Value Residual Learning

Zhanchao Zhou[1,2,3] **Tianyi Wu**[4†*] **Zhiyun Jiang**[5†*] **Fares Obeid**[6] **Zhenzhong Lan**[2◇]
[1]Zhejiang University [2]Westlake University [3]Ant Group
[4]University of Electronic Science and Technology of China
[5]China University of Mining and Technology [6]Imperial College London

Abstract

While Transformer models have achieved remarkable success in various domains, the effectiveness of information propagation through deep networks remains a critical challenge. Standard hidden state residuals often fail to adequately preserve initial token-level information in deeper layers. This paper introduces ResFormer, a novel architecture that enhances information flow by incorporating value residual connections in addition to hidden state residuals. And a variant is SVFormer, where all layers share the first layer's value embedding. Comprehensive empirical evidence demonstrates ResFormer achieves equivalent validation loss with 16.11% fewer model parameters and 20.3% less training data compared to Transformer, while maintaining similar memory usage and computational cost. Besides, SVFormer reduces KV cache size by nearly half with only a small performance penalty and can be integrated with other KV-efficient methods, yielding further reductions in KV cache, with performance influenced by sequence length and cumulative learning rate.

1 Introduction

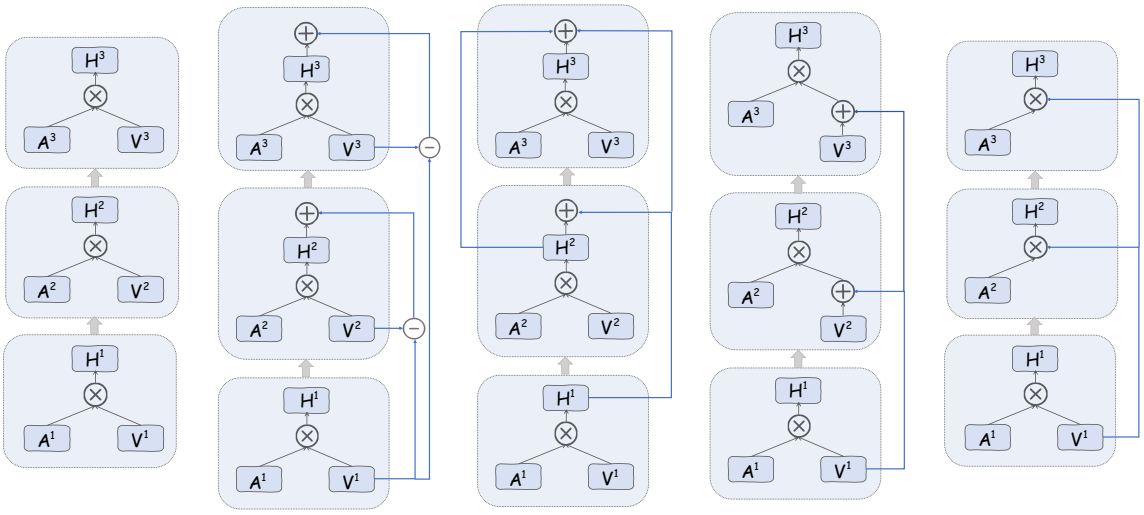
The Transformer (Vaswani et al., 2017) model has become one of the leading architectures in recent years, excelling in both language modeling (Devlin et al., 2019; Lan et al., 2020; Brown et al., 2020) and computer vision tasks (Dosovitskiy et al., 2021). Among its variants, decoder-only architectures have become the most prominent (Kaplan et al., 2020; Dubey et al., 2024). The discovery of scaling laws (Hoffmann et al., 2022; Kaplan et al., 2020) has driven the pursuit of larger Transformer models by increasing network depth and width.

In a standard decoder-only transformer, initial token embeddings contain localized information, which rapidly evolves into abstract semantic features through early attention layers (Sun et al., 2024b; Clark et al., 2019). As Transformers deepen, a critical question arises: **How effectively is the initial information propagated to deeper layers**? One common answer is that residual connections of hidden states ensure access to initial information throughout the network. However, some studies (Zhou et al., 2021; Shi et al., 2022) have identified that the smoothing effect of attention mechanisms leads to over-smoothing, where token representations become increasingly similar as the network deepens. This indicates that in deeper layers, sequence-level features become dominant, while token-level features are diluted. DenseFormer (Pagliardini et al., 2024) applied the idea of learnable dense connections from DenseNet (Huang et al., 2016) to Transformer, and the learned connection coefficients shows that deeper layers indeed require larger attention to initial embeddings. Given the low similarity between initial token embeddings and deeper hidden states (Sun et al., 2024b), their directly summation may significantly impact the modeling of attention distribution for abstract semantic information in later layers. NeuTRENO (Nguyen et al., 2023) alleviates over-smoothing from the view of regularizers by considering the difference between value vectors of the first and current layers.

In this paper, we propose ResFormer, which enhances the propagation of initial local information by introducing value residual connections in addition to the standard hidden residual connections. Specifically, ResFormer applies a residual connection between the value vectors of the current layer and the first layer before the attention operation. In other words, both value states share the existing attention matrix of the current layer. The value states of the first attention layer and the preceding hidden states differ only by a linear transformation along the channel dimension, both represent-

*Equal Contribution; †Work done during internship at Westlake University; ◇ Corresponding author.

Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 28341–28356



(a) Transformer (b) NeuTRENO (c) DenseFormer (d) ResFormer (e) SVFormer

Figure 1: Simplified illustration of the vanilla Transformer, NeuTRENO, DenseFormer, ResFormer, and SVFormer, with only three-layer structures and no operations other than attention. $\mathbf{A}^i$, $\mathbf{V}^i$, and $\mathbf{H}^i$ denote the attention matrix, value vectors, and attention outputs at the i-th layer, respectively. $\oplus$, $\ominus$, and $\otimes$ represent standard matrix addition, subtraction, and multiplication, respectively.

ing token-level raw information. We hypothesize that introducing residual connections for values has a less impact on modeling attention distributions for sequence-level semantic information in higher layers and complements the original hidden state residual. Fig. 1 illustrates a comparison of the extra skip connections introduced by different models.

During inference, deep networks require substantial KV cache, severely impacting model deployment (Xiao et al., 2024). Existing KV-efficient methods often process keys and values simultaneously. Building on ResFormer, we decouple the value from the attention operation and propose a new kind of Transformer (SVFormer) where all layers share a single value state.

We experiment on a 20B SlimPajama subsampled dataset, using settings similar to popular large language models (Wei et al., 2023; Dubey et al., 2024; Kaplan et al., 2020). We compare different models based on the valid loss against the vanilla Transformer. Results show that ResFormer outperforms the vanilla Transformer, DenseFormer, and NeuTRENO. ResFormer achieves equivalent validation loss with 16.11% fewer model parameters and 20.3% less training data compared to Transformer, while maintaining similar memory usage and computational cost. Besides, SVFormer, while reducing the KV-cache by nearly half, requires a 12.2% increase in parameters to achieve the same validation loss as Transformer. And SVFormer performs better when the training sequence length is longer. It further reduces the KV cache when integrated with GQA (Ainslie et al., 2023).

2 Related Work

2.1 Shortcut Connections

Deep learning models often consist of multiple layers, posing a challenge to minimize information loss during transmission. ResNet (He et al., 2016) mitigates the vanishing gradient problem with identity connections. Stochastic Depth (Huang et al., 2016) enhances training by randomly dropping layers. DenseNet (Huang et al., 2017) allows subsequent layers to directly access the hidden states of all preceding layers. These two methods further enhance the information flow after ResNet.

Related research indicates that, although increasing depth continues to yield performance improvements in language modeling tasks, the gains become less significant with further increases (Petty et al., 2024). Furthermore, (Zhou et al., 2021) illustrates that a 32-layer ViT underperforms a 24-layer ViT. DenseFormer (Pagliardini et al., 2024) integrates weighted fusion of outputs from all preceding layers after each layer. To explore why increasing depth in Transformers does not yield expected gains, (Wang et al., 2022) finds that self-attention acts as a low-pass filter, smoothing token representations in ViTs. Additionally, (Shi et al., 2022)

investigates over-smoothing from a graph perspective in BERT-based language modeling tasks. NeuTRENO (Nguyen et al., 2023) adds the difference between the value vectors of the first and current layers to each layer's attention output and significantly alleviates the over-smoothing problem.

