Block Recursive Transformers for Structured Parameter Sharing

Josep Marques[1], Hyunyoung A. Cho[1], Youngeun Kim[2], Youjia Zhang[1], Aecheon Jung[1], Kyuhong Shim[1], Sungeun Hong[1†]

[1]Sungkyunkwan University
[2]Korea University

Abstract

Recursive Transformers depend not only on how many parameters are shared, but also on how shared parameters are organized across depth. Existing static schemes adopt two extreme reuse topologies: layer-wise repetition, where each layer is reused independently, and global cyclic reuse, where the entire stack is repeatedly applied as a single shared function. We argue that sharing granularity is an important factor in recursive parameter sharing. This study proposes Block Recursive Transformers (**BRT**), which partition layers into contiguous blocks and recursively reuse each block before moving to the next. This intermediate topology enables local refinement within blocks while preserving depth-wise transitions, without adding routing modules, auxiliary states, or extra training objectives. Across decoder-only models up to 1.7B parameters, with additional encoder-decoder and encoder-only studies, **BRT** improves validation negative log-likelihood and downstream accuracy over static recursive sharing baselines. Ablations and representation analyses further show that intermediate block granularity better balances local refinement and depth-wise differentiation. Project website: https://aim-skku.github.io/BRT/

1 Introduction

Recursive Transformers reduce the number of unique model parameters by reusing layers while preserving the executed depth of the network (Shen et al., 2022; Bae et al., 2025a; Geiping et al., 2025). This makes parameter sharing attractive for compact language models, where the goal is to retain the benefits of deep computation while reducing the number of unique trainable layers.

Existing static sharing strategies follow two extreme reuse topologies. Sequence sharing repeats each layer independently before moving to the next, as in Sequence-style assignment and intra-layer recurrence (Takase and Kiyono, 2023; Nguyen and Lin, 2025). This preserves a notion of depth, but treats each layer as a separate reuse unit, making it difficult to reuse multi-layer transformations as a coherent stage. In contrast, Cycle sharing repeatedly applies the shared stack as a single global function, as in cyclic or recursive reuse schemes (Takase and Kiyono, 2023; Bae et al., 2025a,b; Yu et al., 2026). This improves downstream performance, but forces the same transformation to serve all depth-specific roles, which can cause redundant updates and representational saturation (Yu et al., 2026). Other compact language model designs use layer sharing for efficient generation (Liu et al., 2024; Reid et al., 2021), but do not explicitly study recursive reuse topology as the main design axis. As illustrated in Figure 1, Sequence and Cycle sharing form two boundary cases: overly fine-grained reuse and overly global reuse.

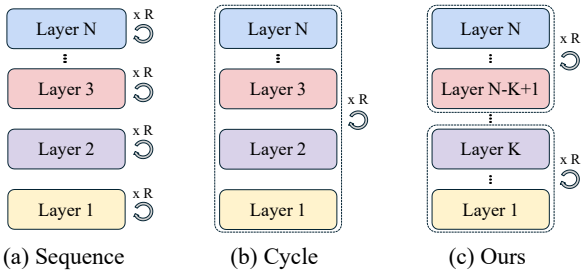


Figure 1: Comparison of sharing strategies. (a) Sequence sharing repeats each layer independently. (b) Cycle sharing reuses the full stack as one global unit. (c) Our BRT recursively reuses contiguous layer blocks before moving to the next block. Sequence and Cycle sharing are boundary cases.

This raises the central question of this work: *How should shared parameters be organized across depth?* Transformer depth is not a homogeneous sequence of interchangeable layers. Prior analyses suggest that layers at different depths often play specialized functional roles (Liu, 2025; Sun et al.,

†Corresponding author: Sungeun Hong (csehong@skku.edu)

Findings of the Association for Computational Linguistics: EMNLP 2026, pages 39879–39891
October 24-29, 2026 ©2026 Association for Computational Linguistics

2025; Shim et al., 2022). For example, lower layers tend to preserve more local and lexical information, while upper layers integrate higher-level features and become more closely tied to prediction behavior (Zhang et al., 2024). This motivates a reuse unit that is larger than a single layer, so that local transformations can be reused as a coherent stage, but smaller than the entire network, so that different depth regions can remain distinct.

In this study, we introduce *Block Recursive Transformers* (**BRT**), a recursive sharing architecture that partitions depth into contiguous blocks and recursively reuses each block before moving to the next. This intermediate topology enables local refinement, repeated application of a short multi-layer transformation within the same depth region, while preserving transitions across depth. We add no routing modules (Bae et al., 2025b; Yu et al., 2026), auxiliary memory states (Yu et al., 2026), layer-wise adaptation modules (Bae et al., 2025a), or extra training objectives. By only changing how shared layers are organized and executed, **BRT** isolates reuse topology as the main variable within a unified static sharing framework.

Our method improves downstream accuracy and validation negative log-likelihood over static recursive sharing baselines, and ablations and representation analyses support the benefit of intermediate block granularity. Rather than adding adaptive mechanisms, BRT isolates how static shared layers should be organized across depth, treating reuse topology as the main design axis for comparing Sequence, Cycle, and block-level reuse within a unified framework.

Our main contributions are as follows:

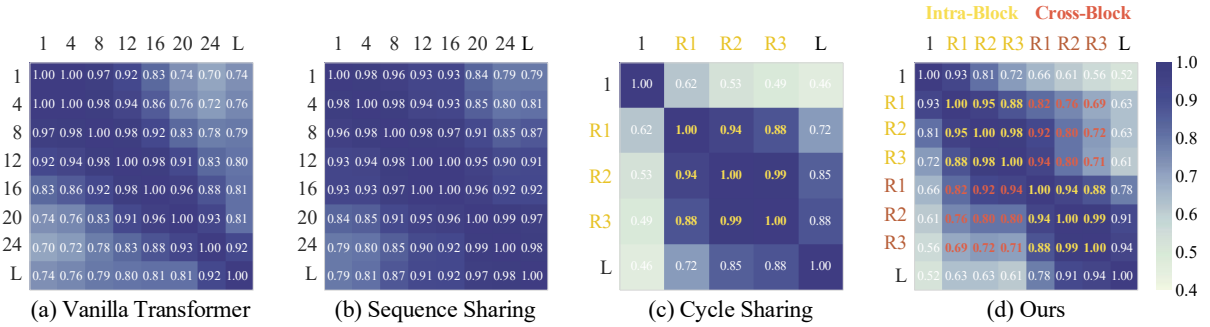
- We formulate static recursive parameter sharing as a reuse-topology problem and introduce *Block Recursive Transformers* (**BRT**), which recursively reuse contiguous groups of layers.
- We show that Sequence and Cycle sharing are boundary cases of the same block-wise formulation, enabling controlled comparison across reuse granularities without routing modules, auxiliary states, or extra objectives.
- We evaluate **BRT** across decoder-only model sizes, data budgets, and recursion settings, with additional encoder-decoder and encoder-only validation-NLL studies. **BRT** improves over static recursive sharing baselines with fewer unique trainable parameters.

2 Related Work

Cross-layer Parameter Sharing. Parameter sharing has been widely studied as a way to reduce the number of unique parameters in Transformers. Early approaches impose strong tying across depth. Universal Transformers apply the same transformation recurrently across layers (Dehghani et al., 2019), while ALBERT shares parameters across Transformer layers to reduce memory usage (Lan et al., 2020). Although effective for compression, reusing a single layer can limit the representational capacity of different depth regions.

