA, ARC Challenge and ARC Easy, LAMBADA (Paperno et al., 2016), HellaSwag, and F1 score for SQuAD and CoQA.

We consider two main training settings. **(1) Training from Scratch:** All models are pre-trained on the C4 English subset[1] (Raffel et al., 2020) using a causal next-token prediction objective over 10B tokens. **(2) Fine-Tuning Pretrained Models:** We further fine-tune widely used pretrained models such as GPT-2 and OPT (Zhang et al., 2022), demonstrating that our approach can be readily applied to large pre-trained models with minimal modification.

Our architecture is based on GPT-2 (Radford et al., 2019), with each layer constrained to approximately 10M parameters. The total number of layers is $L = 6$. To enforce a fixed computational budget, we define a total of 6 "network computation cycles," where each layer in a standard configuration (i.e., without parameter sharing) is counted as one cycle. We compare several approaches, all based on decoder-only Transformers. One variant, trained with early exit, incorporates classification heads at each cycle to enable intermediate predictions. These models are denoted by the suffix **-EE**. The variant that performs inference with adaptive exit is indicated by appending the suffix **-AE**.

- **Vanilla Transformer (VT):** A standard Transformer with L distinct layers. The total computation cost is effectively L cycles.
- **Basic Cycling Transformer (BCT):** The model has L layers but shares parameters across layers by cycling them N times. This results in a total computation cost of $L \times N$.
- **Head-Tail Decoupled Cycling Transformer (HTDCT):** Instead of cycling *all* L layers, only the *intermediate* layers are reused N times, while the first and last layers remain distinct. This structure aims to preserve the special functions of the input and output layers. We also consider an early exit variant of this scheme.
- **Zero-Token Transformer (ZTT):** Our method, which only cycles the intermediate layers and introduces a *Zero Token* in each attention layer. This token provides a learnable *key* while carrying zero-value vectors, enabling the model to distinguish between different cycles and facilitate *adaptive computation*. We also include an early exit variant that uses the learned *Zero Attention* signals to decide when to stop.

4.1.2 Experimental Results on Training from Scratch

Table 1 reports results for models trained from scratch. The main observations are:

Effect of using fewer layers. By comparing VT and VT-Small, we can see that reducing both the parameter count and computational budget from $L = 6$ to $L = 3$ ("Vanilla-Small") leads to a significant accuracy drop (e.g., from 25.52% to 23.29%). This suggests that simply using fewer layers cannot maintain adequate performance without cycling.

Effect of cycling. To mitigate the performance gap, BC reuses a 3-layer stack twice (for a total of 6 cycles), partially recovering performance to 24.50%. This confirms that increased "computational depth" via cycling can help, though it still lags behind the original 6-layer Transformer. Introducing early exit (BCT-EE) in BCT leads to a slight accuracy drop (23.88%), suggesting that training additional intermediate heads can sometimes introduce optimization trade-offs.

Effect of head-tail decoupling. By fixing the first and last layers while only cycling the intermediate ones, we achieve 25.64% (no early exit) and 24.88% (early exit), surpassing Basic Cycling. This underscores the importance of preserving specialized head and tail layers.

Effect of Zero-Token mechanism. Building on head-tail separation, our proposed Zero Token mechanism further boosts accuracy to 26.18% without early exit, and 25.61% with early exit. Notably, ZTT consistently outperforms both BC and HT across tasks, demonstrating that the Zero-Token mechanism effectively guides each cycle's computation and alleviates functional conflicts in shared parameters.

Overall, these results confirm the benefit of our ZTT approach in balancing parameter efficiency

[1] https://huggingface.co/datasets/allenai/c4

Models	Size	Layers	Looped Layers	Loop Count	PQ	ARC-c	ARC-e	LD	HS	SQuAD	CoQA	Avg	Cycle Avg	Δ
VT-small	60.65M	3	0	-	61.32	17.83	37.84	13.8	26.75	2.05	3.43	23.29	23.29	-
VT	81.9M	6	0	-	64.15	19.28	40.32	19.08	27.04	3.6	5.2	25.52	25.52	+2.23
BCT	60.65M	3	3	-	63.44	18.6	38.8	16.32	26.77	2.17	5.43	24.50	24.50	+1.21
BCT-EE	60.65M	3	3	1	61.7	17.83	37.88	13.24	26.93	1.97	3.67	23.32	23.88	+0.59
				2	62.73	18.17	39.94	16.32	26.90	1.92	5.06	24.43		
HTDCT	60.65M	3	1	4	63.17	19.2	40.15	16.52	26.73	2.19	11.5	25.64	25.64	+2.35
HTDCT-EE	60.65M	3	1	1	63.55	18.77	39.65	16.24	26.67	3.33	5.16	24.77	24.88	+1.59
				2	64.04	17.92	40.87	13.33	26.55	3.16	6.28	24.59		
				3	62.95	18.26	39.73	15.33	26.68	2.86	8.39	24.89		
				4	63.71	18.69	39.86	12.78	26.79	3.23	11.91	25.28		
ZTT	61.77M	3	1	4	62.51	19.3	40.87	17.94	26.98	2.93	12.75	26.18	26.18	+2.89
ZTT-EE	61.77M	3	1	1	62.95	17.66	38.76	16.71	26.76	4.34	6.59	24.82	25.16	+1.87
				2	63.44	18.26	41.16	16.63	26.81	3.44	6.34	25.15		
				3	64.25	17.58	40.61	14.28	26.85	3.58	8.75	25.13		
				4	63.55	18.00	41.04	13.59	26.93	3.48	12.07	25.52		
ZTT-AE	61.77M	3	1	3.32	63.22	18.17	41.46	15.27	26.87	4.62	12.51	26.01	26.01	+2.72

Table 1: Evaluation results of different models pre-trained on the C4 dataset and fine-tuned on the test datasets. For -EE variants, Cycle Avg is the unweighted average of **Avg** over fixed Loop Count settings; for AE and other methods, it is identical to **Avg**. Δ is relative to VT-small.

and modeling capacity. Even when the total parameter budget and computation cycles are constrained, introducing Zero Tokens with a head-tail separation strategy yields superior accuracy.

4.1.3 Experimental Results on Fine-tuning Pretrained Models

We further validate our method on large, pretrained models: GPT-2 (Radford et al., 2019) and OPT (Zhang et al., 2022). Table 2 presents the performance of various cycling strategies after fine-tuning on the same downstream tasks.

Across all model scales, cycling-based methods consistently outperform the Vanilla baseline. Basic Cycling transformer (BCT) yields notable accuracy improvements over vanilla transformer, highlighting the effectiveness of reusing parameters through repeated computation. However, when early exit is employed (BCT-EE), performance may occasionally decline slightly due to the additional overhead associated with optimizing intermediate outputs.

Among all approaches, the Zero-Token Transformer (ZTT) achieves the highest accuracy, surpassing both BCT and VT. These improvements suggest that incorporating a Zero Token during fine-tuning enables the model to effectively leverage repeated reasoning within a fixed parameter budget. Moreover, the early-exit variant, Zero-Token Transformer trained with early exit (ZTT-EE), maintains accuracy comparable to the full ZTT while substantially reducing computational costs. This demonstrates that adaptive inference can be effectively scaled to large pretrained models.

