Transformers with Selective Access to Early Representations

Anonymous authors
Paper under double-blind review

Abstract

Several recent Transformer architectures expose later layers to representations computed in the earliest layers, motivated by the observation that low-level features can become harder to recover as the residual stream is repeatedly transformed through depth. The cheapest among these methods add static value residuals: learned mixing coefficients that expose the first-layer value projection V_1 uniformly across tokens and heads. More expressive dense or dynamic alternatives recover finer-grained access, but at higher memory cost and lower throughput. The usefulness of V_1 is unlikely to be constant across tokens, heads, and contexts; different positions plausibly require different amounts of access to early lexical or semantic information. We therefore treat early-representation reuse as a retrieval problem rather than a connectivity problem, and introduce **Selective Access Transformer (SATFormer)**, which preserves the first-layer value pathway while controlling access with a context-dependent gate. Across models from 130M to 1.3B parameters, SATFormer consistently improves validation loss and zero-shot accuracy over the static value-residual and Transformer baselines. Its strongest gains appear on retrieval-intensive benchmarks, where it improves over static value residuals by approximately 1.5 average points, while maintaining throughput and memory usage close to the baseline Transformer. Gate analyses suggest sparse, depth-dependent, head-specific, and category-sensitive access patterns, supporting the interpretation that SATFormer learns selective reuse of early representations rather than uniform residual copying. Our code is available at https://anonymous.4open.science/r/SATFormer-B09C/README.md.

1 Introduction

Information flow across depth has motivated a broad family of architectural modifications to the Transformer. Although the residual stream provides a direct pathway for propagating token representations, each layer transforms those representations through attention, normalization, and feed-forward computation. Prior work has argued that low-level lexical, structural, and token-specific features computed in early layers can become difficult to recover deeper in the network (Zhou et al., 2025). A natural response is to expose later layers to earlier representations, specifically the first-layer value projection V_1, through additional cross-layer pathways. This idea underlies work on residual scaling and normalization, gated residuals, dense layer aggregation, and dynamic cross-layer routing (Bachlechner et al., 2021; Touvron et al., 2021; Wang et al., 2024; Srivastava et al., 2015; Pagliardini et al., 2024; Xiao et al., 2025; Zhu et al., 2024).

These methods largely share an underlying framing: cross-layer reuse is a connectivity problem, where the design question is how many pathways to add between layers and how richly to combine them. The simplest instantiation, value residual learning (ResFormer) (Zhou et al., 2025), exposes V_1 to all subsequent layers through a learned scalar coefficient per layer. This mechanism is computationally cheap, but it applies the same amount of V_1 to every token and every attention head within a layer. More expressive methods such as DenseFormer, MUDDFormer, and HyperConnections recover finer-grained access by aggregating over wider layer histories or generating dynamic dense pathways, but these gains come with higher memory cost and lower throughput (Pagliardini et al., 2024; Xiao et al., 2025; Zhu et al., 2024).

This leaves a more targeted question: can a single early-value pathway become competitive if the model learns *control* over when and where that pathway is used? The usefulness of V_1 should vary across tokens, heads, layers, and contexts: a token that depends on prompt-specific lexical or semantic information may

benefit from direct access to early features, while another token or head may not need that access at all. A layer-wise scalar cannot express this variation, because it exposes or suppresses V_1 broadly across the layer.

To put this into practice, we introduce **Selective Access Transformer** (**SATFormer**), in which a computationally cheap per-token, per-head gate modulates how much of V_1 enters each layer's value computation. Empirically, this control-based design yields favorable performance-efficiency trade-offs. Across 130M-1.3B parameter models, SATFormer consistently improves validation loss and zero-shot accuracy over the baselines. The gains are particularly visible on retrieval-intensive evaluations, where SATFormer outperforms ResFormer by nearly 1.5 average points. At the same time, SATFormer maintains throughput and memory usage close to the Transformer and ResFormer baselines, while improving on their perplexity at all scales. We further analyze the learned gating behavior to understand how SATFormer uses its retrieval pathway.

Our contributions are as follows:

- **A control-based framing for early-representation reuse.** We treat V_1 access as a question of control across tokens, heads, and depth over a single early pathway, rather than connectivity over many pathways.
- **The SATFormer architecture.** SATFormer replaces ResFormer's per-layer scalar with a per-token, per-head gate from a single linear projection of the current hidden state, adding negligible parameters and runtime cost.
- **Pareto-favorable scaling and retrieval.** SATFormer improves validation loss over Transformer and ResFormer at every evaluated scale (130M–1.3B), with the largest zero-shot gain over ResFormer at 1.3B (+0.67 average accuracy). On retrieval-intensive benchmarks, it gains approximately 1.5 points over ResFormer and matches or exceeds dense alternatives such as MUDDFormer at roughly 1.8× their throughput.
- **Mechanistic evidence for structured access.** Gate analyses reveal sparse, depth-dependent, head-specific, and token-specific access, concentrated in a subset of heads. Ablation and logit-lens experiments confirm that both the V_1 pathway and the input-dependent structure of the gate are functionally consequential to language modeling performance.