Recent approaches relax this constraint by sharing some components or adding layer-specific capacity. Subformer combines middle-layer sharing with embedding factorization for generative Transformers (Reid et al., 2021), and EdgeFormer augments shared layers with layer-specific adaptation for efficient on-device generation (Ge et al., 2022). MobileLLM uses layer sharing for compact sub-billion language models (Liu et al., 2024), while Basis Sharing decomposes layer weights into shared bases and layer-specific coefficients (Wang et al., 2025). Overall, these methods mainly study which parameters are shared or how sharing constraints are relaxed. In contrast, **BRT** studies how shared parameters are reused after they are tied. It keeps the layer parameterization unchanged, but reorganizes execution by grouping adjacent layers into recursive blocks. This shifts the focus from parameter tying itself to the topology of reuse across depth.

Recursive and Adaptive-depth Transformers. Recursive Transformers construct depth by repeatedly applying shared layers, making parameter reuse an explicit iterative computation mechanism. Relaxed Recursive Transformers improve this idea with layer-wise LoRA adaptation (Bae et al., 2025a), while recurrent-depth models study repeated latent computation for test-time scaling (Geiping et al., 2025). Related looped Transformer studies further show that repeated computation can support iterative refinement and algorithmic behavior (Yang et al., 2024; Giannou et al., 2023; Fan et al., 2025; Gatmiry et al., 2024). Most static recursive sharing schemes, however, rely on either layer-wise repetition or a single global loop, corresponding to Sequence and Cycle sharing strategies (Takase and Kiyono, 2023). These topologies are parameter-efficient, but they either fragment reuse into individual layers or force a single shared stack



(a) Vanilla Transformer (b) Sequence Sharing (c) Cycle Sharing (d) Ours

Figure 2: Centered Kernel Alignment (CKA) similarity across depth and recursion steps. (a) *Vanilla Transformer* shows gradual representation changes across depth. (b) *Sequence sharing* shows high similarity across layers, suggesting limited depth-wise differentiation. (c) *Cycle sharing* shows high similarity across recursion steps, consistent with representational saturation under global reuse. (d) Our BRT shows stronger intra-block similarity and lower cross-block similarity, suggesting local refinement while preserving transitions across depth regions.

to serve all depth-specific roles.

Recent adaptive-depth methods introduce mechanisms to decide how much computation each token should receive. Mixture-of-Recursions learns token-level recursive depths through routing (Bae et al., 2025b), while MeSH introduces a routing mechanism to support recursive computation (Yu et al., 2026). Unlike these methods, which change the amount or path of computation through routing, memory states, or adaptive schedules, we keep the computation schedule fixed and change only the granularity of reuse. This makes **BRT** a controlled setting for studying reuse topology and an orthogonal complement to routing- or memory-based recursive models.

3 Block Recursive Transformers

3.1 Reuse Granularity Analysis

BRT treats reuse granularity as a controllable design axis. Instead of reusing individual layers or the full stack, it recursively reuses contiguous groups of layers, reflecting the observation that Transformer layers often form depth-dependent functional groups rather than a uniform stack of interchangeable transformations (Geva et al., 2021; Sun et al., 2025; Zhang et al., 2024).

To examine how different reuse topologies affect depth-wise representations, we use Centered Kernel Alignment (CKA) (Kornblith et al., 2019) as a diagnostic tool. CKA measures the similarity between hidden-state representations, allowing us to compare how representations evolve across layers and recursion steps. Importantly, CKA is not used to choose block boundaries. Instead, **BRT** adopts a fixed contiguous partition, so that the number of blocks directly controls reuse granularity and can be evaluated through ablations.

Figure 2 shows the resulting CKA patterns. The vanilla Transformer exhibits gradual representation changes across depth, suggesting that adjacent layers often form locally coherent stages. Sequence sharing shows high similarity across layers, indicating limited depth-wise differentiation when each layer is reused independently. Cycle sharing shows high similarity across recursion steps, consistent with representational saturation under global reuse (Yu et al., 2026). In contrast, **BRT** shows higher similarity within each block and lower similarity across blocks. This pattern supports the view that block-level reuse enables local refinement within each depth region while preserving transitions between different regions. Additional CKA visualizations are provided in Appendix B.

3.2 Formulation

We start from a standard L-layer Transformer (Vaswani et al., 2017), whose hidden states are updated as:

$$h^{\ell} = \mathcal{B}\left(h^{\ell-1}; \theta_{\ell}\right), \qquad \ell \in [1, L], \tag{1}$$

where h^0 is obtained from the token embedding, $\mathcal{B}$ denotes a Transformer block, and θ_ℓ denotes the parameters of layer ℓ.

Block Reuse. Given a target executed depth L, let E denote the number of non-recursive boundary layers and R the recursion count. Following prior recursive Transformer settings (Bae et al., 2025a,b), the recurrent core contains $P = (L - E)/R$ unique recurrent layers when exact depth matching is feasible. The total number of unique layers

is then $T = P + E$, and the executed depth is $L = P \cdot R + E$.

We partition the P unique recurrent layers into S contiguous blocks, where $1 \leq S \leq P$. For clarity, we first describe the case where P is divisible by S, so each block contains $K = P/S$ layers. When this divisibility condition is not satisfied, we use the closest feasible contiguous partition: we choose the largest recurrent-core size $P' \leq P$ such that P' is divisible by S, set $K = P'/S$, and use $D_{\text{exec}} = P'R + E$ as the executed depth. This rule keeps block sizes equal, preserves contiguity, and never gives **BRT** a larger executed depth than the target vanilla model. We report the resulting executed depth for all configurations in Appendix A.

For block s, we define its computation as:

$$\mathcal{F}_s = \mathcal{B}(\cdot; \theta_{s,K}) \circ \cdots \circ \mathcal{B}(\cdot; \theta_{s,1}), \tag{2}$$

where $\theta_{s,1}, \ldots, \theta_{s,K}$ are the parameters of the K layers in block s. The composition follows the original layer order within each block, matching the forward execution order of a standard Transformer.

The block count S controls the granularity of reuse. As S increases, each reuse unit becomes smaller and the topology approaches layer-wise repetition. When $S = 1$, the recurrent core becomes a single global loop. Intermediate values $1 < S < P$ define **BRT**. In this regime, each block is applied R times before representations move to the next block, enabling local refinement within contiguous layer groups while preserving transitions across depth.

Design Rationale. We use contiguous blocks to keep recursive refinement local while allowing different depth regions to remain distinct. Thus, block-level reuse provides an intermediate topology between layer-wise repetition and global cyclic reuse.

By default, we set $E = 2$, keeping the first and last layers independent of the recursive core, following prior work (Bae et al., 2025b). The overall computation is then:

$$h_{\text{out}} = \mathcal{B}\Big(\mathcal{F}_S^R\Big(\cdots \mathcal{F}_1^R(\mathcal{B}(h^0; \theta_1))\Big); \theta_L\Big), \tag{3}$$

where $\mathcal{F}_s^R$ denotes R repetitions of block $\mathcal{F}_s$, and θ_1, θ_L denote the independent boundary layers.

Unifying Static Sharing Topologies. With S controlling reuse granularity and E controlling boundary-layer sharing, prior static strategies arise as boundary cases.