2.2 KV cache compressing

The KV cache significantly impacts the efficiency of long-text model inference, attracting extensive research. One category of Transformer-based methods addresses this by employing parameter or activation value sharing techniques. The most representative works include Multi-Query Attention (Shazeer, 2019) and Grouped-Query Attention (Ainslie et al., 2023) which suggest to share key and value across a group of queries. Besides, CLA (Brandon et al., 2024) and LISA (Mu et al., 2024) respectively point out that we can reuse keys, values, or the attention matrix across layers to reduce redundancy between layers. While these methods typically process both key and value simultaneously, SVFormer is the first approach to decouple value from query and key during attention.

3 Method

3.1 Preliminary

Notations Let $\mathbf{H}_n \in \mathbb{R}^{l\times d}$ be the output hidden state of the n-th layer, where l denotes the sequence length and d is the dimension size. For each layer, the hidden state $\mathbf{H}_{n-1}$ will be firstly projected into $\mathbf{Q}_n, \mathbf{K}_n, \mathbf{V}_n \in \mathbb{R}^{l\times d}$ through three linear projections $\mathbf{W}_\mathbf{n}^\mathbf{Q}, \mathbf{W}_\mathbf{n}^\mathbf{K}, \mathbf{W}_\mathbf{n}^\mathbf{V} \in \mathbb{R}^{d\times d}$ respectively. After these projections, the attention operation (Attn), output projection ($\mathbf{W}_n^\mathbf{O} \in \mathbb{R}^{d\times d}$), and Multi-Layer-Perceptron (Mlp) are applied sequentially:

$$\mathbf{U}_n = \mathrm{Attn}(\mathbf{Q}_n, \mathbf{K}_n, \mathbf{V}_n). \quad (1)$$

$$\mathbf{H}_n = \mathrm{Mlp}(\mathbf{U}_n\mathbf{W}_n^\mathbf{O}). \quad (2)$$

NeuTRENO and DenseFormer After Eqn. 1, NeuTRENO adds the difference between the first and current layer's value:

$$\mathbf{U}_n = \mathrm{Attn}(\mathbf{Q}_n, \mathbf{K}_n, \mathbf{V}_n) + \boldsymbol{\lambda}_n(\mathbf{V}_1 - \mathbf{V}_n). \quad (3)$$

After Eqn. 2, DenseFormer performs a weighted average between all previous hidden states:

$$\mathbf{H}_n = \boldsymbol{\lambda}_{n,n}\,\mathrm{Mlp}(\mathbf{U}_n\mathbf{W}_n^\mathbf{O}) + \sum_{i=0}^{n-1}\boldsymbol{\lambda}_{n,i}\mathbf{H}_i. \quad (4)$$

where $\mathbf{H}_0 = \mathrm{Embedding}(\mathbf{X})$ for the input $\mathbf{X}$. $\boldsymbol{\lambda}_n$ in Eqn. 3 and $\{\boldsymbol{\lambda}_{n,i}\}_{i=0}^{n-1}$ in Eqn. 4 are new parameters. Unless noted, $\boldsymbol{\lambda}_n$ is set to 0.4 for NeuTRENO suggested by (Nguyen et al., 2023). For DenseFormer, only $\boldsymbol{\lambda}_{n,n}$ are set to 1 and the others are set to zero during initialization here.

3.2 ResFormer

In contrast, before Eqn. 1, ResFormer introduces a skip connection from the first layer's value $\mathbf{V}_1 = \mathbf{H}_0\mathbf{W}_1^\mathbf{V}$ to current layer's value $\mathbf{V}_n = \mathbf{H}_{n-1}\mathbf{W}_n^\mathbf{V}$:

$$\mathbf{V}_n = \boldsymbol{\lambda}_{n,1}\mathbf{V}_1 + \boldsymbol{\lambda}_{n,2}\mathbf{H}_{n-1}\mathbf{W}_n^\mathbf{V}. \quad (5)$$

where $\boldsymbol{\lambda}_{n,1}$ and $\boldsymbol{\lambda}_{n,2}$ are flexible scalars.

When all $\boldsymbol{\lambda}_{n,1}$ and $\boldsymbol{\lambda}_{n,2}$ are predetermined constants, it is termed **Constant-ResFormer**. If $\{\boldsymbol{\lambda}_{n,1} = \boldsymbol{\lambda}_{n,2}\}_{n=1}^N$, where N is the total number of layers, the model is called **Identity-ResFormer**. Another variant where some layers have $\boldsymbol{\lambda}_{n,1} = 0$ is referred to as **Sparse-ResFormer**. Besides, if $\boldsymbol{\lambda}_{n,1}$ and $\boldsymbol{\lambda}_{n,2}$ are trainable parameters, the model is termed **Learnable-ResFormer**. Unless otherwise specified, $\boldsymbol{\lambda}_{n,1}$ and $\boldsymbol{\lambda}_{n,2}$ are initialized to 0.5 for Learnable-ResFormer and are predetermined as 0.5 for Identity-ResFormer. Furthermore, given that higher layers require more supplementary information from $\mathbf{V}_1$, we propose the **Learnable ResFormer Plus**. The initialization strategy is as follows: 1). $\boldsymbol{\lambda}_{n,2}$ is initialized to 0.5, where $n = 1, 2, \ldots, N$. 2). $\boldsymbol{\lambda}_{n,1}$ is initialized to $\boldsymbol{\lambda}_{\text{scale}} \cdot \frac{e^{\lambda'_{n,1}}}{\sum_{j=1}^N e^{\lambda'_{j,1}}}$, where $\boldsymbol{\lambda}_{\text{scale}}$ is initialized to the total layer number N and is shared across all layers.

A more general form is **Dense-ResFormer**, defined as $\mathbf{V}_n = \boldsymbol{\lambda}_{n,n}\mathbf{H}_{n-1}\mathbf{W}_n^\mathbf{V} + \sum_{i=1}^{n-1}\boldsymbol{\lambda}_{n,i}\mathbf{V}_i$ for $n \geq 2$, where $\{\boldsymbol{\lambda}_{n,i}\}_{i=1}^{n-1}$ are constants or trainable scalars. Unless noted, all $\boldsymbol{\lambda}_{n,n}$ are set to 1.

3.3 SVFormer

Shared Parts	-	values	keys	keys & values
Valid Loss	2.739	2.743	2.753	2.776

Table 1: Results of sharing different parts every 2 layers.

Beyond ResFormer, SVFormer adopts standard attention in the first layer and obtain the attention output $\mathbf{U}_\mathbf{n}$ for n-th layer when $n \geq 2$ through $\mathbf{U}_n = \mathbf{A}_n\mathbf{V}_1$, where $\mathbf{A}_n$ is the attention matrix of n-th layer. Its main advantage is that it only requires computing and storing the value vectors for the first layer, saving nearly half of the KV cache

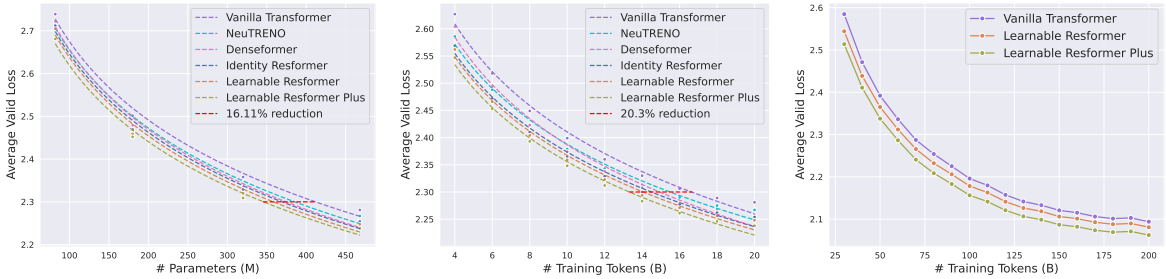


Figure 2: (Left) Validation loss as model size scales from 82M to 468M parameters on 20B tokens. (Medium) Validation loss for the 468M parameter model evaluated every 2B tokens. ResFormer achieves approximately 16.1%-20.3% reduction in both model parameters and training data. (Right) Validation loss for the 1.6B parameter model evaluated every 10B tokens.