2 Related Work

Prior work improves depth-wise information flow by changing either the stability of the residual stream or the amount of cross-layer history available to later computation. Residual scaling and normalization methods such as ReZero (Bachlechner et al., 2021), LayerScale (Touvron et al., 2021), and DeepNorm (Wang et al., 2024) improve optimization and signal propagation without adding explicit access to a large layer history. Gated residual mechanisms such as Highway networks (Srivastava et al., 2015) similarly introduce control over residual updates. A second line of work increases cross-layer access more directly: multi-stream recurrence methods such as HyperConnections (Zhu et al., 2024) and mHC (Xie et al., 2025) expand the residual stream into multiple pathways, while aggregation and routing methods such as DenseNet-style connectivity (Huang et al., 2017), ELMo-style learned layer mixtures (Peters et al., 2018), DenseFormer (Pagliardini et al., 2024), MUDDFormer (Xiao et al., 2025), MRLA (Fang et al., 2023), LAuReL (Menghani et al., 2024), and Attention Residuals (Team et al., 2026) combine information from earlier layers more explicitly. These methods trace an efficiency–granularity frontier: static scalar connections are inexpensive but coarse, whereas dense aggregation and dynamic routing provide finer-grained access at higher memory cost and lower throughput. SATFormer targets a different point on this frontier: adaptive control over a single early-value pathway, rather than denser access to many previous representations.

The closest prior work to SATFormer is value residual learning (ResFormer) introduced by (Zhou et al., 2025). ResFormer addresses the dilution of initial token information by adding a direct connection from the first-layer value projection to the value vectors of subsequent layers. For a Transformer layer $n > 1$, the modified value vector is computed as

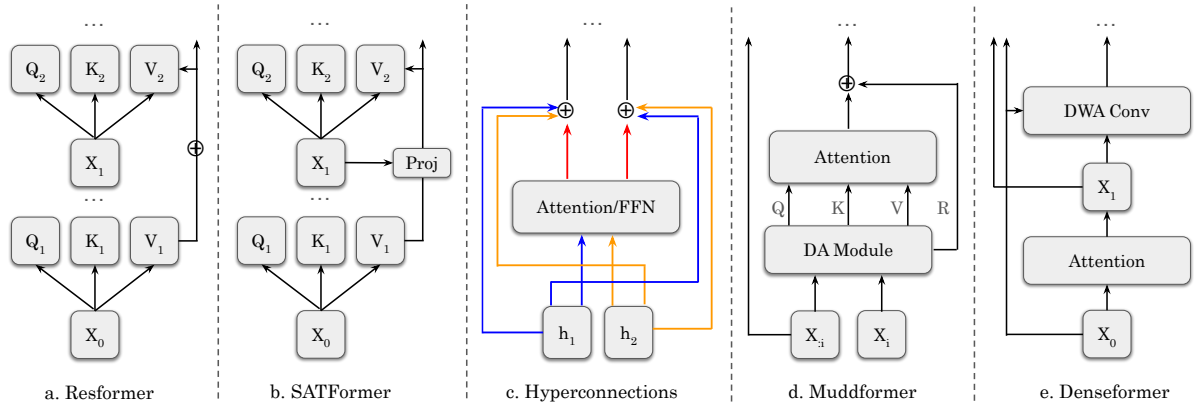

Figure 1: ResFormer provides efficient but layer-wise static access to V_1; DenseFormer, MUDDFormer, and HyperConnections increase cross-layer access through denser or more dynamic pathways at substantially higher cost. SATFormer preserves the single early-value pathway of ResFormer but makes access token- and head-dependent through a computationally cheap gate, achieving greater adaptability without increasing cross-layer connectivity.

$$V_n' = \lambda_{n_1} V_1 + \lambda_{n_2} V_n \tag{1}$$

where V_1 is the value representation from the first layer, λ_{n_1} is typically a learned layer-wise scalar initialized to $\lambda_{\text{scale}} \cdot \frac{e^{\lambda'_{n,1}}}{\sum_{j=1}^{N} e^{\lambda'_{j,1}}}$ where λ_{scale} is initialized to the total layers N and λ_{n_2} is held constant across layers. Note, in the definition above we refer to the highest performing ResFormer variant, Learnable ResFormer Plus.

We focus our empirical comparisons on four methods that instantiate distinct points along the efficiency–granularity frontier. Value residual learning (Zhou et al., 2025) addresses the dilution of initial token information by adding a direct residual connection to the value vectors of all subsequent layers, using learned layer-wise scalar coefficients. HyperConnections (Zhu et al., 2024) provides a learnable alternative to traditional residuals by replicating hidden states into multiple pathways. MUDDFormer (Xiao et al., 2025) extends this to the most flexible setting, generating dense connection weights dynamically for each sequence position via a lightweight MLP. DenseFormer (Pagliardini et al., 2024) introduces a Depth-Weighted-Average module after each Transformer block, computing a weighted average over all preceding representations.

SATFormer targets a narrower and more controlled design point. Rather than aggregating over all previous layers, expanding the residual stream, or routing across a dense layer history, it asks whether a single early-value pathway can become competitive when access to that pathway is made input-dependent. SATFormer does not increase the amount of cross-layer history available to the model; it increases the model's control over when and where one early representation is reused.