- $S = 1$ gives global cyclic reuse, where the recurrent core is reused as one block. With $E = 0$, this recovers Cycle sharing; with $E = 2$, it corresponds to Middle-Cycle sharing.
- $S = P$ gives layer-wise repetition, where each recurrent layer is repeated before moving to the next. With $E = 0$, this recovers Sequence sharing; with $E = 2$, it corresponds to Middle-Sequence sharing.
- Intermediate values $1 < S < P$ define **BRT**, interpolating between these extremes through block-level recursive reuse.

4 Experiments

We evaluate **BRT** to answer three questions: (i) whether block-level reuse improves over static recursive sharing baselines under matched training budgets, (ii) whether the gains persist across model scale, data scale, and recursion depth, and (iii) whether the same reuse topology transfers beyond decoder-only architectures.

4.1 Experimental Setup

We use LLaMA-based SmolLM models up to 1.7B parameters (Touvron et al., 2023; Allal et al., 2024), pretrained from scratch on FineWeb-Edu (Penedo et al., 2024). We evaluate ARC, WinoGrande, MMLU, HellaSwag, and PIQA using the Language Model Evaluation Harness (Sutawika et al., 2024), and report mean performance across three seeds. We further evaluate encoder-decoder T5-220M and encoder-only ModernBERT-149M (Raffel et al., 2020; Warner et al., 2025). For language-modeling quality, we compute validation negative log-likelihood on 2,000 FineWeb-Edu samples.

Unless otherwise stated, recursive variants are compared under the same recursion count and training-token budget. When exact depth matching is infeasible, we use the closest feasible recursive configuration following prior recursive-sharing setups (Bae et al., 2025a,b). Reported parameter counts denote unique trainable non-embedding parameters. All main decoder-only models use the same optimizer, learning-rate schedule, batch size, and hardware setup; full details are provided in Appendix A.

4.2 Main Results

We first compare **BRT** with static recursive sharing baselines using SmolLM-360M trained from

Model	Recursion		Pretrain		Few-shot Accuracy ↑					
	N_r	Share	Param	N_{tok}	ARC	WG	MMLU	HS	PIQA	Avg
Vanilla	-	-	315M	20B	34.87 ± 0.31	50.92 ± 0.41	25.12 ± 0.17	28.32 ± 0.16	59.45 ± 0.32	39.84 ± 0.14
Recursive	2	Seq	157M	20B	32.90 ± 0.40	50.03 ± 0.42	24.88 ± 0.34	27.51 ± 0.22	59.61 ± 0.31	38.98 ± 0.30
		Cyc	157M	20B	34.12 ± 0.50	50.37 ± 1.13	24.39 ± 0.26	28.37 ± 0.10	**59.90** ± 0.43	39.43 ± 0.20
		M-Seq	167M	20B	32.96 ± 0.11	50.76 ± 0.34	25.21 ± 0.41	28.02 ± 0.34	59.17 ± 0.50	39.22 ± 0.23
		M-Cyc	167M	20B	34.73 ± 0.21	51.12 ± 0.63	25.12 ± 0.19	28.59 ± 0.12	59.23 ± 0.10	39.76 ± 0.12
		Ours	157M	20B	**35.48** ± 0.35	51.32 ± 0.64	25.46 ± 0.22	**28.98** ± 0.07	59.74 ± 0.50	**40.20** ± 0.23
	4	Seq	78M	20B	32.03 ± 0.33	50.83 ± 1.08	24.38 ± 0.21	27.34 ± 0.22	57.88 ± 0.28	38.49 ± 0.26
		Cyc	78M	20B	33.48 ± 0.23	50.22 ± 1.04	**25.51** ± 0.18	28.46 ± 0.10	58.60 ± 0.35	39.25 ± 0.13
		M-Seq	98M	20B	32.56 ± 0.54	51.79 ± 0.52	24.90 ± 0.32	27.62 ± 0.46	58.61 ± 0.53	39.09 ± 0.38
		M-Cyc	98M	20B	33.67 ± 0.23	51.89 ± 0.59	25.18 ± 0.23	28.47 ± 0.23	59.55 ± 0.24	39.75 ± 0.13
		Ours	78M	20B	34.43 ± 0.09	**51.91** ± 0.41	25.32 ± 0.16	28.73 ± 0.35	59.43 ± 0.21	39.96 ± 0.10

Table 1: Main results on SmolLM-360M under a fixed 20B-token training budget. We compare vanilla and recursive Transformers with different static sharing strategies. N_r denotes the number of recursion steps, and N_{tok} denotes the number of training tokens. The best average performance is highlighted in bold. Results are reported as mean and standard deviation across 3 seeds.

scratch for 20B tokens. The baselines include Sequence and Cycle sharing, as well as Middle-Sequence and Middle-Cycle, where the first and last layers are kept independent of the recursive core. We evaluate two recursion settings, $R = 2$ and $R = 4$, which reduce the number of unique recurrent layers under matched or closest feasible executed depth. All models are evaluated on five benchmarks, and we report task-wise accuracy and average performance over three seeds. Detailed configurations and executed-depth accounting for all recursive variants are provided in Appendix A.

As shown in Table 1, **BRT** achieves the best average few-shot accuracy among static recursive sharing baselines under comparable or smaller unique-parameter budgets. With $R = 2$, **BRT** reaches 40.20 average accuracy, making it competitive with the vanilla model while using substantially fewer unique parameters. It also improves over Middle-Cycle, the strongest recursive baseline in this setting. With $R = 4$, **BRT** outperforms all recursive baselines and remains close to the vanilla model while using only 78M parameters. These results suggest that downstream performance depends not only on the number of shared parameters, but also on how reused layers are organized across depth.

4.3 Scaling with Compute and Data

Training Compute. We evaluate language-modeling quality under different training compute budgets. Figure 3 reports validation negative log-likelihood (NLL) for SmolLM-360M variants with $R = 2$ and $R = 4$.

We compute NLL on 2,000 FineWeb-Edu validation samples and estimate training compute following Bae et al. (2025b), based on the scaling-law compute estimate of Kaplan et al. (2020). Across both recursion settings, **BRT** achieves the lowest validation NLL among static sharing strategies at all evaluated compute budgets. With $R = 2$, **BRT** improves over both Sequence and Cycle variants, showing that block-level reuse is beneficial even under moderate recursion. The gains are more pronounced with $R = 4$, where stronger compression leads to larger degradation for prior sharing variants. This trend is consistent with the view that global cyclic reuse can produce redundant updates, while overly fine-grained layer-wise reuse is also suboptimal. In contrast, **BRT** provides a more stable quality-compute trade-off by reusing intermediate contiguous blocks. Together with Table 1, these results show that **BRT** improves both downstream accuracy and validation NLL.

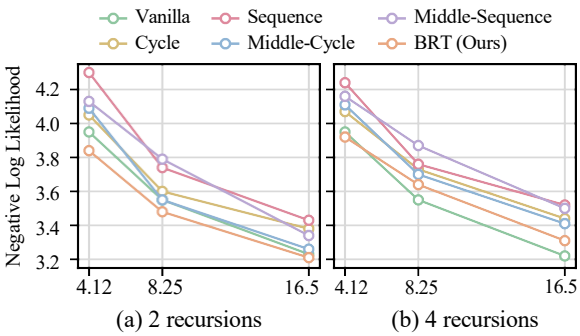


Figure 3: Validation NLL of SmolLM-360M variants under different training compute budgets. The x-axis shows estimated training FLOPs ($\times 10^{18}$). NLL is computed on 2,000 FineWeb-Edu validation samples. All models are pretrained from scratch; sharing configurations are listed in Table 5.