Model	Wiki. PPL	LMB. PPL	Arc-c ACC	Arc-e ACC	BoolQ ACC	Hella. ACC	LMB. ACC	Obqa ACC	Wino. ACC	PiQA ACC	Avg. ACC
Transformer	24.8	33.4	18.8	49.6	57.5	31.1	34.1	17.6	49.8	66.1	40.6
NeuTRENO	24.3	36.3	20.7	48.8	59.2	31.6	34.2	19.0	51.8	65.9	41.4
DenseFormer	24.0	**28.0**	20.1	48.7	56.5	32.1	36.6	17.6	49.3	65.6	40.8
Identity ResFormer	23.8	32.9	20.2	49.0	59.0	32.1	36.0	16.8	51.2	66.1	41.3
Learnable ResFormer	23.7	32.5	**21.2**	**50.3**	**60.9**	32.3	**36.3**	**18.8**	**51.2**	67.0	**42.3**
Learnable ResFormer plus	**23.2**	31.4	**21.2**	49.7	60.6	**32.4**	36.0	17.8	51.1	**67.5**	42.0

Table 2: Downstream evaluation of different models with 468M parameters trained on 20B tokens.

during inference. Similar methods like CLA reduce KV cache by sharing both of the key and value vectors every two layers. However, the results in Table 1 show that sharing values has less negative impact compared with sharing keys.

4 Experiments

4.1 Setting

Training Details Following (Brandon et al., 2024), we choose the Llama-like architecture and SlimPajama (Soboleva et al., 2023) data for main experiments. Specifically, the architecture includes pre-normalization, SwiGLU activations (Shazeer, 2020), rotary position embedding (Su et al., 2024), and no dropout. For SlimPajama, we randomly sample nearly 20B tokens based on the original data distribution of seven domains during training and adopt tokenizer used for "RedPajama-INCITE-7B-Base". See Table 12 in Appendix for data details.

Unless otherwise noted, we train all models using AdamW optimizer with 0.1 weight decay, $\beta_1 = 0.9$, $\beta_2 = 0.95$ and the max grad norm 1.0. The batch size is set to be around 2M tokens (Zhang et al., 2024) with a sequence length of 2,048 and the total steps is fixed 10,000 steps (Kaplan et al., 2020). We adopt linear learning rate warmup for the first 1,200 steps with the initial learning rate and the peak learning rate to be 1e-7 and 6e-4 respectively. The cosine decay schedule gradually decays to 10% of the peak learning rate by the end of training (Zhou et al., 2024; Wei et al., 2023). The detailed hyperparameters for models of various sizes and different training sequence lengths in Appendix A.3. Moreover, All models are trained with 8 Nvidia A100 80G GPUs using mixed-precision training in FP16. We adopt deepspeed zero-2 optimizer and flash attention mechanism.

Scaling Details We conduct scaling experiments on a 1.6B parameter model using approximately 200B tokens of internal pre-training data. Training is performed with a sequence length of 8,192, learning rate of 1.5e-4, and batch size of 2M tokens, requiring about 3,584 H800 GPU hours per run.

Evaluation For model evaluation and comparison, we primarily utilized the average validation loss across seven domains, computed on the entire SlimPajama validation split. Additionally, we randomly selected a fixed set of 1,000 sample sequences for subsequent visualization analysis.

We also compare different models on several classical reasoning tasks following (Zhang et al., 2024) in a zero-shot way. The tasks include Hellaswag (Hella.) (Zellers et al., 2019) Openbookqa (Obqa.) (Mihaylov et al., 2018), WinoGrande (Wino.) (Sakaguchi et al., 2019), ARC-Easy (Arc-e) and ARC-Challenge (Arc-c) (Clark et al., 2018) and PiQA (Bisk et al., 2020). In addition to accu-

racy on reasoning tasks, we also reports the perplexity (PPL) on Wikitext (Wiki.) and LAMBADA (LMB.).

4.2 ResFormer vs. NeuTRENO, DenseFormer

We analyze how different models scale with model size and data size under similar experimental settings. We train models with 82M, 180M, 320M, and 468M parameters on 20B tokens and evaluate on a validation set. Fig.2 (Left) shows ResFormer achieves equivalent validation loss to Transformer while using 16.11% fewer parameters. We also evaluate 468M models every 2B tokens, finding ResFormer requires 20.3% fewer training tokens to match Transformer's loss (Fig.2 (Medium)). All ResFormer variants demonstrate superior scaling compared to NeuTRENO and DenseFormer. Table 2 confirms ResFormer outperforms other models on downstream tasks, with Learnable ResFormer achieving 1.7 point average accuracy improvement over vanilla Transformer.

To validate the performance improvement of value residual at larger training scales, we conduct additional experiments following the scaling details in Sec.4.1. As shown in Fig.2 (Right), value residual consistently improves performance throughout training.

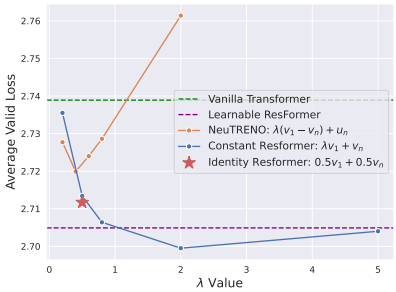


Figure 3: The impact of varying $\boldsymbol{\lambda}$ values on 82M 8-layer Constant-ResFormer and NeuTRENO.

Both Constant-ResFormer and NeuTRENO rely on predetermined $\boldsymbol{\lambda}$ constants. Fig. 3 shows the performance curves of these models against varying $\boldsymbol{\lambda}$. Results indicate that Constant-ResFormer significantly outperforms NeuTRENO and demonstrates greater robustness across a wider range of $\boldsymbol{\lambda}$ values, achieving optimal performance at $\boldsymbol{\lambda} = 2$.

Furthermore, we test the performance of other varaints of ResFormer mentioned in Sec.3.2. The $\boldsymbol{\lambda}$ values for Constant-ResFormer and Sparse-ResFormer were optimized through multiple experiments. All ResFormer variants, including the simplest Identity-ResFormer, show significant performance improvements. However, manually-tuned Sparse-ResFormer and Learnable-ResFormer-Plus outperform the standard Learnable-ResFormer. It demonstrates the challenge to pre-determine the optimal layers for $\mathbf{V}_1$ connections and their corresponding $\boldsymbol{\lambda}_1$ values in more general scenarios. Interestingly, the third to last row shows that Sparse-ResFormer achieved better performance despite having three fewer $\mathbf{W}^{\mathbf{V}}$.

Model	Initial Form	Loss
Baselines		
Vanilla Transformer	-	2.739
DenseFormer*	$\mathbf{H}'_n = 1\times\mathbf{H}_n + \sum_{i=0}^{n-1} 0\times\mathbf{H}_i$	2.722
NeuTRENO	$\mathbf{U}'_n = 0.4(\mathbf{V}_1 - \mathbf{V}_n) + \mathbf{U}_n$	2.72
ResFormer		
Identity-ResFormer	$\mathbf{V}'_n = 0.5\mathbf{V}_1 + 0.5\mathbf{V}_n$	2.712
Dense-ResFormer*	$\mathbf{V}'_n = \sum_{i=1}^{n} 1\times\mathbf{V}_i$	2.709
Learnable-ResFormer*	$\mathbf{V}'_n = 0.5\mathbf{V}_1 + 0.5\mathbf{V}_n$	2.705
Constant-ResFormer	$\mathbf{V}'_n = 2\mathbf{V}_1 + \mathbf{V}_n$	2.7
Sparse-ResFormer	$\begin{cases} \mathbf{V}'_n = \mathbf{V}_n, 1 \le n \le 5 \\ \mathbf{V}'_n = \mathbf{V}_1, 6 \le n \le 8 \end{cases}$	2.696
Sparse-ResFormer	$\begin{cases} \mathbf{V}'_n = \mathbf{V}_n, 1 \le n \le 5 \\ \mathbf{V}'_n = 5\mathbf{V}_1 + \mathbf{V}_n, \\ \qquad 6 \le n \le 8 \end{cases}$	2.687
ResFormer-Plus*	See Sec.3.2	2.681

Table 3: Average valid loss for 8-layer, 82M-parameter models. "Initial form" shows deviations from vanilla transformer. Red numbers are the $\boldsymbol{\lambda}$ values from Eqn. 3, Eqn. 4, and Eqn. 5. For models marked with "*" , $\boldsymbol{\lambda}$ is learnable, and the red numbers indicate the initial value; otherwise, red numbers are fixed constants.