Notably, as model sizes increase—such as GPT-2 Large with 811M parameters—both ZTT and ZTT-EE continue to deliver substantial accuracy gains while preserving parameter efficiency. These findings underscore the broad applicability and scalability of our proposed Zero-Token approach, establishing it as a robust fine-tuning strategy for large-scale language models.

4.2 Ablation Study and Analysis

We analyze the contribution of key components; additional results are deferred to Appendix D.

4.2.1 The effect of N

Figure 2a shows the perplexity on WikiText-2 under an equivalent total computational cost (*number of layers* $\times$ *number of cycles*). Given the same computational budget, our Head–Tail Decoupled Cyclic Transformer (HTDCT) achieves lower perplexity than the naive "all-layer cycling" design, confirming that preserving specialized boundary layers helps mitigate the performance gap. Furthermore, a 1-layer BCT executed over 12 cycles yields lower perplexity than a 1-layer vanilla Transformer (i.e., $N = 1$), though it still underperforms a standard 12-layer Transformer with an equivalent computational budget. Similarly, attaching classification heads after each cycle to enable early exiting ("Early exit" in Figure 2a) further degrades performance, indicating that the requirement for intermediate outputs introduces additional burden on the cyclic layers. Figure 2b reports the intermediate perplexity of the early-exit variants, showing that for a fixed model with early exit, allowing more cycles of computation consistently reduces

Models		Size	All Layers	Looped Layers	Loop Count	PQ	ARC-c	ARC-e	LD	HS	SQuAD	CoQA	Avg	Cycle Avg	Δ
GPT-2	VT	124.44M	12	0	–	62.89	19.03	43.81	25.97	28.92	12.59	39.37	33.23	33.23	-
	BCT	124.44M	12	12	2	65.23	20.65	43.35	29.34	28.26	12.17	39.26	34.04	34.04	+0.81
	BCT-EE	124.44M	12	12	1	64.69	20.22	43.81	30.22	28.39	11.55	39.86	34.11	33.87	+0.64
		124.44M	12	12	2	64.47	20.31	43.30	27.93	28.20	10.97	40.22	33.63		
	ZTT	128.91M	12	10	2	65.23	20.05	44.61	28.88	28.17	13.90	39.33	34.31	34.31	+1.08
	ZTT-EE	128.91M	12	10	1	65.51	20.65	44.23	26.45	28.46	13.28	40.47	34.15	**34.34**	+1.11
		128.91M	12	10	2	64.69	20.22	44.78	29.42	28.33	13.55	40.68	34.52		
GPT-2 Medium	VT	354.82M	24	0	–	67.63	21.50	49.07	37.69	33.31	14.87	45.05	38.45	38.45	-
	BCT	354.82M	24	24	2	69.64	21.84	49.96	39.59	32.27	15.55	46.69	39.36	39.36	+0.91
	BCT-EE	354.82M	24	24	1	69.64	23.12	50.80	39.83	32.34	13.03	46.72	39.35	38.80	+0.35
		354.82M	24	24	2	68.10	22.53	48.58	38.00	32.21	12.25	46.08	38.25		
	ZTT	370.67M	24	22	2	69.53	22.96	49.49	39.83	32.46	13.73	48.81	39.54	39.54	+1.09
	ZTT-EE	370.67M	24	22	1	69.15	22.44	50.84	39.20	32.32	14.72	48.27	39.56	**39.35**	+0.90
		370.67M	24	22	2	68.08	22.50	50.24	38.77	32.14	15.88	46.39	39.14		
GPT-2 Large	VT	774.03M	36	0	–	70.35	21.67	49.07	40.40	36.40	23.40	49.87	41.59	41.59	-
	BCT	774.03M	36	36	2	70.62	25.91	51.52	43.99	35.78	23.24	46.35	42.49	42.49	+0.90
	BCT-EE	774.03M	36	36	1	71.38	25.51	50.80	43.92	35.78	20.54	50.19	42.59	42.30	+0.71
		774.03M	36	36	2	71.16	25.43	50.55	41.96	35.98	21.74	47.20	42.00		
	ZTT	811.12M	36	34	2	71.04	25.57	51.35	43.95	35.93	20.68	49.90	42.63	42.63	+1.04
	ZTT-EE	811.12M	36	34	1	70.84	25.89	51.47	43.79	35.50	21.84	49.68	42.72	**43.25**	+1.66
		811.12M	36	34	2	70.48	25.74	51.33	43.65	35.20	31.28	48.83	43.79		
GPT-2 xl	VT	1.56B	48	0	–	70.84	25.00	58.29	44.61	40.04	23.40	51.32	44.78	44.78	-
	BCT	1.56B	48	48	2	71.67	27.05	61.22	47.35	38.82	16.24	50.56	44.70	44.70	-0.08
	BCT-EE	1.56B	48	48	1	72.14	26.28	60.94	47.20	38.51	13.54	52.69	44.47	44.35	-0.43
		1.56B	48	48	2	71.78	26.37	60.89	47.29	37.52	14.74	51.08	44.23		
	ZTT	1.63B	48	46	2	72.11	28.16	61.30	48.13	38.52	13.68	52.06	44.85	44.85	+0.07
	ZTT-EE	1.63B	48	46	1	72.03	28.24	60.23	48.13	38.95	14.84	51.48	44.84	**45.44**	+0.66
		1.63B	48	46	2	71.71	28.33	60.06	48.65	38.67	24.28	50.63	46.04		
OPT 6.7B	VT	6.66B	32	0	–	76.28	30.63	65.61	67.63	50.47	35.88	63.92	55.77	55.77	-
	BCT	6.66B	32	32	2	76.68	32.59	68.40	68.56	49.72	33.60	62.42	55.99	55.99	+0.22
	BCT-EE	6.66B	32	32	1	76.10	32.93	67.00	67.12	49.35	34.85	62.91	55.75	55.86	+0.09
		6.66B	32	32	2	76.35	32.25	67.71	67.52	49.40	36.21	62.34	55.96		
	ZTT	6.99B	32	30	2	77.33	33.44	68.77	68.69	49.89	44.69	64.81	58.23	58.23	+2.46
	ZTT-EE	6.99B	32	30	1	77.20	33.36	68.45	68.21	49.75	40.17	64.90	57.43	**57.46**	+1.69
		6.99B	32	30	2	76.95	32.84	68.31	67.65	49.85	42.58	64.23	57.48		

Table 2: Results of fine-tuning pretrained models on downstream tasks. For -EE variants, Cycle Avg is the unweighted average of Avg over fixed Loop Count settings; for AE and other methods, it is identical to Avg. Δ is computed w.r.t. the VT Cycle Avg within the same backbone.

Loop Count	1	2	3	4
Zero Attention	0.21	0.47	0.54	0.65
Gate Value	0.55	0.32	0.20	0.15
PPL	37.02	34.58	34.03	33.99

Table 3: Perplexity (PPL) and Zero Attention metrics of the Zero Token Transformer (early exit) method on the C4 dataset with different loop counts.