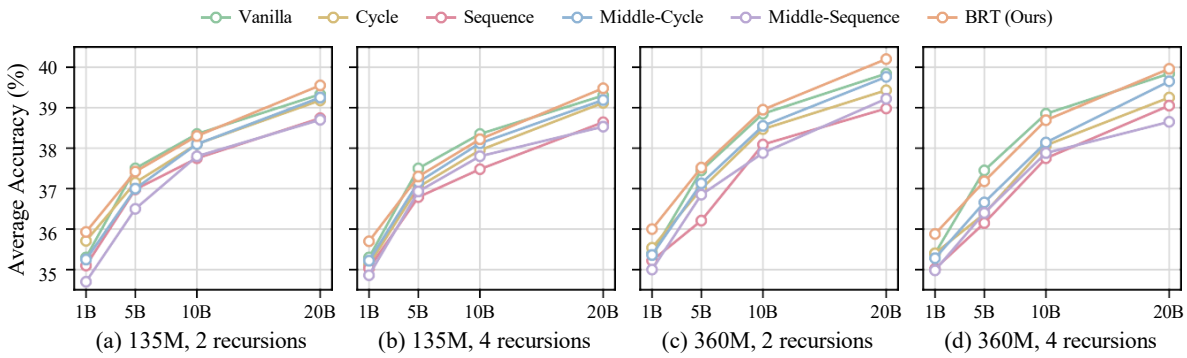


Figure 4: Comparison of parameter-sharing strategies under matched training-token budgets. We pretrain SmolLM-135M and SmolLM-360M variants with 1B, 5B, 10B, and 20B tokens under two and four recursion settings. We report mean accuracy over five few-shot benchmarks across three seeds.

Model	Recursion		Pretrain		Few-shot Accuracy ↑					
	N_r	Share	Param	N_{tok}	ARC	WG	MMLU	HS	PIQA	Avg
Vanilla	-	-	1.61B	20B	38.12 ± 0.22	**51.55** ± 0.72	25.97 ± 0.26	**31.76** ± 0.17	61.03 ± 0.16	**41.68** ± 0.25
Recursive	2	M-Cyc	870M	20B	37.65 ± 0.26	50.93 ± 0.89	25.64 ± 0.21	31.31 ± 0.32	**61.13** ± 0.12	41.33 ± 0.20
	2	Ours	805M	20B	**38.25** ± 0.29	51.30 ± 0.61	**26.10** ± 0.18	31.69 ± 0.22	60.82 ± 0.27	<u>41.63</u> ± 0.22

Table 2: Comparison of parameter-sharing strategies in vanilla and recursive Transformers using SmolLM-1.7B. All models are trained for 20B tokens. We report task-wise and mean accuracy across three seeds. The best overall performance is highlighted in bold, while the best recursive strategy is underlined.

Matched Training Tokens. We next evaluate how different sharing strategies scale with training tokens. Figure 4 reports mean accuracy over five benchmarks using SmolLM-135M and SmolLM-360M trained with 1B, 5B, 10B, and 20B tokens.

We evaluate both $R = 2$ and $R = 4$, and report mean performance over three seeds. For SmolLM-135M, vanilla models remain stronger than recursive variants in some low-data regimes, consistent with prior observations on small recursive models (Bae et al., 2025a,b). However, **BRT** continues to benefit from additional training data and improves over the vanilla model at 20B tokens for both recursion settings. For SmolLM-360M, **BRT** outperforms static recursive baselines across token budgets and remains competitive with the vanilla model, especially under $R = 2$. Middle-Cycle also improves over Cycle at larger scale, suggesting that keeping the first and last layers independent is beneficial. Overall, these results suggest that block-level reuse becomes more stable as model size and training budget increase.

Billion-scale Architectures. To test whether the gains of **BRT** extend beyond sub-billion models, we evaluate SmolLM-1.7B recursive architecture variants trained for 20B tokens. We compare **BRT** against Middle-Cycle, the strongest recursive baseline in our main experiments, and the vanilla model. We report task-wise and average performance across three seeds. In Table 2, **BRT** achieves 41.63 average accuracy, outperforming the Middle-Cycle variant with 41.33 average accuracy. It also remains competitive with the vanilla model, which achieves 41.68 average accuracy with approximately twice as many trainable parameters. The largest gain over Middle-Cycle appears on ARC, suggesting that block-level reuse can better preserve downstream performance in this setting. These results suggest that the benefits of **BRT** are not limited to sub-billion models and remain effective at billion-scale model sizes.

4.4 In-depth Analysis

We conduct ablation studies to analyze the effect of block granularity in **BRT** and to verify whether its gains come from the block-level reuse topology itself, rather than solely from adding boundary-independent layers. We pretrain SmolLM-360M from scratch on 10B tokens and evaluate all parameter-sharing variants on five few-shot benchmarks across three seeds. Table 3 provides the full task-wise results, while Figure 5 reports the average accuracy with maximum and minimum values

Sharing			Pretrain		Few-shot Accuracy ↑					
S	Untied	Share	Param	N_{tok}	ARC	WG	MMLU	HS	PIQA	Avg
1	No	**BRT**	157M	10B	31.35 ± 0.39	50.75 ± 0.59	24.93 ± 0.26	27.33 ± 0.13	58.01 ± 0.16	38.47 ± 0.24
2	No	**BRT**	157M	10B	$\mathbf{32.10} \pm 0.35$	50.31 ± 0.26	24.74 ± 0.22	27.70 ± 0.11	$\mathbf{59.05} \pm 0.75$	38.77 ± 0.24
3	No	**BRT**	147M	10B	32.08 ± 0.36	50.31 ± 0.36	24.53 ± 0.31	27.54 ± 0.13	57.92 ± 0.31	38.47 ± 0.04
4	No	**BRT**	157M	10B	32.07 ± 0.17	49.40 ± 0.53	24.35 ± 0.29	27.49 ± 0.11	58.48 ± 0.32	38.34 ± 0.07
16	No	**BRT**	157M	10B	30.42 ± 0.50	51.22 ± 1.17	24.71 ± 0.19	26.79 ± 0.20	57.20 ± 0.20	38.09 ± 0.32
1	Yes	**BRT**	167M	10B	31.16 ± 0.20	$\mathbf{51.85} \pm 1.09$	24.70 ± 0.19	27.39 ± 0.16	57.65 ± 0.44	38.55 ± 0.28
2	Yes	**BRT**	157M	10B	31.95 ± 0.45	51.57 ± 0.16	24.84 ± 0.21	27.47 ± 0.08	58.94 ± 0.53	$\mathbf{38.95} \pm 0.07$
3	Yes	**BRT**	167M	10B	31.92 ± 0.38	50.76 ± 0.37	$\mathbf{24.96} \pm 0.36$	$\mathbf{27.78} \pm 0.08$	58.44 ± 0.13	38.76 ± 0.08
4	Yes	**BRT**	138M	10B	31.47 ± 0.32	50.60 ± 0.71	24.68 ± 0.27	27.23 ± 0.13	58.13 ± 0.44	38.42 ± 0.17
15	Yes	**BRT**	167M	10B	30.31 ± 0.58	51.28 ± 1.16	24.23 ± 0.17	27.37 ± 0.04	57.18 ± 0.68	37.90 ± 0.29

Table 3: Block-count ablation of **BRT** on SmolLM-360M. S denotes the number of blocks. The *Untied* column indicates whether the first and last layers are excluded from recursive sharing. The largest S differs by setting because the recursive core has a different number of layers when boundary layers are untied. $S = 1$ corresponds to cycle-style sharing, while the largest S values correspond to sequence-style sharing. All models are trained for 10B tokens and evaluated on five benchmarks.