4.3 Truly Better or Just Faster?

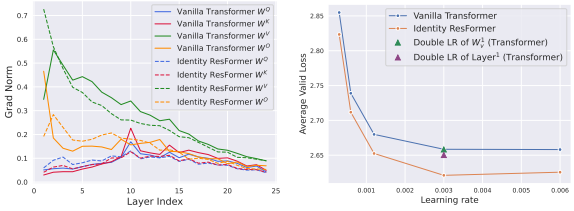


Figure 4: (Left) Average gradient norms of model outputs with respect to parameter matrices across different layers in Transformer and ResFormer. (Right) Comparison of Transformer and ResFormer performance across various learning rates during training.

To verify that ResFormer's performance improvements are not solely due to accelerated training from its shortcuts, we examined model performance across different learning rates. We com-

pared Identity ResFormer and vanilla Transformer under five learning rate settings. As shown in Fig. 4 (Right), both models achieved optimal results around a learning rate of 0.003, with Identity ResFormer significantly outperforming vanilla Transformer across all rates.

Analysis of the grad norm for the four parameter matrices ($\mathbf{W^Q}$, $\mathbf{W^K}$, $\mathbf{W^V}$, $\mathbf{W^O}$) in each layer's attention module revealed that Identity ResFormer's output had approximately twice the grad norm for $\mathbf{W}_1^{\mathbf{V}}$ and half for $\mathbf{W}_1^{\mathbf{O}}$ in the first layer compared to vanilla Transformer. This indicates that a portion of the gradient originally propagated to $\mathbf{V}_1$ through $\mathbf{H}_1$ is now transmitted via the value residual directly for Identity ResFormer.

In this way, we conducted the other two ablation experiments on vanilla Transformer: doubling the learning rate for only the first layer, and doubling it exclusively for $\mathbf{W}_1^{\mathbf{V}}$ in the first layer. Neither modification yielded significant improvements. This further demonstrates that the performance improvements brought by ResFormer are unrelated to the changes in gradient magnitude.

4.4 Ablation Study of Value Residual

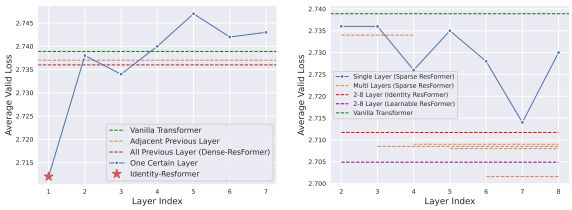


Figure 5: (Left) Impact of value skip connections source from different layers on model performance, where all connections are identity connections and $\boldsymbol{\lambda} = 1$ in Dense-ResFormer. (Right) Average validation loss of various Sparse-ResFormer configurations, which retain only single or multiple skip connections from $\mathbf{V}_1$.

Where from, where to? We analyzed which value skip-connections are necessary for the vanilla transformer. For an 8-layer transformer, we added various pre-defined value skip-connections (with constant $\boldsymbol{\lambda}$) and evaluated the resulting validation loss. As shown in Fig. 5 (Left), we first examined the impact of skip-connections from different sources. Our findings indicate that only skip-connections originating from the first layer's value ($\mathbf{V}_1$) yield significant performance improvements. Skip-connections from the second layer's value ($\mathbf{V}_2$) offer no significant benefit to subsequent layers. Skip-connections from later layers, occurring only in the final few layers, even lead to performance degradation. Both of the two special cases in Fig. 5 (Left) include $\mathbf{V}_1$ skip-connections. However, when these connections occur only between adjacent layers, the information in $\mathbf{V}_1$ fails to effectively reach the final layers. Conversely, dense value skip-connections dilute the impact of $\mathbf{V}_1$ with information from other sources.

Furthermore, we investigated spare ResFormer, a variant of identity ResFormer where the value residual connection $\mathbf{V}'_n = 0.5\mathbf{V}_1 + 0.5\mathbf{V}_n$ is applied selectively to specific layers. As shown in Fig. 5 (Right), for an 8-layer model, when limited to a single layer, applying the residual connection to the 7th layer yields the most significant improvement. When applied to multiple layers, the greatest benefit is observed when incorporating layers 6 to 8. Extending the residual connection to earlier layers, such as the 5th, diminishes the overall effect. It suggests that the model's final few layers benefit most from the first layer's value information.

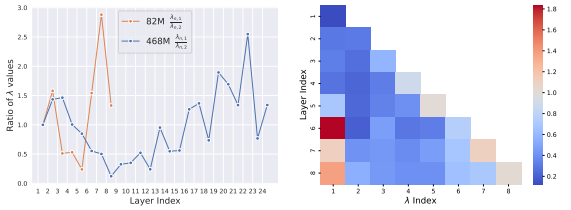


Figure 6: (Left) Visualization of $\boldsymbol{\lambda}_1/\boldsymbol{\lambda}_2$ across different layers in the 82M and 468M Learnable-ResFormer. (Right) Heatmap visualization of learned $\boldsymbol{\lambda}$ across different layers in the 468M Dense-Learnable-ResFormer.

We further trained 8-layer and 24-layer Learnable-ResFormers, as well as an 8-layer Learnable-Dense-ResFormer, and visualized the learned $\boldsymbol{\lambda}$ values. As shown in Fig. 6, the later layers tend to require more value residual connections from $\mathbf{V}_1$, which aligns with the findings in Fig. 5. Fortunately, the Learnable-ResFormer can, to some extent, identify similar sparse residual patterns to those of the best performing Sparse-ResFormer in Table 3. Notably, the Learnable-Dense-ResFormer learns value residual patterns that closely resemble those of the Learnable-ResFormer.

Why needed beyond hidden residual? Our experiments revealed that $\mathbf{V}_1$ information provides additional benefits to later network layers, despite both $\mathbf{H}_0$ and $\mathbf{V}_1$ containing initial, unfused token information. $\mathbf{H}_0$ is propagated through default hidden residual connections, but it may be diluted by

Residual Type	Initial Form	Valid Loss
-	-	2.7389
value	$\mathbf{V}'_n = 0.5\mathbf{V}_1 + 0.5\mathbf{V}_n$	2.705
hidden	$\mathbf{H}'_n = 0.5\mathbf{H}_0 + 0.5\mathbf{H}_n$	2.781
value	$\mathbf{V}'_n = 0\times\mathbf{V}_1 + 1\mathbf{V}_n$	2.73
hidden	$\mathbf{H}'_n = 0\times\mathbf{H}_0 + 1\mathbf{H}_n$	2.722

Table 4: Comparison of additional value residual (to $\mathbf{V}_1$) and hidden residual (to $\mathbf{H}_0$) connections against the default hidden residual, under various λ initializations. Trainable λ parameters are highlighted in red.

subsequent information, hindering its effective utilization in later layers. To test this hypothesis, we introduced an additional skip connection to $\mathbf{H}_0$: $\mathbf{H}'_n = \lambda_0\mathbf{H}_1 + \lambda_2\mathbf{H}_n$, where λ is learnable. We conducted experiments with two λ initialization settings and compared them to value residual.

Results showed that when $\lambda_1 = \lambda_2$ initially, the extra hidden residual had adverse effects. However, initializing $\lambda_1 = 0$ yielded some improvements, suggesting possible dilution of $\mathbf{H}_0$ information. Nevertheless, these gains were smaller than those from value residual connections, which consistently outperformed vanilla transformers across different initializations. Actually, the connection $\mathbf{H}_0 : \mathbf{H}'_n = \lambda_1\mathbf{H}_1 + \lambda_2\mathbf{H}_n$, is similar to applying residuals to queries, keys, and values at the same time, may disrupt attention distributions and hinder higher-level semantic information fusing. The reduction performance brought by identity residuals of queries or keys shown in Table 6 can support it.

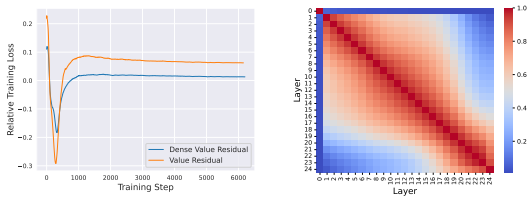


Figure 7: (Left) The relative training loss curve between different cross layer residual and vanilla hidden residual. (Right) Layer-to-layer hidden states similarity.