P	0.2	0.5	0.7	1.0
Avg_Acc	24.19	26.01	25.81	25.52
Avg_Loop	1.63	3.45	3.78	4.00

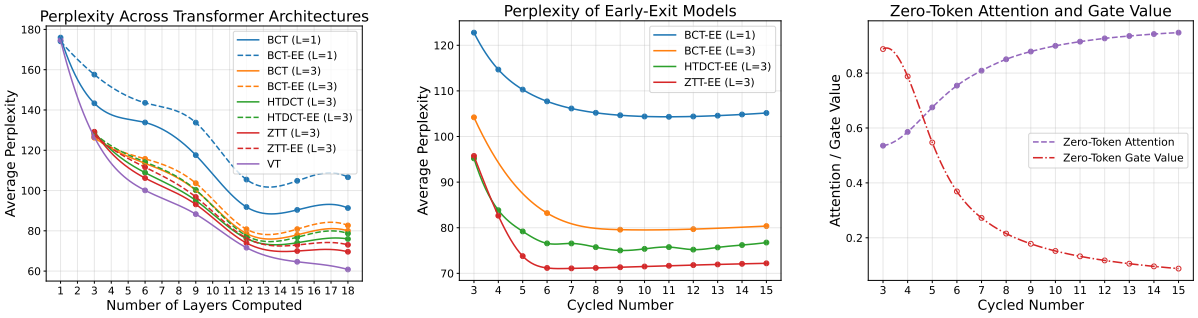
Table 4: Selection of the Zero-Token attention threshold P used as the early-exit criterion, reporting average accuracy and cycles. Per-task results are provided in Appendix D.1.

perplexity and thus improves performance. In Figure 2c, we plot the average attention score to the Zero Token across different cycles. Higher attention reflects the model's tendency to "opt out" of deeper recalculation, whereas lower attention corresponds to committing more computation to refine the prediction.

4.2.2 Adaptive Exit with Zero Attention Score

We next investigate whether *Zero Attention*—defined as the average attention across all heads assigned to the Zero Token—can serve as a stopping criterion for adaptive inference. Specifically, once the Zero Attention score surpasses a predefined threshold P, the representation is deemed sufficiently refined, and further cycles are terminated. The experiments are conducted on the C4 dataset. In Table 3, we report perplexity (PPL), Zero Attention, and Gate Value across different cycle counts. As the loop depth increases, Zero Attention progressively rises, while the Gate Value in the feed-forward network declines, suggesting diminishing returns from continued computation.

Table 4 further investigates the impact of varying thresholds P on both model accuracy and the average number of computation cycles. A lower threshold ($P = 0.2$) induces early termination,



(a) Perplexity vs. compute budget. (b) Early-exit intermediate perplexity. (c) Zero-Token attention and gate value.

Figure 2: Comparison of model performance under equal computational complexity. **(a)** Effect of varying computational complexity, where $L = 1$ denotes the original model with a single layer, and increased complexity corresponds to repeated model calls. **(b)** Intermediate results of different models under the "early exit" condition when the total computational complexity is fixed at 15 (cycles $\times$ layers). **(c)** Zero-Token attention (dashed) and gate value (dash-dotted) as a function of the cycled number, explicitly showing their inverse relationship.

Gate	ZT	PQ	ARC-c	ARC-e	LD	HS	SQuAD	CoQA	Avg
✓	✓	63.55	17.88	40.39	15.30	26.84	3.71	8.43	**25.16**
✓	×	62.95	18.04	39.62	15.77	26.75	3.60	7.84	24.93
×	✓	63.25	18.12	40.07	15.22	26.70	4.44	6.88	24.95
×	×	63.56	18.41	40.03	14.42	26.67	3.14	7.93	24.88

Table 5: Results of the ablation study, where Gate represents the gating unit in the FFN, and ZT stands for Zero Token.

Model	FLOPs (G)	Params (M)	Latency (ms)	Memory (MB)
VT_small	15.32	60.65	1.58	415.65
VT	20.76	81.91	2.54	570.03
BCT/HTDCT	20.76	60.65	2.53	487.42
ZTT	21.11	61.77	2.82	511.79
ZTT-AE	19.50	61.77	2.50	483.37

Table 6: Comparison of computational cost, latency, and memory footprint.

substantially reducing computational cost but at the expense of some performance degradation. In contrast, a moderate threshold ($P = 0.5$) strikes a favorable balance, achieving 26.01% accuracy with an average of only 3.45 cycles—comparable to or even exceeding the full 4-cycle baseline. These results demonstrate that Zero Attention can effectively guide dynamic computation, enabling models to adaptively allocate reasoning cycles while sustaining high performance. This represents a promising approach for efficient inference in resource-constrained environments.

To pinpoint the contribution of each component, we conduct an ablation study by selectively removing the Zero Token (ZT) or the Gate in the FFN layer. The results, summarized in Table 5, highlight the individual and combined effects of these components. The full model (ZT + Gate) achieves the highest average accuracy of 25.16%, demonstrating the complementary benefits of these two mechanisms. When the Gate is removed, the model experiences a slight performance drop, indicating that the gating mechanism refines the computation flow within the feed-forward network. Similarly, removing only the Zero Token yields a comparable drop, suggesting that the Zero Token is important for dynamic cycle awareness. When both components are disabled, the model reaches its lowest performance, confirming that these mechanisms jointly contribute to reasoning efficiency and predictive accuracy. These results further clarify the role of the Zero Token in our framework. While head–tail decoupling determines which layers are reused, the Zero Token provides a control signal for deciding whether further refinement is beneficial: its zero-valued branch offers a no-op option under cyclic reuse, while its attention score naturally supports adaptive exit. Together with the behavior of Zero Attention across cycles, these findings show that the Zero Token contributes not only to accuracy but also to compute-efficient inference.

4.3 Computation Comparisons

Table 6 presents a comparison of FLOPs, latency, and memory usage across different models. Although ZTT introduces a slight computational and memory overhead compared to BCT/HTDCT, ZTT-AE demonstrates notable efficiency improvements. Specifically, it reduces the FLOPs from 21.11G to 19.50G and lowers both latency and memory usage once the model learns to exit early.

5 Conclusion

We presented the *Zero-Token Transformer (ZTT)*, a parameter-sharing framework that jointly addresses the core questions of *which* layers to reuse, *how* to

manage shared parameters, and *when* to stop iterating. By decoupling the head and tail layers from the cyclic process and introducing a learnable Zero Token within each attention block, ZTT enables *adaptive computation*, dynamically adjusting reasoning depth based on model confidence. Experiments demonstrate that ZTT is effective for both *training from scratch* and *fine-tuning pre-trained models*, consistently improving performance under a limited parameter budget via parameter reuse (with only minor overhead). The Zero Token not only supports parameter-efficient reasoning but also provides a simple criterion for early exiting, reducing redundant computation while preserving accuracy. These results highlight the promise of *dynamic parameter-sharing strategies* for large-scale language models, especially in resource-constrained settings. We believe that further exploration of zero-token prompting, gating mechanisms, and cyclic reasoning will inspire increasingly efficient and adaptive Transformer designs.

Limitations

In the current design of the Zero-Token Transformer, a fixed number of refinement iterations is employed during the cyclic computation phase to ensure stable and efficient training. While this choice provides reliable convergence, it may limit the model's flexibility in adapting to inputs of varying complexity. Future work could explore reinforcement learning strategies to dynamically determine the number of iterations, enabling the model to adjust its computational effort based on confidence signals or task demands. This would further enhance both the efficiency and adaptability of the architecture under constrained resource settings.