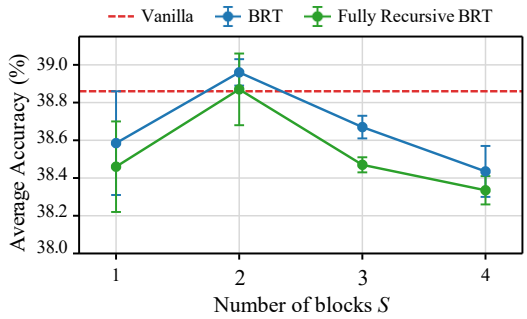


Figure 5: Block-granularity analysis of BRT on SmolLM-360M architecture. Fully Recursive BRT denotes the setting without independent boundary layers ($E = 0$). All models are trained for 10B tokens. Points show mean accuracy over five benchmarks and three seeds; error bars show the seed-wise range.

across seeds.

Block Granularity. We first vary the number of blocks S, while keeping the first and last layers independent. As shown in Table 3 and Figure 5, **BRT** achieves the best performance when $S = 2$, reaching 38.95 average accuracy and outperforming both the vanilla model and recursive sharing baselines. Increasing the number of blocks beyond this point does not further improve performance. Although $S = 3$ remains competitive with the vanilla model and outperforms Cycle sharing, $S = 4$ degrades downstream accuracy and even falls below the Cycle baseline. These results suggest that **BRT** benefits from an intermediate block granularity, where each recursive unit is large enough to coordinate local transformations but still avoids the overly global reuse imposed by Cycle sharing ($S = 1$).

Fully Recursive Setting. We then remove the independent layers and force all layers to participate in the sharing strategy. This setting tests whether the advantage of **BRT** persists when the model becomes fully recursive. As shown in Table 3, **BRT** with $S = 2$ outperforms both Cycle sharing ($S = 1$) and Sequence sharing ($S = 16$), while remaining competitive with the vanilla model. This result is important because it shows that **BRT** can recover most of the vanilla model's performance, unlike prior recursive sharing baselines.

In the fully recursive setting, we also observe that finer block partitions are not necessarily better. Increasing the number of blocks beyond two does not improve performance, following the same trend as when including the boundary-independent layers. As the block granularity becomes too fine, **BRT** approaches a structure similar to Sequence sharing and may inherit its limitations.

Overall, the $S = 2$ configuration provides the best balance between global reuse, as in Cycle sharing, and layer-level independent reuse, as in Sequence sharing. The fully recursive setting further shows that the gains of **BRT** cannot be explained solely by keeping the first and last layers independent. Although independent layers improve performance, block-level recursion remains stronger than prior sharing baselines even when all layers are recursively reused. These findings suggest that even with a smaller number of parameters and sometimes smaller executed depth, **BRT** provides a strong inductive bias for recursive Transformers.

4.5 Architecture Generalization

To examine whether the benefits of **BRT** extend beyond decoder-only architectures, we evaluate it on encoder-decoder and encoder-only Transformers. We use T5-220M for the encoder-decoder setting

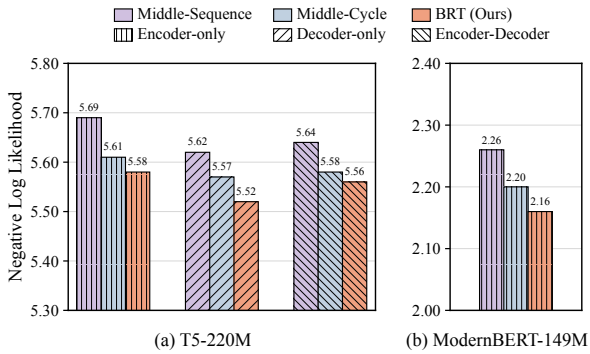


Figure 6: Comparison on encoder-decoder and encoder-only architectures under matched training tokens. (a) T5-220M with recursion applied to both stacks, decoder only, or encoder only. (b) Encoder-only ModernBERT-149M. All models are trained for 10B tokens, and NLL is computed on 2,000 FineWeb-Edu validation samples.

and ModernBERT-149M for the encoder-only setting (Raffel et al., 2020; Warner et al., 2025).

For each architecture, we compare **BRT** against the strongest Cycle and Sequence sharing variants, Middle-Cycle and Middle-Sequence. All models are pretrained from scratch on 10B tokens from FineWeb-Edu (Penedo et al., 2024), and we report the negative log-likelihood on the FineWeb-Edu validation set. Since T5 and ModernBERT differ in architecture and training objective, we analyze them separately rather than comparing their absolute NLL values.

Encoder-decoder Architecture. We first evaluate T5-220M. Since encoder-decoder models contain two Transformer stacks, we ablate where the recursive structure is applied. We consider three variants: applying recursion to both the encoder and decoder, only to the decoder, and only to the encoder. This allows us to analyze whether **BRT** is more effective in the bidirectional encoder, the autoregressive decoder, or their combination.

In Figure 6, **BRT** achieves the lowest negative log-likelihood in all three settings. When both encoder and decoder are recursive, **BRT** reduces NLL from 5.64 to 5.56. The same trend holds when only one stack is recursive, with the best NLL of 5.52 under decoder-only recursion. We hypothesize that tying the decoder's cross-attention projection across recursions makes each decoder block act as an iterative source-conditioned refinement step.

Encoder-only Architecture. We next evaluate an encoder-only architecture (ModernBERT-149M) to test whether the benefits of **BRT** also hold under a masked language modeling objective. Unlike T5, ModernBERT consists of a single bidirectional Transformer stack. In Figure 6, **BRT** achieves the lowest validation NLL among encoder-only variants, improving over both Middle-Sequence and Middle-Cycle. This result indicates that the advantage of **BRT** is not limited to autoregressive decoding or encoder-decoder attention. Even in a fully bidirectional encoder, organizing parameter reuse into blocks provides a more effective recursive structure than either repeating individual layers or cycling through the entire stack.

Overall, these results suggest that block-level reuse is not specific to decoder-only models. Across encoder-decoder and encoder-only architectures, **BRT** consistently improves over prior static sharing baselines, supporting block granularity and reuse topology as important design factors for recursive Transformers.

5 Conclusion

We introduced Block Recursive Transformers (**BRT**), a static recursive sharing architecture that treats reuse topology as a key design factor. The proposed method partitions layers into contiguous blocks and recursively reuses each block before transitioning to the next, providing an intermediate topology between layer-wise repetition and global cyclic reuse. Across model scales, data budgets, recursion settings, and architecture types, **BRT** improves validation negative log-likelihood and downstream accuracy over static recursive sharing baselines. Ablations and representation analyses further show that intermediate block granularity better balances local refinement and depth-wise differentiation. More broadly, our findings suggest that the Recursive Transformer community should consider not only how much sharing is used, but also how reuse is organized across depth.

Limitations

Our analysis uses a fixed block partition to study block-level recursion in a controlled setting. We do not examine flexible partitioning strategies, such as asymmetric block designs or learned block partitioning. Investigating adaptive block structures across model scales and architectures is a future direction. In addition, our task coverage is limited to five few-shot evaluation benchmarks. Evaluating **BRT** on broader tasks, including multilingual settings, coding, and complex reasoning, is another future direction.