3 Methodology

Previous work exposes V_1 through layer-wise scalar coefficients, making early-value access uniform across tokens and heads within a layer (Zhou et al., 2025). To enable more dynamic input-aware gating, we replace the static scalar λ_n in Equation 1 with an input-dependent mixing coefficient. This allows the model to perform *context-aware value retrieval*. Let $x_t^{(n)} \in \mathbb{R}^{d_{\text{model}}}$ denote the normalized hidden state for the token at position t in layer n. SATFormer computes a per-token, per-KV-head gate $\alpha_{t,j}^{(n)}$ as

$$\alpha_{t,j}^{(n)} = \left[\mathrm{ReLU}\left(x_t^{(n)} W_\alpha^{(n)}\right)\right]_j, \tag{2}$$

where $W_\alpha^{(n)} \in \mathbb{R}^{d_{\text{model}} \times N_{kv}}$ is a single linear projection, N_{kv} is the number of key-value heads, and $j \in \{1, \ldots, N_{kv}\}$ indexes the KV head. The value used by attention is then

$$V'_{t,j,r} = V_{t,j,r} + \alpha_{t,j}^{(n)} \cdot V_{t,j,r}^{(1)}, \tag{3}$$

where r indexes the per-head value dimension. The ReLU activation keeps the mixing coefficient non-negative and allows the gate to become exactly zero. A deeper comparison of gating mechanisms can be found in Appendix B.1.

Design rationale. SATFormer makes three deliberately minimal choices. **First**, the gate is per-token and per-head because the usefulness of V_1 varies along both axes: a token-level gate alone would ignore head-level specialization, while a head-level gate alone would make access uniform across the sequence, preventing the model from selecting particular tokens whose early representations should be preserved. **Second**, the gate is computed with a single linear projection from the current normalized hidden state. This makes access depend on the representation already available to the layer, while adding only $d_{\text{model}} N_{\text{kv}}$ parameters per layer. **Third**, we use a ReLU gate so that access is non-negative and can be exactly zero. Non-negativity keeps the pathway interpretable as additive reuse of the first-layer value stream, while exact zeros allow the model to fully disengage from V_1 when early information is unhelpful. Further ablation studies regarding gating functions can be found in Appendix B.1. This construction preserves the same early-value source as ResFormer, but changes the access policy: rather than applying a single layer-wise coefficient, SATFormer lets each token and KV head choose how much of V_1 to reuse.

4 Experiments

We organize our evaluation around the central claim of this paper: selective access to early value representations provides an efficient alternative to dense cross-layer connectivity, with particular benefits for in-context retrieval and language-modeling performance. We therefore begin with retrieval-intensive benchmarks, where selective access should be most directly useful. We then evaluate wall-clock efficiency, throughput, and memory usage to test whether SATFormer preserves the computational profile of sparse connectivity. Finally, we report standard language modeling, zero-shot, and scaling results to verify that these targeted gains do not come at the cost of model quality.

Pretraining setup. Along with SATFormer, our experiments include Transformer, ResFormer, DenseFormer, MUDDFormer, and HyperConnections as baselines. We train all models under matched data, optimizer, and hardware conditions across three main scales: Small (130M parameters, 5B tokens, $\eta = 10^{-3}$), Medium (340M parameters, 10B tokens, $\eta = 1.5 \times 10^{-3}$), and Large (760M parameters, 20B tokens, $\eta = 2 \times 10^{-3}$). We additionally train Transformer, ResFormer, and SATFormer at the XL scale (1.3B parameters, 30B tokens) to study scaling relative to the closest static value-residual baseline. All models are trained on 2 NVIDIA H200 GPUs, with configurations closely following Yang et al. (2024).

We train on a randomly sampled subset of FineWeb-Edu and use the Llama-2 tokenizer with a vocabulary size of 32,000. Training uses a sequence length of 4K tokens and the AdamW optimizer with weight decay 0.1 and gradient clipping 1.0. The learning rate η follows a cosine annealing schedule, decaying to a minimum of $0.1 \times \eta_{base}$ after a linear warmup period covering the first 1% of training tokens.

Model configurations. Table 1 reports the model configurations used at each scale. For each baseline, we use the strongest or recommended variant from the corresponding prior work. DenseFormer uses a dilation factor of 4 and a Depth-Weighted-Average (DWA) period of 5. MUDDFormer uses full Query, Key, Value, and Residual connectivity with parameter reallocation and pre/post Depth Aggregation normalization. HyperConnections uses the dynamic connection variant with expansion rate $n = 2$, tanh activation, and layer normalization. ResFormer follows the Learnable ResFormer Plus variant defined by the weighted residual sum in Equation 1.

4.1 Retrieval-Intensive Evaluation

We first evaluate SATFormer on retrieval-intensive benchmarks, where preserving prompt-specific information should be especially useful. As shown in Table 2, SATFormer achieves the highest average score among the evaluated architectures, narrowly surpassing MUDDFormer and improving over ResFormer by approximately 1.5 average points.

Table 1: **Model Configurations.**

Variant	d_{model}	L	n_{heads}	d_{ff}
Small	768	11	12	3072
Medium	1024	18	16	4096
Large	1536	19	24	6144
XL	2048	24	32	5632

This comparison is particularly informative because SATFormer and ResFormer both expose the first-layer value stream. Their main difference is that ResFormer mixes this stream using layer-wise static coefficients, whereas SATFormer conditions access on the current hidden state at the token and head level. The observed improvement therefore suggests that the benefit is not simply due to making V_1 available, but may also depend on controlling when and where the early representation is reused.