Acknowledgments

This work is supported by the National Key R&D Program of China (2022YFB4701400/4701402), SSTIC Grant (KJZD20230923115106012, KJZD20230923114916032, GJHZ20240218113604008).

References

Stephen H. Bach, Victor Sanh, Zheng-Xin Yong, Albert Webson, Colin Raffel, Nihal V. Nayak, Abheesht Sharma, Taewoon Kim, M Saiful Bari, Thibault Fevry, Zaid Alyafeai, Manan Dey, Andrea Santilli, Zhiqing Sun, Srulik Ben-David, Canwen Xu, Gunjan Chhablani, Han Wang, Jason Alan Fries, and 8 others. 2022. Promptsource: An integrated development environment and repository for natural language prompts. *Preprint*, arXiv:2202.01279.

Sangmin Bae, Adam Fisch, Hrayr Harutyunyan, Ziwei Ji, Seungyeon Kim, and Tal Schuster. 2024. Relaxed recursive transformers: Effective parameter sharing with layer-wise lora. *arXiv preprint arXiv:2410.20672*.

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

Shaojie Bai, Vladlen Koltun, and J Zico Kolter. 2020. Multiscale deep equilibrium models. *Advances in neural information processing systems*, 33:5238–5250.

Nikita Balagansky and Daniil Gavrilov. 2022. Palbert: Teaching albert to ponder. In *Advances in Neural Information Processing Systems*, volume 35, pages 14002–14012. Curran Associates, Inc.

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

Yonatan Bisk, Rowan Zellers, Jianfeng Gao, Yejin Choi, and 1 others. 2020. Piqa: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 7432–7439.

Yanxi Chen, Xuchen Pan, Yaliang Li, Bolin Ding, and Jingren Zhou. 2023. Ee-llm: Large-scale training and inference of early-exit large language models with 3d parallelism. *arXiv preprint arXiv:2312.04916*.

Jishnu Ray Chowdhury and Cornelia Caragea. 2024. Recurrent transformers with dynamic halt. *arXiv e-prints*, pages arXiv–2402.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training verifiers to solve math word problems. *Preprint*, arXiv:2110.14168.

Róbert Csordás, Kazuki Irie, Jürgen Schmidhuber, Christopher Potts, and Christopher D Manning. 2024. Moeut: Mixture-of-experts universal transformers. *arXiv preprint arXiv:2405.16039*.

Raj Dabre and Atsushi Fujita. 2019. Recurrent stacking of layers for compact neural machine translation models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 6292–6299.

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

David Eigen, Jason Rolfe, Rob Fergus, and Yann LeCun. 2013. Understanding deep architectures using a recursive convolutional network. *arXiv preprint arXiv:1312.1847*.

Mostafa Elhoushi, Akshat Shrivastava, Diana Liskovich, Basil Hosmer, Bram Wasti, Liangzhen Lai, Anas Mahmoud, Bilge Acun, Saurabh Agarwal, Ahmed Roman, and 1 others. 2024. Layerskip: Enabling early exit inference and self-speculative decoding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 12622–12642.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, and 5 others. 2024. The language model evaluation harness.

Jonas Geiping, Sean McLeish, Neel Jain, John Kirchenbauer, Siddharth Singh, Brian R Bartoldson, Bhavya Kailkhura, Abhinav Bhatele, and Tom Goldstein. 2025. Scaling up test-time compute with latent reasoning: A recurrent depth approach. *arXiv preprint arXiv:2502.05171*.

Alex Graves. 2016. Adaptive computation time for recurrent neural networks. *arXiv preprint arXiv:1603.08983*.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, and 1 others. 2022. Lora: Low-rank adaptation of large language models. *ICLR*, 1(2):3.

Weizhe Hua, Zihang Dai, Hanxiao Liu, and Quoc Le. 2022. Transformer quality in linear time. In *International conference on machine learning*, pages 9099–9117. PMLR.

Yigitcan Kaya, Sanghyun Hong, and Tudor Dumitras. 2019. Shallow-deep networks: Understanding and mitigating network overthinking. In *International conference on machine learning*, pages 3301–3310. PMLR.

Dahyun Kim, Chanjun Park, Sanghoon Kim, Wonsung Lee, Wonho Song, Yunsu Kim, Hyeonwoo Kim, Yungi Kim, Hyeonju Lee, Jihoo Kim, and 1 others. 2023. Solar 10.7 b: Scaling large language models with simple yet effective depth up-scaling. *arXiv preprint arXiv:2312.15166*.

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

Ehsan Latif, Luyang Fang, Ping Ma, and Xiaoming Zhai. 2023. Knowledge distillation of llm for automatic scoring of science education assessments. *arXiv preprint arXiv:2312.15842*.

Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. The power of scale for parameter-efficient prompt tuning. In *Proceedings of the 2021 conference on empirical methods in natural language processing*, pages 3045–3059.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. 2023. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning*, pages 19274–19286. PMLR.

Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Guangxuan Xiao, and Song Han. 2025. Awq: Activation-aware weight quantization for on-device llm compression and acceleration. *GetMobile: Mobile Computing and Communications*, 28(4):12–17.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, and 1 others. 2024. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*.

Zechun Liu, Barlas Oguz, Changsheng Zhao, Ernie Chang, Pierre Stock, Yashar Mehdad, Yangyang Shi, Raghuraman Krishnamoorthi, and Vikas Chandra. 2023. Llm-qat: Data-free quantization aware training for large language models. *arXiv preprint arXiv:2305.17888*.

Xinyin Ma, Gongfan Fang, and Xinchao Wang. 2023. Llm-pruner: On the structural pruning of large language models. *Advances in neural information processing systems*, 36:21702–21720.

Jeremiah Milbauer, Annie Louis, Mohammad Javad Hosseini, Alex Fabrikant, Donald Metzler, and Tal Schuster. 2023. Lait: Efficient multi-segment encoding in transformers with layer-adjustable interaction. *arXiv preprint arXiv:2305.19585*.

Xuchen Pan, Yanxi Chen, Yaliang Li, Bolin Ding, and Jingren Zhou. 2024. Ee-tuning: An economical yet scalable solution for tuning early-exit large language models. *arXiv preprint arXiv:2402.00518*.

Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Quan Ngoc Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. 2016. The lambada dataset: Word prediction requiring a broad discourse context. *arXiv preprint arXiv:1606.06031*.

Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, Jonathan Heek, Kefan Xiao, Shivani Agrawal, and Jeff Dean. 2023. Efficiently scaling transformer inference. *Proceedings of Machine Learning and Systems*, 5:606–624.

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

Jack W Rae, Sebastian Borgeaud, Trevor Cai, Katie Millican, Jordan Hoffmann, Francis Song, John Aslanides, Sarah Henderson, Roman Ring, Susannah Young, and 1 others. 2021. Scaling language models: Methods, analysis & insights from training gopher. *arXiv preprint arXiv:2112.11446*.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*.

Siva Reddy, Danqi Chen, and Christopher D Manning. 2019. Coqa: A conversational question answering challenge. *Transactions of the Association for Computational Linguistics*, 7:249–266.

Jonathan S Rosenfeld, Amir Rosenfeld, Yonatan Belinkov, and Nir Shavit. 2019. A constructive prediction of the generalization error across scales. *arXiv preprint arXiv:1909.12673*.