Acknowledgements

This work was supported by the National Research Foundation (NRF) grant funded by the Korean government (MSIT) (RS-2026-25477177). This research was supported by the MSIT (Ministry of Science and ICT), Korea, under the Graduate School of Virtual Convergence support program (IITP-2026-RS-2023-00254129, 33%) and the ICAN (ICT Challenge and Advanced Network of HRD) support program (IITP-2026-RS-2024-00436934, 33%), supervised by the IITP (Institute for Information & Communications Technology Planning & Evaluation).

References

Loubna Ben Allal, Anton Lozhkov, Elie Bakouch, Leandro von Werra, and Thomas Wolf. 2024. SmolLM - blazingly fast and remarkably powerful.

Sangmin Bae, Adam Fisch, Hrayr Harutyunyan, Ziwei Ji, Seungyeon Kim, and Tal Schuster. 2025a. Relaxed recursive transformers: Effective parameter sharing with layer-wise LoRA. In *Proc. of Int'l Conf. on Learning Representations (ICLR)*, pages 34282–34327.

Sangmin Bae, Yujin Kim, Reza Bayat, Sungnyun Kim, Jiyoun Ha, Tal Schuster, Adam Fisch, Hrayr Harutyunyan, Ziwei Ji, Aaron Courville, and Se-Young Yun. 2025b. Mixture-of-recursions: Learning dynamic recursive depths for adaptive token-level computation. In *Proc. of Neural Information Processing Systems (NeurIPS)*, volume 38, Main Conference, pages 96572–96617.

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. 2020. PIQA: Reasoning about physical commonsense in natural language. In *Proc. of Int'l Conf. on Artificial Intelligence (AAAI)*, volume 34, pages 7432–7439.

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.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Lukasz Kaiser. 2019. Universal transformers. In *Proc. of Int'l Conf. on Learning Representations (ICLR)*.

Ying Fan, Yilun Du, Kannan Ramchandran, and Kangwook Lee. 2025. Looped transformers for length generalization. In *Proc. of Int'l Conf. on Learning Representations (ICLR)*, pages 14502–14520.

Khashayar Gatmiry, Nikunj Saunshi, Sashank Reddi, Stefanie Jegelka, and Sanjiv Kumar. 2024. Can looped transformers learn to implement multi-step gradient descent for in-context learning? In *Proc. of Int'l Conf. on Machine Learning (ICML)*, pages 15130–15152.

Tao Ge, Si-Qing Chen, and Furu Wei. 2022. EdgeFormer: A parameter-efficient transformer for on-device Seq2seq generation. In *Proc. of Conf. on Empirical Methods in Natural Language Processing (EMNLP)*, pages 10786–10798.

Jonas Geiping, Sean McLeish, Neel Jain, John Kirchenbauer, Siddharth Singh, Brian Bartoldson, Bhavya Kailkhura, Abhinav Bhatele, and Tom Goldstein. 2025. Scaling up test-time compute with latent reasoning: A recurrent depth approach. *Proc. of Neural Information Processing Systems (NeurIPS)*, 38:41340–41391.

Mor Geva, Roei Schuster, Jonathan Berant, and Omer Levy. 2021. Transformer feed-forward layers are key-value memories. In *Proc. of Conf. on Empirical Methods in Natural Language Processing (EMNLP)*, pages 5484–5495.

Angeliki Giannou, Shashank Rajput, Jy-yong Sohn, Kangwook Lee, Jason D Lee, and Dimitris Papailiopoulos. 2023. Looped transformers as programmable computers. In *Proc. of Int'l Conf. on Machine Learning (ICML)*, pages 11398–11442. PMLR.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2021. Measuring massive multitask language understanding. In *Proc. of Int'l Conf. on Learning Representations (ICLR)*.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling laws for neural language models.

Simon Kornblith, Mohammad Norouzi, Honglak Lee, and Geoffrey Hinton. 2019. Similarity of neural network representations revisited. In *Proc. of Int'l Conf. on Machine Learning (ICML)*, pages 3519–3529. PMLR.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2020. ALBERT: A Lite BERT for self-supervised learning of language representations. In *Proc. of Int'l Conf. on Learning Representations (ICLR)*.

Jing Liu. 2025. Layer specialization underlying compositional reasoning in transformers. *arXiv preprint arXiv:2510.17469*.

Zechun Liu, Changsheng Zhao, Forrest Iandola, Chen Lai, Yuandong Tian, Igor Fedorov, Yunyang Xiong, Ernie Chang, Yangyang Shi, Raghuraman Krishnamoorthi, Liangzhen Lai, and Vikas Chandra. 2024. MobileLLM: Optimizing sub-billion parameter language models for on-device use cases. In *Proc. of Int'l Conf. on Machine Learning (ICML)*, volume 235, pages 32431–32454.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. In *Proc. of Int'l Conf. on Learning Representations (ICLR)*.

Anthony Nguyen and Wenjun Lin. 2025. Intra-Layer Recurrence in Transformers for Language Modeling. *Proceedings of the Canadian Conference on Artificial Intelligence*.

Guilherme Penedo, Hynek Kydlíček, Loubna Ben allal, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro Von Werra, and Thomas Wolf. 2024. The FineWeb datasets: Decanting the web for the finest text data at scale. In *Proc. of Neural Information Processing Systems (NeurIPS)*, volume 37, pages 30811–30849.

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).

Machel Reid, Edison Marrese-Taylor, and Yutaka Matsuo. 2021. Subformer: Exploring weight sharing for parameter efficiency in generative transformers. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 4081–4090.

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

Zhiqiang Shen, Zechun Liu, and Eric Xing. 2022. Sliced recursive transformer. In *Proc. of European Conf. on Computer Vision (ECCV)*, pages 727–744. Springer.

Kyuhong Shim, Jungwook Choi, and Wonyong Sung. 2022. Understanding the role of self attention for efficient speech recognition. In *Proc. of Int'l Conf. on Learning Representations (ICLR)*.

Qi Sun, Marc Pickett, Aakash Kumar Nain, and Llion Jones. 2025. Transformer layers as painters. In *Proc. of Int'l Conf. on Artificial Intelligence (AAAI)*, volume 39, pages 25219–25227.

Lintang Sutawika, Hailey Schoelkopf, Leo Gao, Baber Abbasi, Stella Biderman, Jonathan Tow, ben fattori, Charles Lovering, farzanehnakhaee70, Jason Phang, Anish Thite, Fazz, Aflah, Niklas Muennighoff, Thomas Wang, sdtblck, nopperl, gakada, tttyuntian, and 11 others. 2024. EleutherAI/lm-evaluation-harness: v0.4.3. Version v0.4.3.

Sho Takase and Shun Kiyono. 2023. Lessons on parameter sharing across layers in transformers. In *Proceedings of The Fourth Workshop on Simple and Efficient Natural Language Processing (SustaiNLP)*, pages 78–90.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, and 49 others. 2023. Llama 2: Open foundation and fine-tuned chat models.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Proc. of Neural Information Processing Systems (NeurIPS)*, volume 30.

Jingcun Wang, Yu-Guang Chen, Ing-Chao Lin, Bing Li, and Grace Li Zhang. 2025. Basis sharing: Cross-layer parameter sharing for large language model compression. In *Proc. of Int'l Conf. on Learning Representations (ICLR)*, volume 2025, pages 13386–13404.