Table 2: **Performance on retrieval-intensive tasks. Input length was truncated to 2k tokens during evaluation. Evaluation protocol follows Arora et al. (2024).**

Model	TriviaQA	SWDE	SQUAD	NQ	FDA	Drop	Avg
Transformer	52.9	29.43	34.91	12.06	19.14	20.07	28.085
DenseFormer	52.66	32.31	35.32	11.24	16.42	22.66	28.435
MUDDFormer	54.32	**36.81**	34.78	12.06	20.32	20.84	29.855
HyperConnections	**55.27**	25.83	36.72	**12.82**	15.78	**24.86**	28.546
ResFormer	53.9	27.9	34.88	11.46	19.41	22.8	28.392
SATFormer	54.26	32.4	**36.96**	12.16	**21.96**	21.41	**29.858**

Importantly, SATFormer achieves this retrieval performance while preserving a computational profile close to the baseline Transformer and ResFormer. In contrast, MUDDFormer attains a similar retrieval average but trains substantially more slowly under matched hardware. These results are consistent with the hypothesis that selective early-value access is useful in settings where performance depends on retaining prompt-specific lexical or semantic information, while avoiding the higher cost of dense cross-layer connectivity.

4.2 Efficiency Benchmarking

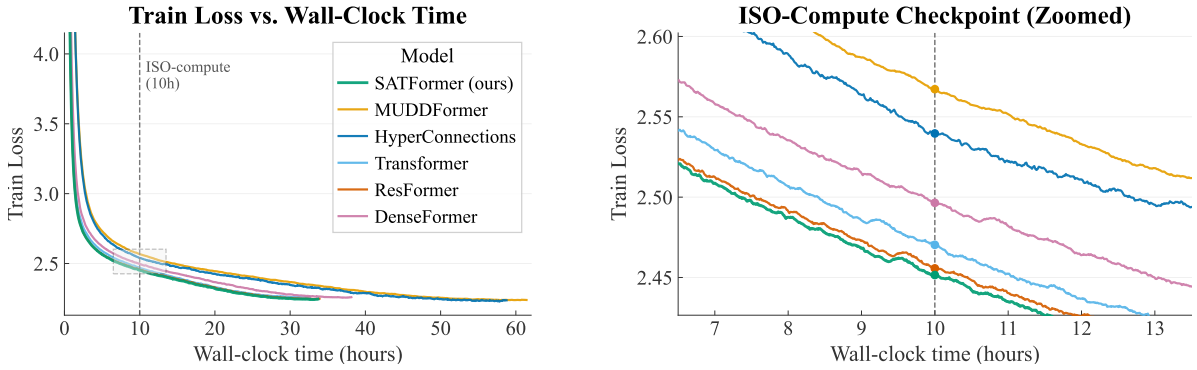

Figure 2: **(Left)** Train loss plotted against wall-clock time in hours. The dashed vertical line indicates the 10-hour wall-clock checkpoint used for comparison. **(Right)** A zoomed-in view of the 10-hour checkpoint highlighting the loss differentials. Metrics computed from the 760M training run.

Final validation loss and zero-shot accuracy do not fully capture the practical cost of dense cross-layer connectivity. For instance, architectures that expand the residual stream, aggregate over many previous layers, or generate dynamic dense pathways may improve endpoint performance while making substantially slower wall-clock progress. We therefore evaluate training efficiency directly, focusing on wall-clock loss, throughput, and memory usage under matched hardware.

Figure 2 compares training loss as a function of wall-clock time. At the 10-hour checkpoint, SATFormer achieves a loss of 2.4515 compared to 2.4557 for ResFormer, 2.4703 for the baseline Transformer, and 2.4965 for DenseFormer. In contrast, HyperConnections and MUDDFormer make substantially slower wall-clock progress in this setup, reaching losses of 2.5396 and 2.5672 at the same checkpoint.

Table 3 reports memory usage and throughput at the 760M scale with a batch size 16, 4 gradient accumulation steps, sequence length 4096, 2 NVIDIA H200 GPUs, and 5 warmup steps. SATFormer maintains throughput close to the baseline Transformer and ResFormer while using comparable memory. In terms of efficiency, SATFormer achieves roughly 1.75× higher throughput and uses 31% less memory than HyperConnections, 1.82× higher throughput and 5% less memory than MUDDFormer, and 4% less memory than ResFormer. Thus, while dense dynamic methods and static value residuals can achieve strong endpoint performance, SATFormer provides a more favorable wall-clock Pareto trade-off in throughput-constrained training regimes.

Table 3: **Training Throughput (toks/sec) & Memory Usage (GB).** Metrics computed at the 760M scale.