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

Pedro Savarese and Michael Maire. 2019. Learning implicitly recurrent cnns through parameter sharing. *arXiv preprint arXiv:1902.09701*.

Tal Schuster, Adam Fisch, Jai Gupta, Mostafa Dehghani, Dara Bahri, Vinh Tran, Yi Tay, and Donald Metzler. 2022. Confident adaptive language modeling. *Advances in Neural Information Processing Systems*, 35:17456–17472.

Noam Shazeer. 2019. Fast transformer decoding: One write-head is all you need. *arXiv preprint arXiv:1911.02150*.

Alex Sherstinsky. 2020. Fundamentals of recurrent neural network (rnn) and long short-term memory (lstm) network. *Physica D: Nonlinear Phenomena*, 404:132306.

KaShun Shum, Minrui Xu, Jianshu Zhang, Zixin Chen, Shizhe Diao, Hanze Dong, Jipeng Zhang, and Muhammad Omer Raza. 2024. First: Teach a reliable large language model through efficient trustworthy distillation. *arXiv preprint arXiv:2408.12168*.

Romain Storaï, Jaeseong Lee, and Seung-won Hwang. 2025. Smarter, not harder: Training-free adaptive computation for transformers. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 8147–8155.

Mingjie Sun, Zhuang Liu, Anna Bair, and J Zico Kolter. 2023. A simple and effective pruning approach for large language models. *arXiv preprint arXiv:2306.11695*.

Qi Sun, Marc Pickett, Aakash Kumar Nain, and Llion Jones. 2025. Transformer layers as painters. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 25219–25227.

Sho Takase and Shun Kiyono. 2021. Lessons on parameter sharing across layers in transformers. *arXiv preprint arXiv:2104.06022*.

Shawn Tan, Yikang Shen, Zhenfang Chen, Aaron Courville, and Chuang Gan. 2023. Sparse universal transformer. *arXiv preprint arXiv:2310.07096*.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, and 1 others. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Yingce Xia, Tianyu He, Xu Tan, Fei Tian, Di He, and Tao Qin. 2019. Tied transformers: Neural machine translation with shared encoder and decoder. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 5466–5473.

Mengwei Xu, Dongqi Cai, Yaozong Wu, Xiang Li, and Shangguang Wang. 2024. Fwdllm: Efficient federated finetuning of large language models with perturbed inferences. In *2024 USENIX Annual Technical Conference (USENIX ATC 24)*, pages 579–596.

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

Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, and 1 others. 2022. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*.

Wangchunshu Zhou, Canwen Xu, Tao Ge, Julian McAuley, Ke Xu, and Furu Wei. 2020. Bert loses patience: Fast and robust inference with early exit. *Advances in Neural Information Processing Systems*, 33:18330–18341.

Zixuan Zhou, Xuefei Ning, Ke Hong, Tianyu Fu, Jiaming Xu, Shiyao Li, Yuming Lou, Luning Wang, Zhihang Yuan, Xiuhong Li, and 1 others. 2024. A survey on efficient inference for large language models. *arXiv preprint arXiv:2404.14294*.

A Experimental Setup

A.1 Evaluation Details

To assess the effectiveness of our proposed method, we evaluate models on a diverse set of well-established NLP benchmarks. These benchmarks span four key reasoning tasks: commonsense physical reasoning, multiple-choice question answering, long-term context recall, and natural language inference. We follow the experimental setup and implementation of prior work (Kim et al., 2023; Bae et al., 2024; Saunshi et al., 2025) and employ the Language Model Evaluation Harness (Gao et al., 2024) for consistent evaluation. We report the metrics produced by the harness, primarily accuracy (ACC) and F1 score. In addition, representative settings in Table 1 were repeated three times and showed a small standard deviation of around 0.15, indicating that run-to-run variance is not the main source of the observed differences.

A.1.1 Reasoning: PIQA

Dataset: The *Physical Interaction Question Answering (PIQA)* dataset (Bisk et al., 2020) evaluates a model's ability to reason about everyday physical interactions. It consists of multiple-choice questions that require an understanding of how objects and tools function in real-world scenarios.

Task Objective: Given a short natural language query, the model must select the most plausible solution from two candidate answers. This task assesses the model's ability to infer practical physical knowledge beyond simple memorization.

Evaluation Metric: Accuracy (ACC), measuring the proportion of correctly predicted answers.

A.1.2 Multiple-Choice Question Answering: ARC Challenge and ARC Easy

Dataset: The *AI2 Reasoning Challenge (ARC)* (Clark et al., 2018) is a standardized multiple-choice QA benchmark designed to evaluate a model's ability to answer science-related questions. It consists of two subsets:

- ARC Challenge: A more difficult subset requiring complex reasoning and deeper knowledge retrieval.
- ARC Easy: A simpler subset containing factual questions that can often be answered with surface-level understanding.

Task Objective: The model is provided with a science-related question and four answer choices, from which it must select the correct one. The dataset requires a combination of commonsense reasoning, logical inference, and scientific knowledge to achieve high performance.

Evaluation Metric: Accuracy (ACC), computed as the percentage of correctly answered questions.

A.1.3 Long-Term Context Recall: LAMBADA

Dataset: The *LAMBADA* dataset (Paperno et al., 2016) is specifically designed to assess a model's capability for long-range context comprehension. Unlike standard language modeling tasks, LAMBADA requires a model to retain and process information over an extended passage to predict a crucial missing word.

Task Objective: Given a long contextual passage, the model must predict the final missing word of the last sentence. The difficulty arises from the fact that the target word is nearly impossible to guess without understanding the full passage.

Evaluation Metric: Accuracy (ACC), where a prediction is considered correct if the entire target word matches the ground truth exactly.

A.1.4 Natural Language Inference: HellaSwag

Dataset: The *HellaSwag* dataset (Zellers et al., 2019) is an advanced benchmark designed to evaluate commonsense inference and story continuation. It builds on the SWAG dataset by incorporating adversarial filtering, making it more challenging for models to rely on surface-level heuristics.

Task Objective: Given an incomplete story or event description , the model must select the most logical next step from four possible continuations. This requires strong contextual understanding and the ability to anticipate real-world event progressions .

Evaluation Metric: Accuracy (ACC), measuring how often the model correctly predicts the most plausible continuation.

A.1.5 Reading Comprehension: SQuAD

Dataset: The *Stanford Question Answering Dataset (SQuAD)* (Rajpurkar et al., 2016) is a widely-used benchmark for evaluating extractive question answering systems. It contains questions posed by crowdworkers on a set of Wikipedia articles, with answers being spans from the corresponding paragraphs.

Task Objective: Given a context paragraph and a question, the model must extract a text span from the context that correctly answers the question.

Evaluation Metric: We report the F1 score, which measures the token-level overlap between the predicted span and the ground truth answer. This provides a balanced metric that accounts for partial matches and is widely used in QA benchmarks.

A.1.6 Conversational QA: CoQA

Dataset: The *Conversational Question Answering (CoQA)* dataset (Reddy et al., 2019) assesses a model's ability to engage in multi-turn question answering where the questions are conversational and often depend on dialogue history.

Task Objective: Given a passage and a sequence of previous question-answer pairs, the model must answer the current question in natural language. The task requires context tracking and conversational understanding.