Why $\mathbf{V}_1$ instead of $\mathbf{V}_2$? In Fig. 5 (Left), connections to $\mathbf{V}_1$ show significant improvement, while those to $\mathbf{V}_2$ yield minimal gains. This likely occurs because the original hidden residual propagates information from $\mathbf{H}_1$ to the network ($\mathbf{V}_2 = \mathbf{H}_1\mathbf{W}_2^{\mathbf{V}}$). To verify, we adjusted residual connections, introducing them at different points.

Hidden Residual Starts Place	Value Residual Target Place	Valid Loss
$\mathbf{H_0}$ (Default)	-	2.739
$\mathbf{H_0}$ (Default)	$\mathbf{V_1}$	2.712
$\mathbf{H_0}$ (Default)	$\mathbf{V_2}$	2.738
$\mathbf{H_1}$	-	2.78
$\mathbf{H_1}$	$\mathbf{V_2}$	2.78
$\mathbf{H_2}$	-	2.82
$\mathbf{H_2}$	$\mathbf{V_2}$	2.787
$\mathbf{H_2}$	-	2.82
$\mathbf{H_2}$	$\mathbf{V_3}$	2.833
$\mathbf{H_3}$	-	3.057
$\mathbf{H_3}$	$\mathbf{V_3}$	2.883

Table 5: Comparison of performance across different value residual target and varying hidden residual start settings. “value residual target place" $\mathbf{V}_i$ indicates the earliest value accessible to subsequent layers, while “hidden residual starts place" denotes the earliest hidden state available, without prior residual connections.

For example, starting from $\mathbf{H}_1$, we use $\mathbf{H}_1 = Layer_1(\mathbf{H}_0)$ instead of $\mathbf{H}_1 = Layer_1(\mathbf{H}_0) + \mathbf{H}_0$.

Table 5 results show that when residual connections begin from $\mathbf{H}_0$ or $\mathbf{H}_1$, allowing $\mathbf{H}_2$ and subsequent layers access to $\mathbf{H}_1$, $\mathbf{V}_2 + \mathbf{V}_n$ offers no improvement. However, starting from $\mathbf{H}_2$, skip connections from $\mathbf{V}_2$ provide substantial benefits. Regarding the disparity in information propagation between $\mathbf{V}_2$ (via $\mathbf{H}_1$) and $\mathbf{V}_1$ (via $\mathbf{H}_0$), we posit that after the first layer's integration, $\mathbf{H}_1$ contains higher-level semantic information more similar to subsequent hidden states, see Fig. 7 (Right). This may ensure that the attention distribution remains relatively undisturbed when connecting to $\mathbf{H}_1$. Besides, Fig. 7 (Left) shows that Dense-ResFormer performs better than ResFormer when there is no cross layer hidden residual.

Residual Type	Valid Loss
-	2.739
Query	2.742
Key	2.746
Attention	2.757
Value	2.712

Table 6: The impact of various residual types, where all residual connections adopt a form similar to $\mathbf{V}'_n = 0.5\mathbf{V}_1 + 0.5\mathbf{V}_n$.

Residual Mapping	Valid Loss
-	2.739
Identity Mapping	3.137
Cross Layer Attention	2.729
Current Attention	2.712

Table 7: Comparison of different mapping matrices when adding $\mathbf{V}_1$ to $\mathbf{U}_n$, with “Current Attention" corresponding to Identity-ResFormer.

Superior to other residual For vanilla transformers, to better propagate information from the first layer, new residual connections can be introduced at various points in addition to the existing hidden residual: query states $\mathbf{Q}$, key states $\mathbf{K}$, value states $\mathbf{V}$, and post-softmax attention matrix $\mathbf{A}$. Results in Table 6 indicate that only the value residual connection improves performance. When connecting $\mathbf{V}_1$ and $\mathbf{V}_n$, three approaches free of extra parameters are possible: (1) the proposed residual connection, directly summing the two and then sharing an attention matrix; (2) cross layer attention $(\text{Softmax}\left(\mathbf{Q}_n \text{ Concat}(\mathbf{K}_n, \mathbf{K}_1)^T\right) \text{Concat}(\mathbf{V}_n, \mathbf{V}_1))$, recomputing an attention matrix for $\mathbf{V}_1$ based on $\mathbf{K}_1$ and $\mathbf{Q}_n$; and (3) directly adding $\mathbf{V}_1$ to $\mathbf{U}_n$ in Eqn. 1, equivalent to using an identity mapping as $\mathbf{V}_1$'s attention matrix in layer N. The second approach significantly increases computational cost. Results in Table 7 demonstrate that sharing the attention matrix yields the best performance.

4.5 Post-Analysis of ResFormer

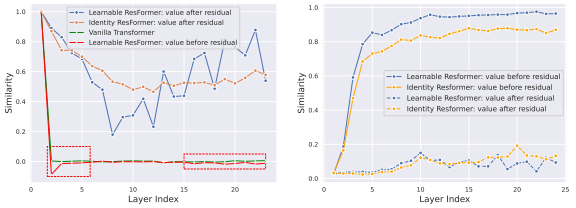


Figure 8: (Left) Similarity between first layer values and other layers' values. (Right) Token-to-token similarity across sequence for value states at different place.

How value residual works? We performed post-analysis on trained ResFormer and vanilla Transformer models to understand value residual learning. Fig. 8 (Left) shows cosine similarities between value states at different layers and the first layer, averaged across token positions. For ResFormer, we calculated this before and after applying value residual. Results show that in vanilla Transformers, the first layer's value has low similarity with other layers. In contrast, ResFormer maintains high similarity between the first layer's value and the post-residual values in subsequent layers due to value residual connections. Notably, in layers where ResFormer relies more heavily on the first layer's value (see Fig. 6), the pre-residual value exhibits lower similarity with the first layer's value, indicating that $\mathbf{W}^{\mathbf{V}}$ in these layers is learning the value residual.

For ResFormer, we also examined the average pairwise similarity between tokens' values before and after the residual connection. The results Fig. 8 (Right) reveal that with value residual connections, the learned values (before the value residual) from each layer become increasingly similar as the network deepens. We hypothesize that this is because, given the default hidden residual and value residual, each layer learns a $\Delta\mathbf{V}$, with the magnitude of necessary adjustments decreasing in later layers. This phenomenon is unique to ResFormer and not observed in vanilla Transformers.

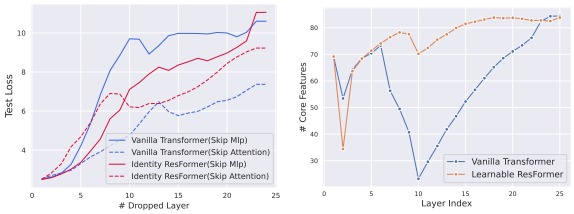


Figure 9: (Left) The change in test loss as model modules are progressively removed, starting from the back to front while keeping the first layer intact. (Right) The number of core features in each layer's hidden state after PCA dimensionality reduction, where core features represent the minimum number of principal components required to explain 99% of the variance.

Representation and Module Analysis We analyzed the overall network changes, focusing on the hidden state representation capabilities and the contributions of different modules. (Tyukin et al., 2024) suggests that removing Attention in Transformers has a significantly smaller impact than removing Mlp. We progressively removed attention or MLP layers, starting from the last layer while retaining the first layer. Fig. 9 (Left) demonstrates that for ResFormer, the impact of removing Attention is more comparable to that of removing Mlp, in contrast to vanilla Transformers. This indicates that the Attention in ResFormer, with value residual, contribute more significantly to each layer's hidden states than in vanilla Transformers.

Furthermore, we performed PCA dimensionality reduction on the hidden states of each layer in both ResFormer and vanilla Transformer models. We determined the minimum number of principal components required to explain 99% of the variance. Fig. 9 reveals that ResFormer, starting from the second layer where value residual connections are introduced, consistently produces hidden states with a higher minimum number of principal components compared to vanilla Transformers. This suggests that ResFormer generates hidden states

with higher information density.