Model	Memory	Throughput
Transformer	85.76	170,308
DenseFormer	93.32	145,639
MUDDFormer	90.52	90,443
HyperConnections	124.73	94,534
ResFormer	89.59	163,826
SATFormer	86.03	164,839

4.3 General Language Modeling and Zero-Shot Evaluation

We next evaluate whether SATFormer's retrieval and efficiency gains come at the cost of general language-modeling quality. Table 4 reports final validation loss, while Table 5 reports zero-shot performance across language modeling and downstream benchmarks. These broader evaluations serve as a general-purpose validation of SATFormer beyond the targeted retrieval and efficiency settings. We first focus on the comparison between SATFormer and ResFormer, since this isolates the effect of replacing static value-residual access with input-dependent selective access. SATFormer improves final validation loss over ResFormer at every evaluated scale: 2.648 vs. 2.674 at 130M, 2.415 vs. 2.429 at 340M, 2.242 vs. 2.246 at 760M, and 2.139 vs. 2.150 at 1.3B. This suggests that selective access to V_1 improves the training objective relative to static value mixing without requiring dense connectivity. This is the central efficiency advantage of selective access: it increases the granularity of cross-layer reuse without substantially increasing the amount of cross-layer connectivity.

Table 4: **Final Validation Loss Values.**[1]

Variant	TF	DF	MF	HC	RF	SAT
Small	2.707	2.704	2.658	2.670	2.674	**2.648**
Medium	2.461	2.452	2.415	**2.410**	2.429	2.415
Large	2.256	2.256	**2.233**	2.234	2.246	2.242
XL	2.152	N/A	N/A	N/A	2.150	**2.139**

One result warrants direct acknowledgment: while SATFormer improves over ResFormer at every evaluated scale, denser alternatives still achieve higher endpoint zero-shot accuracy in some settings. At the Medium scale, HyperConnections outperforms SATFormer in average zero-shot accuracy (46.04 vs. 45.81), and at the Large scale, MUDDFormer and HyperConnections achieve higher average scores (49.67 vs. 48.81). However, SATFormer achieves roughly $1.75 - 1.82\times$ higher throughput than these denser alternatives in our efficiency benchmark. This is consistent with SATFormer's design intent: it is not meant to uniformly dominate all dense connectivity mechanisms on endpoint accuracy, but to provide a favorable Pareto trade-off between adaptivity, model quality, and training efficiency.

Scaling behavior. The scaling results on core language modeling suggest that SATFormer's benefits may depend not only on parameter count, but also on the number of opportunities the model has to reuse V_1 across depth. Notably, the XL setting (1.3B parameters) shows SATFormer's strongest zero-shot gains over ResFormer (+0.67 average accuracy), consistent with deeper models providing more layers at which selective V_1 access can be applied. This interpretation is further supported by the gate analysis in Section 5, where V_1

[1]TF: Transformer, DF: DenseFormer, MF: MUDDFormer, HC: HyperConnections, RF: ResFormer, SAT: SATFormer.

Table 5: **Zero-shot performance across language modeling and downstream tasks.** Best results are **bolded**. Task abbreviations: Wiki (WikiText-103), LMB (Lambada), PIQA (Physical IQA), Hella. (HellaSwag), Wino. (Winogrande), ARC-e/c (ARC Easy/Challenge), SIQA (Social IQA).

Model	Wiki. PPL ↓	LMB. PPL ↓	LMB. ACC ↑	PIQA ACC ↑	Hella. ACC_N ↑	Wino. ACC ↑	ARC-e ACC ↑	ARC-c ACC_N ↑	SIQA ACC ↑	BoolQ ACC ↑	Avg. ↑
Small: 130M params / 5B tokens											
Transformer	41.81	121.31	22.26	61.21	30.57	51.85	46.68	25.51	37.00	**62.08**	42.15
DenseFormer	41.80	134.53	21.62	61.32	30.90	50.43	49.12	23.89	35.93	60.37	41.70
MUDDFormer	39.45	115.12	21.70	60.88	31.48	**52.49**	48.70	23.81	36.28	57.92	41.66
HyperConnections	38.45	95.54	23.71	61.31	31.63	50.98	48.44	**25.68**	**37.25**	55.41	41.80
ResFormer	39.68	106.80	23.38	60.07	31.00	51.30	48.53	23.89	35.93	61.41	41.94
SATFormer	**37.98**	**84.23**	**25.79**	**61.97**	**32.12**	50.91	**49.54**	25.00	36.49	59.88	**42.71**
Medium: 340M params / 10B tokens											
Transformer	29.46	59.29	27.38	65.07	36.71	50.83	52.95	24.57	38.02	62.14	44.71
DenseFormer	28.92	49.68	29.17	65.40	37.01	48.15	54.42	25.68	37.87	61.01	44.84
MUDDFormer	27.97	49.44	27.77	65.72	37.04	51.07	54.17	26.11	37.15	59.33	44.80
HyperConnections	**26.83**	**35.21**	**33.86**	64.85	**38.72**	51.06	54.54	26.70	36.74	**61.89**	**46.04**
ResFormer	27.84	42.38	30.47	64.74	38.03	**51.46**	55.85	25.68	38.18	60.55	45.62
SATFormer	26.98	45.84	28.90	65.89	38.30	51.07	56.27	27.22	37.36	61.44	45.81
Large: 760M params / 20B tokens											
Transformer	21.95	24.27	36.72	67.14	44.20	50.75	61.11	30.89	37.92	60.61	48.67
DenseFormer	21.77	24.57	37.16	68.17	44.28	52.17	61.70	29.52	39.15	55.35	48.44
MUDDFormer	**21.07**	22.30	38.11	68.28	45.53	55.64	61.74	**30.80**	38.84	58.38	**49.67**
HyperConnections	21.14	**21.63**	**38.36**	67.84	**45.93**	**52.95**	60.90	30.03	**39.25**	**62.17**	**49.67**
ResFormer	21.71	25.06	37.03	66.87	44.81	52.64	59.97	29.61	**39.25**	60.15	48.79
SATFormer	21.24	24.92	36.72	**68.34**	45.25	50.67	**61.91**	29.86	38.79	58.96	48.81
XL: 1.3B params / 30B tokens											
Transformer	18.83	18.66	40.38	69.85	50.40	**54.85**	65.61	32.93	39.71	56.17	51.23
ResFormer	18.62	18.28	40.55	**69.36**	50.02	52.88	65.48	**33.87**	39.66	58.53	51.29
SATFormer	**18.47**	**16.44**	**43.56**	68.49	**51.01**	54.30	**65.69**	32.59	**40.63**	**59.48**	**51.96**