Evaluation Metric: We follow the official CoQA evaluation metric, which measures word-level F1 score between the predicted and reference answers.

A.2 Training and Fine-Tuning Settings

In this section, we describe the training settings for both pre-training from scratch and fine-tuning of pre-trained models. The fine-tuning stage is required for all models before final evaluation, while models trained from scratch undergo both pre-training and fine-tuning. The fine-tuning hyperparameters are kept consistent across both settings.

A.2.1 Pre-Training from Scratch

For models trained from scratch, we first conduct pre-training on the *C4 English dataset* (Raffel et al., 2020). The pre-training process follows these configurations:

Pre-Training Protocol

- Dataset: The *C4* English subset.
- Computing Resources: We utilize an `A800 GPU cluster` for training.
- Batch Size per GPU: 80, with gradient accumulation to maintain a global batch size of 256.
- Training Steps: The model is trained for a total of 10B tokens.
- Optimizer: AdamW with a weight decay of 0.01.
- Learning Rate: A linear warmup is applied for the first 1% of total steps, followed by a cosine decay schedule.
- Precision: Training is performed in half-precision (FP16) to optimize memory efficiency.

After pre-training, the models proceed to the fine-tuning stage before being evaluated on downstream tasks.

A.2.2 Fine-Tuning Settings

Before evaluating on the test datasets, we fine-tune our models using the corresponding training sets. For pre-trained models, only fine-tuning is performed, while models trained from scratch undergo both pre-training and fine-tuning. The fine-tuning process is conducted under the same computational settings as pre-training.

Fine-Tuning Protocol

- Fine-Tuning Epochs: Each dataset is fine-tuned for 3 epochs.
- Batch Size per GPU: 20, with gradient accumulation ensuring an effective batch size of 80.
- Optimizer: AdamW with a 0.01 weight decay.
- Learning Rate: The default Hugging Face `Trainer` API learning rate is used.
- Prompt Engineering: We utilize prompt templates from `promptsource` (Bach et al., 2022) to better adapt models to the task format.
- Computing Resources: The same `A800 GPU cluster` is used as in pre-training.
- Training Framework: Fine-tuning is implemented with Hugging Face's `Trainer API`.

Dataset-Specific Fine-Tuning Details Fine-tuning is performed on the following datasets before model evaluation. Table 7 summarizes the fine-tuning settings and the subset sizes used in our experiments.

A.3 Early-Exit Training Settings

To ensure effective intermediate predictions when early-exit mechanisms are applied, we implement additional training for intermediate classifier heads. This helps maintain meaningful intermediate outputs, preventing degradation in performance due to premature exits.

Dataset	Epochs	Training Size	Validation Size
PIQA	3	16,000	1,838
ARC Challenge	3	1,119	1,172
ARC Easy	3	2,251	2,376
LAMBADA	3	4,869	4,869
HellaSwag	3	39,905	10,042
SQuAD	3	130,319	11,873
CoQA	3	7,199	500

Table 7: Fine-tuning settings for each dataset, including the number of training epochs and dataset sizes.

A.3.1 Classifier Placement

- Simple Cycling: The classification head is placed only at the final output layer.
- Head-Tail Separation: The classification head is placed at both the final layer and the last shared layer before cycling begins.

A.3.2 Training Strategy for Early-Exit Models

To optimize models for early exits, we introduce additional supervision at intermediate layers. Instead of relying solely on the final output, we ensure that multiple exit points are trained effectively.

- Intermediate Supervision: The model is trained to produce meaningful predictions at designated early-exit points.
- Exit Point Optimization: Models with multiple cycling blocks undergo training to align their intermediate outputs with final predictions, improving robustness across different exit depths.
- Gradual Refinement: The early-exit heads are optimized using the same fine-tuning data, ensuring consistency across all prediction layers.

By integrating these early-exit classifiers and fine-tuning them separately, we ensure that models can gracefully exit at earlier layers without sacrificing predictive accuracy. This design allows our method to maintain efficiency while preserving strong performance across different computational budgets.

B Comparisons on Language Modeling Task

We study the language modeling ability of our model on the WikiText-2 dataset[2]. We compare our method with PonderNet (Banino et al., 2021), Relaxed Recursive Transformers (RRT) (Bae et al., 2024), SOLAR (Kim et al., 2023), and CALM (Schuster et al., 2022). We implement PonderNet by incorporating its early-exit mechanism into a ∼7M-parameter Transformer trained on WikiText-2; however, it yields significantly worse perplexity, possibly due to conflicts between the KL-divergence objective and standard language modeling. For a fair comparison, RRT and SOLAR follow their original training protocols and are trained from scratch rather than fine-tuned on WikiText-2; meanwhile, we adapt CALM to our cyclic, parameter-sharing setting for a direct comparison under the same backbone and compute budget.

Model	Perplexity
PonderNet (Banino et al., 2021)	232.46
RRT (Bae et al., 2024)	176.52
SOLAR (Kim et al., 2023)	180.56
CLAM (Schuster et al., 2022)	177.42
BCT	178.87
HTDCT	174.23
ZTT	**172.40**

Table 8: The perplexities from competing models on the WikiText-2 dataset.

As shown in Table 8, our model achieves the lowest perplexity on the WikiText-2 dataset and consistently performs competitively across multiple downstream benchmarks. These results suggest that ZTT consistently outperforms or matches existing adaptive-depth and recurrent baselines, demonstrating the effectiveness of its cyclic parameter sharing strategy in enhancing both language modeling and task generalization.

C Simple Theoretical Analysis

We provide a preliminary theoretical perspective by framing our cyclic updates as a fixed-point iteration, following the viewpoint of Deep Equilibrium Models (Bai et al., 2019). Let $\mathbf{H}^{(n)}$ denote the hidden representation at the n-th iteration, and let $f(\cdot)$ be the update function. Each iteration is then defined as:

$$\mathbf{H}^{(n+1)} = f\left(\mathbf{H}^{(n)}\right) \tag{5}$$

As the difference $\|\mathbf{H}^{(n+1)} - \mathbf{H}^{(n)}\|$ approaches zero, the sequence converges to a fixed point $\mathbf{H}^*$. In conventional Transformers, learning an exact identity mapping (i.e., a "no-op" step) can be difficult,

[2] https://huggingface.co/datasets/Salesforce/wikitext

p		PQ	ARC-c	ARC-e	LD	HS	SQuAD	CoQA	Avg
0.2	Loop	1.82	1.61	1.42	1.72	1.33	1.83	1.72	1.63
	Acc	63.02	17.99	39.04	15.26	26.79	2.17	5.06	24.19
0.5	Loop	2.9	3.23	3.13	3.56	3.77	3.82	3.78	3.45
	Acc	63.22	18.17	41.46	15.27	26.87	3.58	10.23	26.01
0.7	Loop	3.87	3.42	3.47	3.93	3.99	3.92	3.89	3.78
	Acc	64.15	18.34	41.25	14.67	26.97	3.33	11.91	25.81
1	Loop	4	4	4	4	4	4	4	4
	Acc	63.55	18	41.04	13.59	26.93	3.48	12.07	25.52

Table 9: More detailed results on adaptive reasoning loop counts.