4.6 SVFormer vs. GQA,CLA

Sequence Length	Model	Valid Loss
2,048	-	2.739
	CLA2	2.776
	GQA2	2.748
	SVFormer	2.774
64,000	-	2.753
	CLA2	2.793
	GQA2	2.773
	SVFormer	2.7485

Sequence Length	Model	Valid Loss
64,000	-	2.753
	GQA8	2.807
	CLA2 +GQA4	2.815
	SVFormer +GQA4	2.741

Table 8: Comparison of valid loss under varying degrees of KV cache reduction. CLA2 denotes parameter sharing every two layers, while GQA2 indicates halving the key-value heads. Left: Model with nearly $1/2$ KV cache. Right: Model with nearly $1/8$ KV cache.

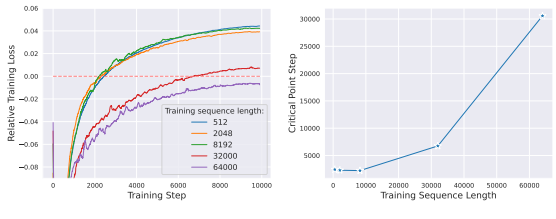


Figure 10: Left: Relative training loss for SVFormer $vs.$ vanilla Transformer under different sequence lengths with a fixed batch size of 2M tokens. Right: Analysis of critical point, and we predict it for length 64,000 using linear regression with the last 1,000 data points.

Model Type	Learning Rate	Warnup Steps	Valid Loss
Llama	1e-4	120	-0.033
	3e-4	120	+0.021
	6e-4	120	+0.035
	6e-4	1,200	+0.036
GPT2	6e-4	120	+0.029

Table 9: Relative validation loss of SVFormer compared to vanilla Transformer under different hyper-parameter settings.

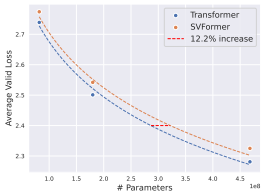


Figure 11: Validation loss for SVFormer as model size scales from 82M to 468M.

In the Table 8, at a training sequence length of 64,000, SVFormer demonstrates lower final loss compared to existing KV-efficient methods such as CLA and GQA. Moreover, it can be used concurrently with GQA to enhance KV efficiency further. However, we observed that with a training sequence length of 2,048, SVFormer underperforms compared to GQA. The results indicate that sequence length significantly affects SVFormer's performance. Thus, we conducted more comprehensive experiments on sequence length.

Effects of sequence length Results in Fig. 10 (Left) demonstrate that SVFormer will always be gradually surpassed by vanilla attention during training while its training speed is faster than vanilla Transformer at the early stage. However, as the training sequence length increases, the SVFormer model performs better. In this way, we focus on the critical point, defined as the number of training steps exceeded. Fig. 10 (Right) illustrates that the relationship between the critical point and sequence length exhibits an exponential trend. We argue that it's due to the challenge deep models face in fully optimizing the increasingly larger first-layer value matrix as the sequence length grows.

Other factors Table 9 show SVFormer benefits more from smaller learning rates than from warmup. This aligns with performance correlating to total summed learning rate (Kaplan et al., 2020). Larger models, requiring smaller learning rates, suit SVFormer better. Results also indicates the SVFormer-Transformer difference is not architecture-sensitive. Compared with Transformer, SVFormer requires a 12.2% increase in parameters to achieve the same loss while reducing the KV-cache by nearly half in Fig. 11.

5 Conclusion

In this paper, we demonstrate the inadequacy of existing hidden residual connections in propagating information from the initial token-level to deeper layers. To address this limitation, we propose ResFormer, which incorporates a residual connection between the value vectors of the current layer and those of the first layer prior to the attention operation. Furthermore, we introduce SVFormer, an extension of ResFormer, which achieves a nearly 50% reduction in the KV cache. We conducted extensive experiments on language modeling tasks to evaluate the efficacy of these two Transformer variants across diverse scenarios.

Limitations

The proposed learnable ResFormer, still falls short of identifying the optimal $\boldsymbol{\lambda}$ setting through current training, instead converging on a relative optimum. This limitation suggests that further refinement of initialization strategies and learning algorithms may be necessary. Due to computational constraints, we were unable to conduct experimental validation on larger-scale models at this time.

Ethics Statement

On the one hand, the data employed in this paper is sourced from publicly available datasets provided by the company, which have undergone a certain level of filtering. On the other hand, the models trained in our study are solely utilized for experimental analysis and will not be publicly deployed.

Acknowledgments

This work was supported by the Scientific Research Project of Westlake University (Grant No. WU2024B003).

References

Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. 2023. Gqa: Training generalized multi-query transformer models from multi-head checkpoints. *arXiv preprint arXiv:2305.13245*.

Stella Biderman, Hailey Schoelkopf, Quentin Gregory Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, USVSN Sai Prashanth, Edward Raff, et al. 2023. Pythia: A suite for analyzing large language models across training and scaling. In *International Conference on Machine Learning*, pages 2397–2430. PMLR.

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. 2020. PIQA: reasoning about physical commonsense in natural language. In *AAAI*, pages 7432–7439. AAAI Press.

William Brandon, Mayank Mishra, Aniruddha Nrusimha, Rameswar Panda, and Jonathan Ragan Kelly. 2024. Reducing transformer key-value cache size with cross-layer attention. *arXiv preprint arXiv:2405.12981*.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *NeurIPS*.

Kevin Clark, Urvashi Khandelwal, Omer Levy, and Christopher D. Manning. 2019. What does BERT look at? an analysis of bert's attention. In *BlackboxNLP@ACL*, pages 276–286. Association for Computational Linguistics.

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. *CoRR*, abs/1803.05457.

Timothée Darcet, Maxime Oquab, Julien Mairal, and Piotr Bojanowski. 2024. Vision transformers need registers. In *International Conference on Learning Representations*. OpenReview.net.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT (1)*, pages 4171–4186. Association for Computational Linguistics.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. 2021. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*. OpenReview.net.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Tianyu Guo, Druv Pai, Yu Bai, Jiantao Jiao, Michael I Jordan, and Song Mei. 2024a. Active-dormant attention heads: Mechanistically demystifying extreme-token phenomena in llms. *arXiv preprint arXiv:2410.13835*.

Zhiyu Guo, Hidetaka Kamigaito, and Taro Watanabe. 2024b. Attention score is not all you need for token importance indicator in kv cache reduction: Value also matters. *arXiv preprint arXiv:2406.12335*.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

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

Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. 2017. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708.

Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q Weinberger. 2016. Deep networks with stochastic depth. In *Computer Vision–ECCV 2016:*

14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part IV 14, pages 646–661. Springer.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de Las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023. Mistral 7b. *CoRR*, abs/2310.06825.

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. *arXiv preprint arXiv:2001.08361*.

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 *International Conference on Learning Representations*. OpenReview.net.

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

Yongyu Mu, Yuzhang Wu, Yuchun Fan, Chenglong Wang, Hengyu Li, Qiaozhi He, Murun Yang, Tong Xiao, and Jingbo Zhu. 2024. Cross-layer attention sharing for large language models. *arXiv preprint arXiv:2408.01890*.

Tam Nguyen, Tan Nguyen, and Richard Baraniuk. 2023. Mitigating over-smoothing in transformers via regularized nonlocal functionals. *Advances in Neural Information Processing Systems*, 36:80233–80256.

Matteo Pagliardini, Amirkeivan Mohtashami, Francois Fleuret, and Martin Jaggi. 2024. Denseformer: Enhancing information flow in transformers via depth weighted averaging. *arXiv preprint arXiv:2402.02622*.

Jackson Petty, Sjoerd Steenkiste, Ishita Dasgupta, Fei Sha, Dan Garrette, and Tal Linzen. 2024. The impact of depth on compositional generalization in transformer language models. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 7232–7245.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2019. Winogrande: An adversarial winograd schema challenge at scale. *Preprint*, arXiv:1907.10641.

Noam Shazeer. 2019. Fast transformer decoding: One write-head is all you need. *CoRR*, abs/1911.02150.

Noam Shazeer. 2020. GLU variants improve transformer. *CoRR*, abs/2002.05202.

Han Shi, Jiahui Gao, Hang Xu, Xiaodan Liang, Zhenguo Li, Lingpeng Kong, Stephen Lee, and James T Kwok. 2022. Revisiting over-smoothing in bert from the perspective of graph. *arXiv preprint arXiv:2202.08625*.

Daria Soboleva, Faisal Al-Khateeb, Robert Myers, Jacob R Steeves, Joel Hestness, and Nolan Dey. 2023. SlimPajama: A 627B token cleaned and deduplicated version of RedPajama.