utilization remains sparse in early layers and increases substantially in deeper layers, and by the additional 760M thin-and-long study in Appendix B.2, which shows that reallocating capacity toward depth improves SATFormer more than ResFormer. These results suggest that the comparatively muted 760M endpoint result is at least partly configuration-dependent, and that selective access benefits from depth as well as scale. A more complete scaling study separating the effects of depth, width, and training tokens remains an important direction for future work.

5 Mechanistic Analysis

The goal of this section is to test a narrower mechanistic prediction of the architecture: if selective access is doing useful work, the learned gate should not behave like a dense residual shortcut. Instead, access to V_1 should be sparse, concentrated in particular heads or layers, and functionally important when perturbed. We focus on three questions: First, is access to V_1 sparse or broadly active? Second, which layers are most functionally sensitive to perturbing the learned gate, and how does this compare to where gate activation is largest? Third, do different token categories receive different amounts of access?

5.1 Sparse and Depth-Dependent Access

Figure 3 shows the mean gate activation across layers and KV heads. The learned access pattern is highly non-uniform. In the early and middle layers, most heads assign little or no weight to V_1, indicating that SATFormer does not simply propagate the first-layer value stream through the network as a dense residual shortcut. Instead, access remains sparse for much of the model depth.

This pattern rules out the simplest interpretation of SATFormer as merely learning a rescaled value residual. If the gate were only compensating for the absence of ResFormer's layer-wise coefficient, access would be distributed broadly across heads within each layer. Instead, the model allocates V_1 access sparsely: most heads remain close to inactive, while a smaller set of later-layer heads carries most of the early-value reuse. Thus, SATFormer learns a division of labor across heads and depth, where the early-value pathway is reserved for specialized computations rather than used as a uniform shortcut.

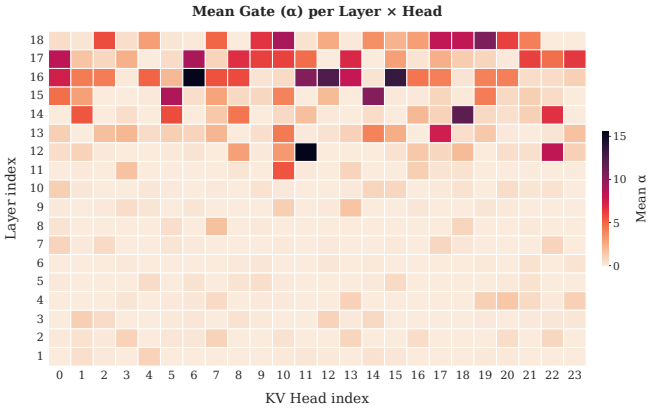

Figure 3: Mean gate activation across layers and KV heads. Early layers show less access to V_1, while some later-layer heads activate sharply, indicating that SATFormer uses the early-value pathway selectively rather than uniformly.

5.2 Gate Intervention Analysis

As shown in Figure 4, both interventions degrade performance relative to the unablated SATFormer. Zeroing the gate tests whether access to V_1 matters at all, while replacing the gate with its mean preserves the average scale of access but removes token- and head-dependent selectivity. The fact that both interventions increase perplexity shows that SATFormer benefits from both components: the early-value pathway itself and the learned structure controlling when that pathway is used.

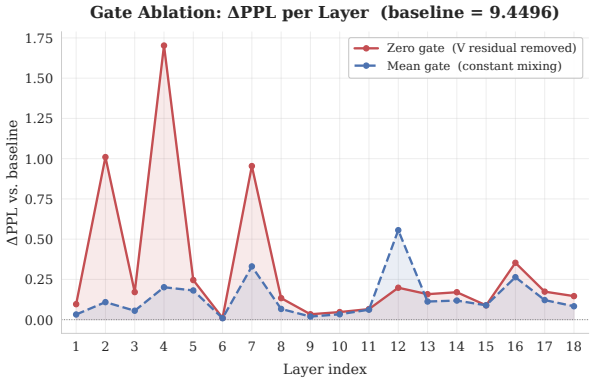

Figure 4: We measure the change in perplexity after either zeroing the gate at a single layer or replacing the input-dependent gate with its mean value. Ablating some earlier layers produces larger perplexity increases.