Models	Size	All Layers	Looped Layers	Loop Count	PQ	ARC-c	ARC-e	LD	HS	SQuAD	CoQA	Avg	Cycle_Avg
Wo Gate	60.66M	3	1	1	63.06	18.69	40.03	16.13	26.64	3.37	4.82	24.67	24.95
				2	63.60	17.83	40.36	15.34	26.76	4.28	5.00	24.73	
				3	63.11	18.03	39.86	15.71	26.84	6.75	7.32	25.37	
				4	63.22	17.92	40.03	13.70	26.65	3.37	10.39	25.04	
Wo ZT	61.76M	3	1	1	62.72	17.75	39.81	15.37	26.81	3.16	5.85	24.49	24.93
				2	62.92	18.22	40.32	16.50	26.76	3.33	6.57	24.94	
				3	63.02	17.77	40.19	16.22	26.88	3.58	7.21	24.98	
				4	63.12	18.43	38.17	14.98	26.55	4.36	11.75	25.33	

Table 10: More detailed ablation study results, including each early-exit outcome.

as each sub-layer inevitably introduces numerical changes.

Our *Zero Token* mechanism enables a more direct path to approximate identity mappings. When the model attends predominantly to the Zero Token (whose value is fixed at zero), the resulting output update becomes negligible, effectively indicating that no further refinement is required. Unlike standard residual connections—which are hard-coded architectural paths—this behavior is softly learned via attention, allowing the model to discover near-identity transformations once it has reached a stable state, and is complementary to other inference-efficiency techniques such as efficient decoding and layer skipping (Shazeer, 2019; Hua et al., 2022; Elhoushi et al., 2024).

D More Experimental Results

To further analyze the effectiveness of our method, we present additional adaptive reasoning loop and ablation experiments in Table 9 and Table 10.

D.1 Analysis of Adaptive Reasoning Loops

Table 9 presents results on our adaptive reasoning loop mechanism, where the model dynamically determines the number of iterations based on the Zero Attention threshold (P).

Key observations:

- Low threshold ($P = 0.2$) results in early exits (1.63 cycles) but slightly lower accuracy (24.19%).
- Balanced performance at $P = 0.5$: The model averages 3.45 cycles and reaches 26.01% accuracy, achieving strong efficiency gains.
- Higher thresholds ($P = 0.7$) lead to more computation (3.78 cycles) and slight accuracy gains (25.81%), but with diminishing returns.
- Full computation ($P = 1$) does not significantly outperform adaptive strategies, confirming that early exit can maintain performance.

These results demonstrate that adaptive early exit strategies reduce computation while maintaining accuracy, with $P = 0.5$ being the most efficient trade-off.

D.2 Ablation Study on Zero Token and Gating Mechanism

Table 10 provides a detailed breakdown of our ablation study, evaluating the impact of the Zero Token (ZT) mechanism and the gating unit (Gate) in the feed-forward network (FFN). The results highlight the individual and combined contributions of these components.

Key findings:

- The full model (ZT + Gate) achieves the highest accuracy (25.16%), demonstrating that both components are essential.

- Removing the Gate leads to a slight performance drop (24.95%), suggesting that gating helps refine reasoning.
- Removing the Zero Token reduces accuracy further (24.93%), indicating its role in guiding iterative reasoning.
- The baseline model (without ZT and Gate) achieves the lowest accuracy (24.88%), confirming that both components contribute positively.

These results validate that both Zero Token and Gate are essential for maximizing model efficiency and reasoning quality.

D.3 Alternative Approaches: Iteration Embeddings and Input Recalls

Methods such as (Geiping et al., 2025) determine exit points by comparing representations across consecutive iterations, which can incur extra reasoning steps and computational cost—especially in deep models. Our method, by contrast, uses attention to the Zero Token as a soft termination signal, providing a lightweight and effective mechanism for early stopping.

Model	Perplexity
HTDCT + Cycle Pos Embedding	172.58
HTDCT + PLUG	175.14
BCT	178.87
HTDCT	174.23
ZTT	**172.40**

Table 11: Comparisons of competing models on WikiText-2.

D.4 Fixed Maximum Cycles

In our experiments, we train models with a fixed maximum loop count (e.g., $N = 5$). At inference time, however, we assess the model's robustness when the number of cycles is extended beyond this training horizon. We observe that ZT (Zero Token) generalizes well, maintaining stable performance by reusing the Zero Token signal from the final iteration. In contrast, BC (Baseline Cycling) tends to degrade due to overthinking, where repeated iterations introduce noise rather than refinement.

Table 12 illustrates this behavior. When extrapolating from 5 to 10 cycles, ZT continues to improve slightly or plateaus, while BC shows diminishing returns and even performance degradation.

D.5 Controlling for Parameter Count

For GPT-2 and OPT, the ZT design introduces additional trainable components—namely the *K vectors*, a *Zero Token Pool*, and *gating units*—compared to BCT/BCT-EE. To evaluate whether these extra parameters are the primary reason behind performance improvements, we created a control baseline: BC with matched parameters, achieved by integrating *prefix tuning* and *projection layers* to approximately match ZT's parameter count.

Table 13 reports the results. Even under comparable parameter budgets, ZT consistently outperforms BCT/BCT-EE, especially on *generative tasks* such as SQuAD and CoQA,. This suggests that ZTT's advantage arises not merely from extra capacity, but from its structural innovation and learned refinement dynamics.

D.6 Attribution of Gains (HTDCT vs. ZTT)

To isolate the contribution of each technique under the fine-tuning setting, we conducted additional experiments on GPT-2 and GPT-2-medium. We compared HTDCT and its early-exit variant HTDCT-EE under different loop counts and juxtaposed these with ZTT. Based on the results in Table 14 and their aggregated summaries in Table 15, we conclude that the primary performance improvement (approximately +1 Avg) arises from HTDCT, while ZTT provides a smaller but consistent additional gain of about +0.1 Avg, alleviates loop conflicts, and enables early-exit behavior. These findings suggest that HTDCT serves as the main driver of task-level accuracy gains, establishing a robust foundation for cyclic refinement, whereas ZTT complements it by improving convergence stability and inference adaptability. In practical scenarios, HTDCT should be prioritized when accuracy is the main concern, while the combination of HTDCT and ZTT is preferred when inference efficiency, stability, and robustness are also critical considerations.

D.7 Additional Evidence on LLaMA-2-7B

To further evaluate the generality of our method on a stronger backbone, we conduct additional experiments on LLaMA-2-7B. Across seven benchmarks (Table 16), ZTT consistently outperforms the corresponding cyclic baseline under a comparable parameter budget, and these gains remain stable for the early-exit variants. Notably, the improvements are more pronounced on this stronger backbone,

Model	1	2	3	4	5	6	7	8	9	10
BCT	206.61	193.58	189.88	188.21	187.34	186.89	186.69	188.54	188.64	188.81
ZTT	**169.45**	**167.82**	**164.61**	**163.18**	**162.39**	**161.89**	**161.56**	**161.53**	**161.52**	**161.54**

Table 12: Perplexity comparison when increasing the number of inference cycles beyond the training limit ($N = 5$).