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

Mingjie Sun, Xinlei Chen, J Zico Kolter, and Zhuang Liu. 2024a. Massive activations in large language models. *arXiv preprint arXiv:2402.17762*.

Qi Sun, Marc Pickett, Aakash Kumar Nain, and Llion Jones. 2024b. Transformer layers as painters. *CoRR*, abs/2407.09298.

Georgy Tyukin, Gbètondji J.-S. Dovonon, Jean Kaddour, and Pasquale Minervini. 2024. Attention is all you need but you don't need all of it for inference of large language models. *CoRR*, abs/2407.15516.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need.(nips), 2017. *arXiv preprint arXiv:1706.03762*, 10:S0140525X16001837.

Peihao Wang, Wenqing Zheng, Tianlong Chen, and Zhangyang Wang. 2022. Anti-oversmoothing in deep vision transformers via the fourier domain analysis: From theory to practice. *arXiv preprint arXiv:2203.05962*.

Tianwen Wei, Liang Zhao, Lichang Zhang, Bo Zhu, Lijie Wang, Haihua Yang, Biye Li, Cheng Cheng, Weiwei Lü, Rui Hu, et al. 2023. Skywork: A more open bilingual foundation model. *arXiv preprint arXiv:2310.19341*.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. 2024. Efficient streaming language models with attention sinks. In *International Conference on Learning Representations*. OpenReview.net.

Sang Michael Xie, Hieu Pham, Xuanyi Dong, Nan Du, Hanxiao Liu, Yifeng Lu, Percy S Liang, Quoc V Le, Tengyu Ma, and Adams Wei Yu. 2024. Doremi: Optimizing data mixtures speeds up language model pretraining. *Advances in Neural Information Processing Systems*, 36.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. Hellaswag: Can a machine really finish your sentence? In *ACL (1)*, pages 4791–4800. Association for Computational Linguistics.

Peiyuan Zhang, Guangtao Zeng, Tianduo Wang, and Wei Lu. 2024. Tinyllama: An open-source small language model. *arXiv preprint arXiv:2401.02385*.

Chunting Zhou, Pengfei Liu, Puxin Xu, Srinivasan Iyer, Jiao Sun, Yuning Mao, Xuezhe Ma, Avia Efrat, Ping Yu, Lili Yu, et al. 2024. Lima: Less is more for alignment. *Advances in Neural Information Processing Systems*, 36.

Daquan Zhou, Bingyi Kang, Xiaojie Jin, Linjie Yang, Xiaochen Lian, Zihang Jiang, Qibin Hou, and Jiashi Feng. 2021. Deepvit: Towards deeper vision transformer. *arXiv preprint arXiv:2103.11886*.

A Appendix

A.1 Attention pattern analysis

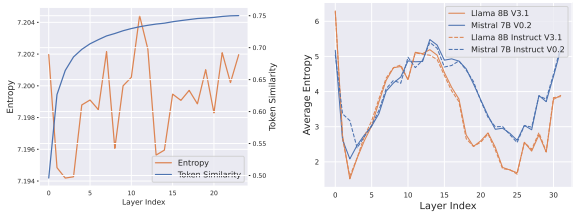
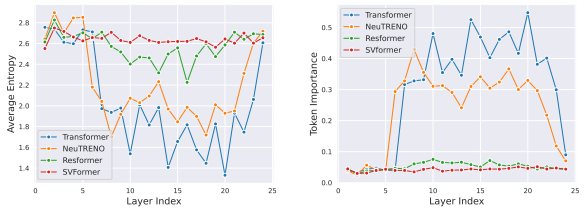


Figure 12: (Left) Average entropy of token importance and the average hidden-state similarity for a randomly initialized 468M model. (Right) Average entropy of token importance across layers in Llama (8B) (Dubey et al., 2024) and Mistral (7B) (Jiang et al., 2023).



(a) Attention Entropy. (b) Token importance.

(c) Norms of value states. (d) Norms of hidden states.

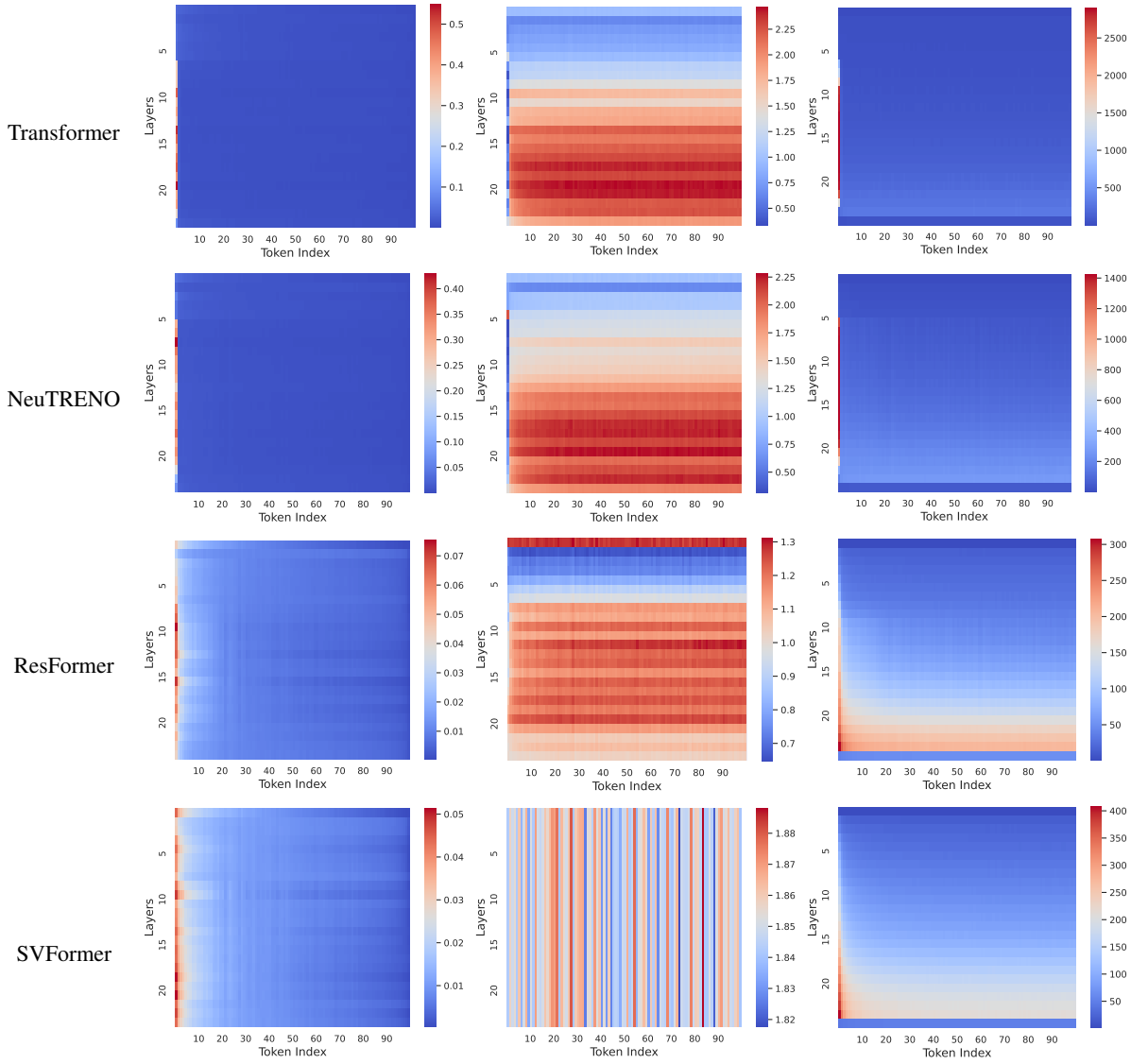
Figure 13: The token importance (Xiao et al., 2024), value-state norms (Guo et al., 2024b), and hidden-state norms (Sun et al., 2024a) of the first token across layers of 468M models. "Attention Entropy" refers to the entropy of token importance across each sequence.