Benjamin Warner, Antoine Chaffin, Benjamin Clavié, Orion Weller, Oskar Hallström, Said Taghadouini, Alexis Gallagher, Raja Biswas, Faisal Ladhak, Tom Aarsen, Griffin Thomas Adams, Jeremy Howard, and Iacopo Poli. 2025. Smarter, better, faster, longer: A modern bidirectional encoder for fast, memory efficient, and long context finetuning and inference. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2526–2547.

Liu Yang, Kangwook Lee, Robert D Nowak, and Dimitris Papailiopoulos. 2024. Looped transformers are better at learning learning algorithms. In *Proc. of Int'l Conf. on Learning Representations (ICLR)*.

Chengting Yu, Xiaobo Shu, Yadao Wang, Yizhen Zhang, Haoyi Wu, Jiaang Li, Rujiao Long, Ziheng Chen, Yuchi Xu, Wenbo Su, and Bo Zheng. 2026. MeSH: Memory-as-state-highways for recursive transformers. In *Proc. of Int'l Conf. on Learning Representations (ICLR)*.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. HellaSwag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4791–4800.

Yang Zhang, Yanfei Dong, and Kenji Kawaguchi. 2024. Investigating layer importance in large language models. In *Proceedings of the 7th BlackboxNLP Workshop: Analyzing and Interpreting Neural Networks for NLP*, pages 469–479.

Appendix

This appendix provides additional implementation details and supporting analyses for the experiments included in the main paper. We include the model configurations used across architectures, describe the training and evaluation settings, and specify how executed depth and parameter counts are computed for each sharing strategy. We also provide additional visualization results to complement the CKA analysis of the main paper, and include the training dynamics of different sharing strategies. We organize these points as follows:

- Appendix A provides the full experimental setup for the experiments across decoder-only, encoder-only and encoder-decoder architectures, details of the different configurations for the parameter-sharing strategies, and explains how depth is computed for **BRT**.
- Appendix B extends the CKA visualizations from the main body, including a layer-by-layer visualization of the vanilla and Sequence-sharing strategy, and includes training-dynamic visualizations for **BRT**, Middle-Cycle and Middle-Sequence.

A Implementation Details

Table 4 summarizes the five base model configurations used in our experiments. These models cover the three Transformer architecture families considered in this work: decoder-only, encoder-decoder, and encoder-only architectures. For each model we report the original parameter count and executed depth before applying the different recursive parameter sharing strategies. We describe their training setup in the following subsections.

Decoder-only experiments. For our main experiments, we use a LLaMA-based (Touvron et al., 2023) SmolLM architecture (Allal et al., 2024). We evaluate the 135M, 360M, and 1.7B variants to compare parameter-sharing strategies across model scales. Unless otherwise specified, all models are pretrained from scratch on a subset of the FineWeb-Edu dataset (Penedo et al., 2024) for up to 20B tokens, depending on the experimental setting. For **BRT**, we use $S = 2$ blocks by default.

For training, we use 8 NVIDIA A6000 GPUs with a total batch size of 1024. We use AdamW (Loshchilov and Hutter, 2019) as the optimizer with $\beta_1 = 0.9$ and $\beta_2 = 0.95$, and set the peak learning rate to 3×10^{-3}. Training is performed using bfloat16 precision, and gradients are clipped to a maximum norm of 1.0. We use a trapezoidal learning rate schedule with a 5% warmup and 10% linear cooldown. We set the maximum input sequence length to 2048.

Models	Description		
	Architecture	**Total Params**	**Layers**
SmolLM	Decoder-only	135M	30
	Decoder-only	360M	32
	Decoder-only	1.7B	24
T5	Encoder-decoder	220M	12+12
ModernBERT	Encoder-only	149M	22

Table 4: Description of the models used in our experiments. Total parameters include embedding and non-embedding parameters. The reported values correspond to the original model versions, considering both the embedding and non-embedding parameters.

We evaluate the models on five benchmarks from the Language Model Evaluation Harness (Sutawika et al., 2024): ARC, WinoGrande (WG), MMLU, HellaSwag (HS), and PIQA. We follow standard settings from prior work (Sakaguchi et al., 2021; Bisk et al., 2020; Clark et al., 2018; Zellers et al., 2019; Hendrycks et al., 2021), and evaluate using 5 shots for all benchmarks except ARC, where 25 shots are used. In addition, we also evaluate the models on the validation set of FineWeb-Edu and report the negative log-likelihood on 2,000 validation samples.

Encoder-decoder and encoder-only experiments. For generalization purposes, we also conduct experiments on encoder-decoder and encoder-only architectures, using T5-220M and ModernBERT-149M, respectively (Raffel et al., 2020; Warner et al., 2025). In both settings, we train recursive variants from scratch for 10B tokens on the FineWeb-Edu dataset, and evaluate validation negative log-likelihood on 2,000 FineWeb-Edu samples (Penedo et al., 2024).

For both architectures, we use AdamW with $\beta_1 = 0.9$ and $\beta_2 = 0.95$, a total batch size of 1024, and 8 NVIDIA A6000 GPUs. Training is performed in bfloat16 precision, and gradients are clipped to a maximum norm of 0.5. We set the peak learning rate to 3×10^{-4} and use a trapezoidal learning-rate schedule with a 10% linear decay.

Unlike the decoder-only experiments, the encoder-decoder experiments use a span-corruption

Algorithm 1 *BRT Depth Calculation*

Given a target vanilla depth L and a recursive configuration (E, R, S), the procedure returns the largest feasible executed depth $D_{exec} \leq L$, together with the number of unique recurrent layers P and layers per block K.

Input: vanilla depth L, independent layers $E \in \{0, 2\}$, recursions $R \in \{2, 4\}$, blocks S
Output: BRT executed depth D_{exec}, unique recurrent layers P, layers per block K
if $(L - E) \bmod R = 0$ **then**
  $P \leftarrow (L - E)/R$ ▷ is divisible
else
  $P \leftarrow \lfloor (L - E)/R \rfloor$ ▷ select 1 less
end if
if $P \bmod S = 0$ **then**
  $K \leftarrow P/S$ ▷ is divisible
else
  $K \leftarrow \lfloor P/S \rfloor$ ▷ select 1 less
  $P \leftarrow S \cdot K$ ▷ Adjust P
end if
$D_{exec} \leftarrow E + R \cdot S \cdot K$ ▷ Final executed depth ($\leq L$)
return $D_{exec},\ P,\ K$

objective. For T5, we use a 20% warmup, relative position embeddings with maximum distance 1024 and 128 buckets, a decoder target length of 448, a noise density of 15%, and a mean span length of 3.1. We set the maximum input sequence length to 2048.

For the encoder-only experiments, we use a masked language modeling objective with a 15% masking rate. ModernBERT uses the same optimizer, peak learning rate, batch size, precision, gradient clipping and learning-rate schedule as the T5 experiments, except that the warmup is reduced to 10%. The maximum input sequence length is set to 1024.