Model	Params	PQ	ARC-C	ARC-E	LD	HS	SQuAD	CoQA	Avg	Cycle Avg
GPT2-XL	BCT (1.63B)	72.06	27.56	61.25	47.32	38.55	14.55	49.25	44.36	44.36
	BCT-EE (1.63B)	72.58	27.03	60.92	47.21	38.22	15.02	49.31	44.33	44.33
		71.32	26.96	61.12	47.30	38.05	16.62	49.02	44.34	
OPT-6.7B	BCT (6.99B)	77.05	33.01	68.52	67.99	49.81	36.88	63.28	56.65	56.65
	BCT-EE (6.99B)	76.93	32.45	67.82	67.88	49.69	37.21	62.96	56.42	56.26
		76.12	33.12	67.45	68.06	49.52	35.42	63.02	56.10	

Table 13: Evaluation results on different benchmarks.

Model	Method	Loop	PQ	ARC-C	ARC-E	LD	HS	SQuAD	CoQA	Avg	Cycle Avg
GPT-2	HTDCT	2	65.13	20.82	44.44	28.42	28.23	13.11	39.06	34.17	34.17
	HTDCT-EE	1	65.07	20.39	44.70	28.26	28.39	12.52	40.18	34.21	34.15
		2	64.53	20.11	44.23	28.33	28.28	12.33	40.84	34.09	
GPT-2-medium	HTDCT	2	69.42	22.35	49.20	39.76	32.33	14.87	48.16	39.44	39.44
	HTDCT-EE	1	69.64	22.44	50.17	39.28	32.14	14.32	47.88	39.41	39.24
		2	69.10	21.67	50.21	38.07	32.34	14.91	47.29	39.08	

Table 14: Attribution of Gains: HTDCT vs. ZTT (rebuttal experiments). "Avg" is the average over all listed tasks; "Cycle Avg" averages across loop counts for the corresponding method.

Model	VT	HTDCT	ZTT
GPT-2	33.23	34.17 (+0.94 vs. VT)	34.31 (+0.14 over HTDCT)
GPT-2-medium	38.45	39.44 (+0.99 vs. VT)	39.54 (+0.10 over HTDCT)

Table 15: Aggregated averages. HTDCT delivers the primary gain (≈+1 Avg) over VT; ZTT adds a consistent ≈+0.1 Avg on top.

which is consistent with our main observation that cyclic refinement benefits more from stronger base representations.

Beyond these standard reasoning benchmarks, we also examine whether the advantages of ZTT transfer to open-ended generation. Since LLaMA-2-7B provides a stronger foundation for generative evaluation, we further test ZTT on GSM8K (Cobbe et al., 2021), where the model must generate the final numeric answer for each math word problem. This setting complements our classification and extractive QA results by probing autoregressive decoding under non-trivial multi-step reasoning. As shown in Table 17, ZTT again yields clear improvements, including for its adaptive/early-exit variants, suggesting that the benefits of our method extend beyond discriminative tasks to sequence generation.

D.8 Layer-wise Representation Similarity of GPT-2

To better understand where parameter cycling is most appropriate, we analyze the layer-wise representation similarity of a pretrained GPT-2 model. Concretely, we run the model on a held-out corpus and record the hidden states after each Transformer block. For each layer l, we average the token representations over the corpus to obtain a single vector $\bar{\mathbf{h}}_l \in \mathbb{R}^d$. We then compute a cosine-similarity matrix

$$S_{ij} = \frac{\bar{\mathbf{h}}_i^\top \bar{\mathbf{h}}_j}{\|\bar{\mathbf{h}}_i\|_2 \, \|\bar{\mathbf{h}}_j\|_2},$$

where i, j index layers including the embedding (layer 0) and the final block.

Figure 3 visualizes this cosine-similarity matrix. We observe a clear block structure: (1) The **middle layers** form a highly homogeneous cluster with

Method	PQ	ARC-c	ARC-e	LD	HS	SQuAD	CoQA	Avg	Cycle Avg	Δ
VT	78.13	43.00	75.42	79.19	57.10	44.15	77.94	64.99	64.99	–
BCT	79.82	47.10	77.15	81.25	57.87	46.56	78.82	66.94	66.94	+1.95
BCT-EE	78.40 79.00	45.90 46.08	77.27 77.95	81.39 80.78	57.78 57.79	47.32 49.99	78.32 79.13	66.63 67.24	66.93	+1.94
ZTT	81.18	52.65	80.60	82.58	58.18	51.82	82.56	69.94	69.94	+4.95
ZTT-EE	80.14 80.96	51.28 50.85	79.92 80.22	82.17 82.54	57.89 58.08	49.82 52.32	80.36 81.01	68.80 69.43	69.11	+4.12

Table 16: Results on LLaMA-2-7B across seven benchmarks. Cycle Avg averages Avg over the early-exit settings of each EE variant.

Model	VT	BCT	BCT-EE	ZTT	ZTT-EE	ZTT-AE
GSM8K Acc.	29.57	33.66	33.42	36.46	35.93	36.61

Table 17: GSM8K results on LLaMA-2-7B.

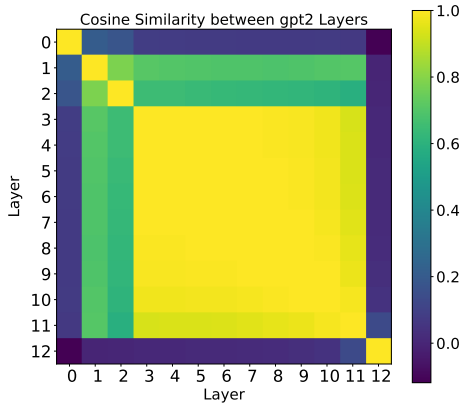


Figure 3: Cosine similarity between layers of the pre-trained GPT-2 model.

cosine similarity close to 1.0, indicating that these layers implement very similar transformations in the representation space. (2) In contrast, the **first few layers** (near the input side) and the **last few layers** (near the output side) are noticeably less similar to the others and to each other, suggesting that they are more specialized for input encoding and output decoding, respectively.

These observations support the design choice behind our HTDCT module. Since the middle layers behave in a highly redundant manner, they are good candidates for parameter cycling and depth extension under a fixed parameter budget. On the other hand, the input- and output-facing layers play distinct roles and therefore benefit from remaining unshared, preserving specialized processing at the head and tail of the network.

E Impact Statement

This work proposes a parameter-efficient approach for cyclic Transformers, aiming to enhance large language models under fixed resource constraints. We believe our contributions could lower the hardware and computational barriers to training and deploying high-capacity models, thereby democratizing advanced language technologies for broader communities. As with any method that enhances model efficiency, careful consideration is required to ensure responsible deployment. While increased accessibility can accelerate innovation, it also underscores the importance of robust safeguards against potential biases in training data or unintended misuse. Future work should focus on integrating stronger alignment techniques and ethical guidelines to mitigate these concerns. Overall, we anticipate that our technique will foster innovation in developing resource-friendly language models while encouraging responsible AI practices to maximize societal benefits.

F The Use of Large Language Models

Large language models (LLMs) were used exclusively for language polishing of this manuscript, including improving clarity, grammar, and minor rewording. They were not used for study design, data collection, annotation, analysis, experiment execution, or result generation. All edits suggested by LLMs were carefully reviewed and approved by the authors before inclusion. We did not upload proprietary evaluation data beyond the manuscript text, and the authors take full responsibility for the accuracy, originality, and integrity of the final content. The use of LLMs does not affect the reproducibility of the experiments or the validity of the reported findings.