Attention concentration Given the attention matrix $\mathbf{A} \in \mathbb{R}^{l\times l}$ at one layer, we use entropy $\boldsymbol{e}$ to represent its concentration effect. To obtain entropy $\boldsymbol{E}$, calculate the importance vector $\mathbf{a} = \frac{1}{l}\sum_{j=1}^{l} A_{ij}$ firstly where $\mathbf{A}$ is a lower triangular matrix. The entropy can be formulated as follows: $\boldsymbol{e} = -\sum_{i=1}^{l} \boldsymbol{a}'_i \log \boldsymbol{a}'_i$, where $a'_i = a_i / \left(\sum_{i=1}^{l} a_i\right)$ for $i = 1, 2, \ldots, l$ and the higher the entropy $\boldsymbol{e}$, the greater the degree of clustering in $\boldsymbol{a}$, i.e., attention matrix $\mathbf{A}$ is more likely to focus on specific tokens.

The phenomenon of attention concentration is inherent to model architecture and emerges during training. Fig. 12 shows that randomly initialized models exhibit over-smoothing but not attention concentration and popular trained models exhibit obvious attention concentration problem. Trained ViT models often focus on low-informative background areas (Darcet et al., 2024), while language models concentrate on low-semantic tokens (Sun et al., 2024a), particularly the start token (attention sink (Xiao et al., 2024)). While previous studies analyzed single-layer attention patterns, our research reveals a "concentration - dispersion - concentration" pattern in deep models, as shown in Fig. 12 (Right), suggesting potential loss of information during concentrated phases.

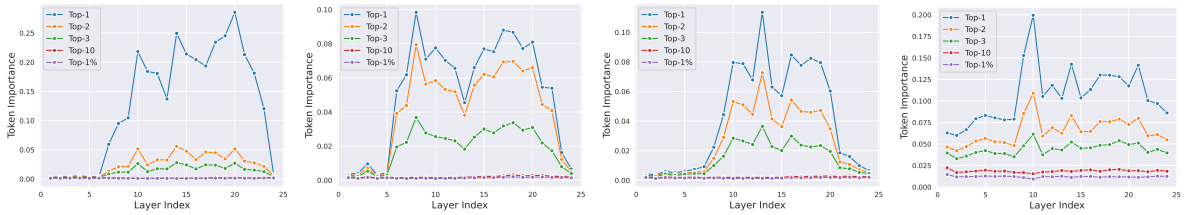
ResFormer alleviates attention concentration Fig. 13a illustrates that the clustering effect of attention increases significantly with the number of layers for the vanilla Transformer, whereas the clustering effect is relatively less pronounced for the ResFormer. We further visualize the attention weights, value-state norms $\|\boldsymbol{v}\|_2$, and hidden-state norms $\|\boldsymbol{h}\|_2$ of tokens at different layers and positions. Given that attention clustering often occurs on the first token, we primarily show its results in Fig. 13. The results indicate that using ResFormer significantly mitigates attention sinks (Xiao et al., 2024), value-state drains (Guo et al., 2024b) and residual-state peaks (Sun et al., 2024a). (Guo et al., 2024a) attributes these phenomena to the mutual reinforcement mechanism of model between value-state drains and attention sinks. We suggest that the value shortcut disrupts this mechanism by alleviating value-state drains due to the absence of value-state drains in the first layer. Specifically, for tokens lacking semantic information like start tokens, a large value state magnitude can adversely affect the prediction of subsequent tokens if they are overly attended to since $\mathbf{U}_n = \mathbf{A}_n\mathbf{V}_n$ in Eqn.1. However, when there is no value-state drains, models will reduce attention clustering to these tokens to minimize loss.

Fig. 14 (First column) demonstrates that the start token easily attracts massive attention despite lack-



(a) Token importance. (b) Value-state norms. (c) Hidden-state drains.

Figure 14: Visualization of token importance, value state norms, and hidden state norms across different token positions and layers in 468M models.



(a) Transformer. (b) NeuTRENO. (c) ResFormer. (d) SVFormer.

Figure 15: The distribution of token importance for different models at different layers.

ing semantic information for Transformer and NeuTRENO. And Fig.15 further illustrates the distribution of token importance, where TOP-i represents the i-th largest token importance within a sequence. Compared to Transformer and NeuTRENO, ResFormer and SVFormer exhibit a more uniform distribution of token importance.

λ value	Layers	Valid loss
-	-	2.739
1	All layers	2.7037
	2,3,4	2.724
	5,6,7	2.698
2	All layers	2.7
	2,3,4	2.728
	5,6,7	2.688
5	All layers	2.704
	2,3,4	2.732
	5,6,7	2.682

Table 10: Performance of Sparse ResFormer with different λ values and different layer configurations.

ResFormer Layers	Norm re-scale	Valid loss
All Layers	No	2.712
	Yes	2.701
2,3,4	No	2.734
	Yes	2.727
6,7,8	No	2.702
	Yes	2.701

Table 11: Ablation study of post-value residual re-scaling.

Any negative effect? Mitigating attention concentration may enhance interpretability but potentially affect transformer sparsity. As attention sinks typically emerge early, applying value residual to later layers should have less impact intuitively. We compared two sparse ResFormer variants on an 8-layer model, applying value residual to layers 2-4 versus 6-8. The results in Table 10 demonstrate that while incorporating value residual generally improves performance compared to vanilla Transformers, increasing λ (the proportion of $\mathbf{V}_1$) in the value residual led to decreased performance for shallower networks. Conversely, deeper networks showed improved results with higher λ values. Notably, the Learnable-ResFormer learned to apply value residual primarily to later layers, minimizing the impact on network sparsity. Moreover, we implemented post-value residual re-scaling ($\mathbf{V}'_n = \frac{\|\mathbf{V}_n\|}{\|0.5\mathbf{V}_1+0.5\mathbf{V}_n\|}(0.5\mathbf{V}_1+0.5\mathbf{V}_n)$) to mitigate its impact on later layers. This approach benefited shallow-sparse ResFormers but had minimal effect on deep-sparse variants. This further suggests that the value residual patterns learned by the Learnable ResFormer do not introduce significant negative effects in this context.

A.2 Pre-train Dataset

Based on the equation $D \geq 5000 \cdot N^{0.74}$ (Kaplan et al., 2020) where D is data size and N is the number of non-embedding parameters, we need to collect at least 17.5B for model has N = 700M non-embedding parameters (corresponding to complete 1B model with 2,048 hidden size, 50,277 vocab size and 2,048 sequence length) to avoid over-fitting. Besides, (Xie et al., 2024) indicates that the mixture proportions of pre-training data domains significantly affects the training results. In this way, we sampled 20B tokens data from original 627B data based on the original data proportions shown in the Table 12.

A.3 Training Details

Section 4.1 introduces the main experimental hyperparameters used in the paper. This section further details the training parameters for various model sizes and training sequence lengths. Table 14 demonstrates the differences among models of various sizes. The configurations for the number of layers, attention heads, hidden dimensions, and FFN dimensions are based on (Biderman et al.,

Data source	proportions	Tokens
Commoncrawl	50%	10 B
C4	20%	4 B
GitHub	10%	2 B
Books	5%	1 B
ArXiv	5%	1 B
Wikpedia	5%	1 B
StackExchange	5%	1 B

Table 12: The details of pre-train dataset.

Max Sequence Length	512	2,048	8,192	32,000	64,000
Total Batch Size	4,096	1,024	256	64	32
Per-GPU Batch Size	128	32	8	2	1
Gradient Accumulation Step			32		
GPUs			8		

Table 13: Training details for training dataset with different sequence length.

2023). Moreover, as reported in Table 13, the batch size that a single GPU can accommodate varies depending on the length of the training sequences. Note that the total number of tokens in each batch is consistently 2 million.

Model Size	2M	82M	180M	468M
Layers	4	8	12	24
Attention Heads	2	8	12	16
Hidden Dimension	16	512	768	1,024
FFN Dimension	56	1,792	2,688	3,584
Tie Word Embedding	False			
(Peak Learning Rate, Final Learning Rate)	$(6e-4, 6e-5)$			
Learning Rate Schedule	Cosine Decay			
Vocabulary Size	50,277			
Activation Function	SwiGLU			
Position Embedding	RoPE (θ = 10,000)			
Batch Size	2M tokens			
Data Size	20B tokens			
(Warmup Steps, Training Steps)	(120, 10,000)			
Adam β	(0.9, 0.95)			
Dropout	0.0			
Weight Decay	0.1			

Table 14: Training details for models with different size.