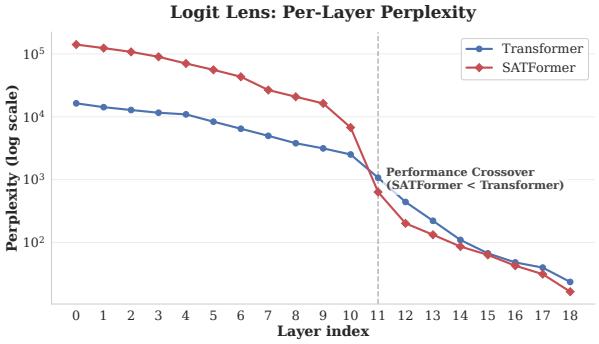

Figure 5: SATFormer begins to outperform the baseline Transformer in layer-wise perplexity near the same depth range where V_1 gate activation increases.

The intervention results also show that gate magnitude alone is not a complete measure of functional importance. Later layers have the largest average gate activations, but some earlier layers produce large perplexity increases when perturbed. This indicates that small early uses of V_1 can affect representations that are amplified or reused later in the network. The logit-lens probe in Figure 5 is consistent with this picture: SATFormer initially lags the baseline Transformer, then crosses over in the same later-depth regime where gate activation becomes stronger. Together, the ablation and logit-lens results show that selective access is not only visible in the gate statistics, but also consequential for model predictions.

5.3 Category-Specific Access

Finally, we ask whether the gate treats different kinds of early features differently. Because V_1 itself is high-dimensional and not directly labeled, we use sparse autoencoders as a coarse labeling tool: SAE features

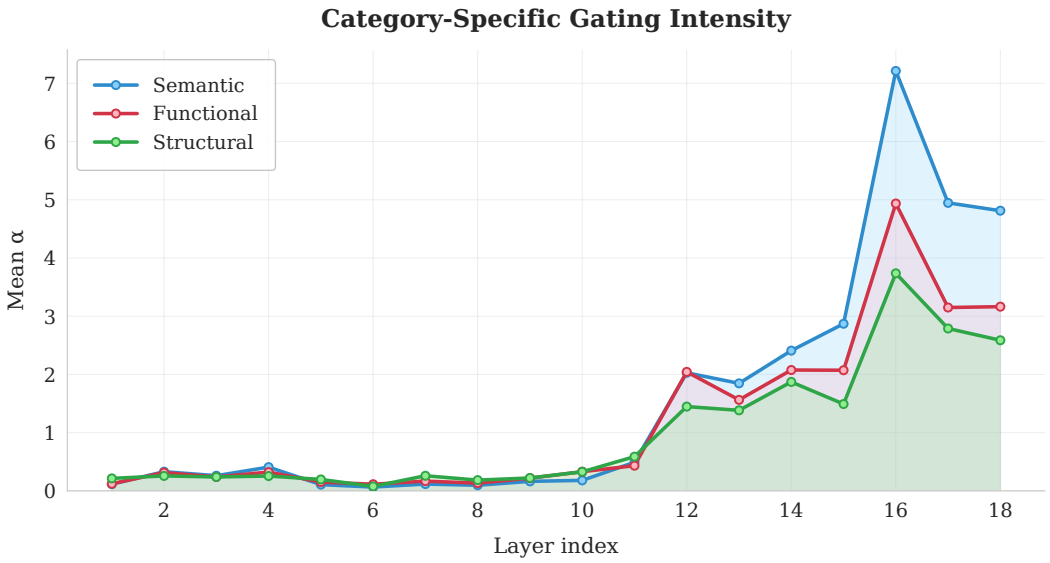

Figure 6: Gating is nearly uniform in the first half of the model, where access to V_1 is generally weak. In later layers, semantic features receive stronger V_1 access than structural and functional features, suggesting that SATFormer preferentially re-accesses content-bearing early representations.

with similar top-activating tokens are grouped into broad semantic, functional, and structural categories. This analysis should be read as a descriptive probe of which feature types receive more access, not as a causal decomposition of SATFormer's downstream gains. Details of the SAE training and labeling pipeline are given in Appendix C.

Figure 6 shows that category differences are small in early layers, where access to V_1 is weak overall. In later layers, however, features labeled as semantic receive higher average gate activation than structural or functional features. This provides a more interpretable view of the sparse late-layer access pattern from Figure 3: the heads that re-access V_1 are not only active late in depth, but also tend to assign more access to content-bearing early features. This does not establish that semantic gating causes the retrieval gains in Table 2, but it supports the intended picture of SATFormer as selectively reusing early token information rather than uniformly copying the first-layer value stream.

Across these analyses, SATFormer's learned gate behaves as a selective and functionally consequential access mechanism. Access to V_1 is sparse in early layers and concentrated in a subset of later-layer heads; per-layer ablations show that disrupting the gate increases perplexity, with especially large effects in some earlier layers; logit-lens analysis shows that SATFormer begins to outperform the baseline Transformer near the later-layer gating transition; and category-level analysis shows stronger access for semantic than structural features. These findings support the interpretation of SATFormer as a computationally cheap mechanism for targeted early-representation reuse; however, we do not claim full causality and leave fine-grained decomposition as future work.