Executed Depth and Total Parameters. In Algorithm 1, we describe how we calculate executed depth for **BRT**. As indicated in Section 3, it is possible that the division operation is not feasible and that the depths do not match perfectly for all sharing strategies in all configurations. For **BRT**, we take this into account, and always select the closest depth value possible to the original vanilla depth. In particular, the depth of **BRT** will never be bigger than the vanilla depth, avoiding any kind of advantage over other baselines. For Sequence, Cycle, and their respective middle variants, we follow (Bae et al., 2025a,b), where sharing strategies are allowed to have executed depths larger than those of the vanilla models. In Table 5 we report the non-embedding parameter count, executed depth, and number of unique layers across different sharing strategies for the different experiments.

Base Model	Setting	Share	N_r	S	Param	D_{exec}	L_{uniq}
SmolLM-135M	Vanilla	–	–	–	106M	30	30
	Recursive	Seq	2	15	53M	30	15
		Cyc		1	53M	30	15
		M-Seq		14	57M	30	16
		M-Cyc		1	57M	30	16
		BRT		2	57M	30	16
		Seq	4	8	28M	32	8
		Cyc		1	28M	32	8
		M-Seq		7	32M	34	9
		M-Cyc		1	32M	34	9
		BRT		2	28M	26	8
SmolLM-360M	Vanilla	–	–	–	315M	32	32
	Recursive	Seq	2	16	157M	32	16
		Cyc		1	157M	32	16
		M-Seq		15	167M	32	17
		M-Cyc		1	167M	32	17
		BRT		2	157M	32	16
				3	147M	30	15
				4	157M	32	16
				2	157M	30	16
				3	167M	32	17
				4	138M	26	14
		Seq	4	8	78M	32	8
		Cyc		1	78M	32	8
		M-Seq		8	98M	34	10
		M-Cyc		1	98M	34	10
		BRT		2	78M	26	8
SmolLM-1.7B	Vanilla	–	–	–	1.61B	24	24
	Recursive	M-Cyc	2	1	870M	24	13
		BRT		2	805M	22	12

Table 5: Comparison of parameter count, executed depth, and unique layers for different sharing strategies on SmolLM. Base Model denotes the original SmolLM scale, Setting indicates vanilla or recursive models, Param denotes the non-embedding parameters, N_r denotes the recursion number, S denotes the number of blocks, D_{exec} denotes executed depth, and L_{uniq} denotes unique layers.

B Extended Visualization Results

In Section 3, we motivated Block Recursive Transformers by the hierarchical functional roles that emerge across depth in Transformer models. Here, we extend our CKA analysis with additional heatmap visualizations to further examine these depth-wise patterns.

As shown in Figure 7, the vanilla Transformer exhibits a smooth evolution of representations across layers, with high similarity between adjacent layers and gradual transitions through depth. This suggests that depth naturally forms a small number of functionally coherent groups rather than a sequence of homogeneous transformations, consistent with our observation that partitioning layers into two blocks improves performance. In contrast, Sequence sharing shows uniformly high similar-

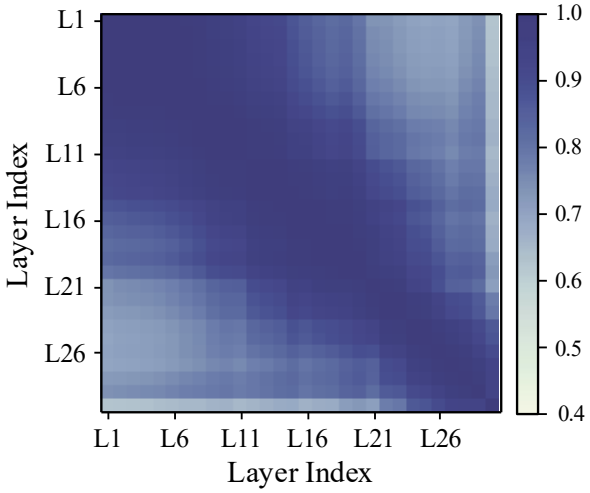


Figure 7: Visualization of the representation similarity across layers in vanilla Transformers.

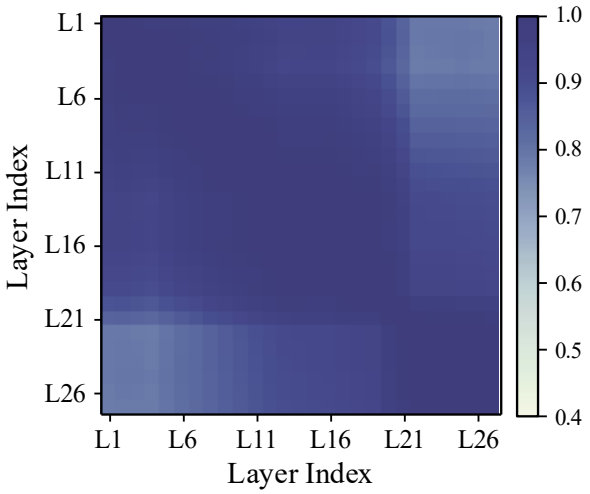


Figure 8: Visualization of the representation similarity across layers in sequence sharing.

ity across layers, including between early and late layers (Figure 8). This indicates limited depth-wise differentiation, aligning with the performance degradation observed for layer-wise sharing.

In addition, we visualize the behavior of global sharing in Figure 2. In this setting, all layers are grouped into a single shared function that is repeatedly applied. This design leads to highly redundant representations and limited evolution across recursion steps. One possible interpretation is that the model converges early to a fixed point, after which further recursions provide negligible additional computation. In contrast, **BRT** appears to better preserve the hierarchical structure of Transformers by exhibiting greater similarity among intra-block layers, with a smooth transition between layers, supporting depth specialization.

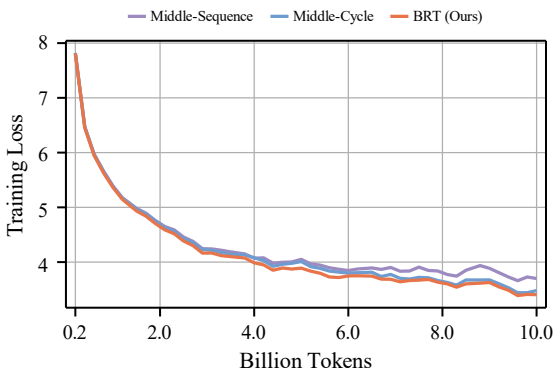


Figure 9: Visualization of the training loss using SmolLM-360M for different sharing strategies. All models are trained for 10B tokens. We set the number of recursions to 2, and the exact executed depth and number of parameters can be found in Table 5.

Training Dynamics. Figure 9 shows the training dynamics of different sharing strategies using SmolLM-360M trained for 10B tokens. All methods follow a similar trajectory in the early stage of training, suggesting that the initial optimization is comparable across sharing strategies. However, after the initial billion tokens, **BRT** starts to separate from Middle-Cycle and Middle-Sequence, and achieves consistently lower training loss. This gap becomes more visible as training progresses, especially compared to Middle-Sequence. **BRT** achieves a training loss of 3.41 at 10B tokens, followed by Middle-Cycle and Middle-Sequence at 3.48 and 3.70, respectively.

We consider these results important because they show that the advantage of block-level reuse appears during training optimization, not only in final downstream performance. In this setting, **BRT** uses fewer parameters and smaller executed depth than the corresponding recursive baselines, so the lower training loss cannot be explained by additional capacity or computation. Instead, the results suggest that a better reuse organization could help optimize the training objective. This also supports the main empirical results obtained in the main paper: intermediate reuse granularity is a more effective sharing strategy relative to cycle and sequence sharing.

Overall, these results position intermediate block granularity as a key design factor for recursive Transformers.