6 Conclusion

We introduced SATFormer, a computationally cheap interpretation of value residual learning that replaces static layer-wise access to the first-layer value stream with token- and head-dependent gating. Across 130M to 1.3B parameter models, SATFormer improves validation loss over ResFormer at every evaluated scale and improves average zero-shot accuracy over ResFormer at 130M, 340M, and 1.3B, including a +0.67 point gain at 1.3B. On retrieval-intensive benchmarks, SATFormer achieves the highest average score among all evaluated architectures (29.86 average across TriviaQA, SWDE, SQuAD, NQ, FDA, and DROP), outperforming ResFormer by roughly 1.5 average points and narrowly surpassing MUDDFormer at a fraction

of the throughput cost. At the same time, SATFormer maintains throughput and memory usage close to the baseline Transformer and ResFormer, achieving roughly 1.75× higher throughput than HyperConnections and using 31% less memory.

These results support a targeted but important conclusion: early representations need not be made available through dense connectivity to be useful. SATFormer occupies a useful point in the architecture design space: substantially cheaper than dense cross-layer routing, more adaptive than static value residuals, and especially effective in retrieval-oriented settings. A single early-value pathway, when equipped with token- and head-dependent control, can provide an efficient mechanism for targeted reuse. SATFormer therefore reframes cross-layer reuse as a problem of selective access rather than maximal connectivity, offering a simple and computationally efficient design point for future Transformer architectures.

Broader Impacts and Safeguards

This work studies architectural mechanisms for improving information flow in Transformer language models. Our experiments are conducted at relatively small research scales, and the trained models are intended for controlled empirical analysis rather than deployment. As such, the models evaluated in this work are unlikely to directly pose substantial real-world risks on their own.

At the same time, SATFormer is a general-purpose architectural modification and could be applied to larger language models. Improvements in language-modeling efficiency or performance may therefore contribute to both beneficial and harmful downstream capabilities, depending on how such models are trained, released, and used. Potential risks include misuse of stronger language models for deception, spam, harmful content generation, or other malicious applications. We therefore emphasize that this work should be used in accordance with standard safeguards for language-model development, including careful dataset curation, capability and misuse evaluations, controlled release practices when appropriate, and monitoring for downstream harms.

The intended benefit of this research is to improve the efficiency and interpretability of Transformer architectures. By studying selective access to early representations, we aim to better understand how models reuse information across depth and to identify architectural designs that improve performance without requiring substantially denser or more computationally expensive connectivity.

References

S. Arora, A. Timalsina, A. Singhal, B. Spector, S. Eyuboglu, X. Zhao, A. Rao, A. Rudra, and C. Ré. Just read twice: closing the recall gap for recurrent language models. *arXiv preprint arXiv:2407.05483*, 2024.

T. Bachlechner, B. P. Majumder, H. Mao, G. Cottrell, and J. McAuley. Rezero is all you need: Fast convergence at large depth. In *Uncertainty in artificial intelligence*, pages 1352–1361. PMLR, 2021.

H. Cunningham, A. Ewart, L. Riggs, R. Huben, and L. Sharkey. Sparse autoencoders find highly interpretable features in language models. *arXiv preprint arXiv:2309.08600*, 2023.

Y. Fang, Y. Cai, J. Chen, J. Zhao, G. Tian, and G. Li. Cross-layer retrospective retrieving via layer attention. *arXiv preprint arXiv:2302.03985*, 2023.

G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

W. Kwon, Z. Li, S. Zhuang, Y. Sheng, L. Zheng, C. H. Yu, J. Gonzalez, H. Zhang, and I. Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th symposium on operating systems principles*, pages 611–626, 2023.

G. Menghani, R. Kumar, and S. Kumar. Laurel: Learned augmented residual layer. *arXiv preprint arXiv:2411.07501*, 2024.

M. Pagliardini, A. Mohtashami, F. Fleuret, and M. Jaggi. Denseformer: Enhancing information flow in transformers via depth weighted averaging. *Advances in neural information processing systems*, 37: 136479–136508, 2024.

M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer. Deep contextualized word representations. In M. Walker, H. Ji, and A. Stent, editors, *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2227–2237, New Orleans, Louisiana, June 2018. Association for Computational Linguistics. doi: 10.18653/v1/N18-1202. URL https://aclanthology.org/N18-1202/.

R. K. Srivastava, K. Greff, and J. Schmidhuber. Highway networks. *arXiv preprint arXiv:1505.00387*, 2015.

K. Team, G. Chen, Y. Zhang, J. Su, W. Xu, S. Pan, Y. Wang, Y. Wang, G. Chen, B. Yin, et al. Attention residuals. *arXiv preprint arXiv:2603.15031*, 2026.

H. Touvron, M. Cord, A. Sablayrolles, G. Synnaeve, and H. Jégou. Going deeper with image transformers. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 32–42, 2021.

H. Wang, S. Ma, L. Dong, S. Huang, D. Zhang, and F. Wei. Deepnet: Scaling transformers to 1,000 layers. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 46(10):6761–6774, 2024.

D. Xiao, Q. Meng, S. Li, and X. Yuan. Muddformer: Breaking residual bottlenecks in transformers via multiway dynamic dense connections. *arXiv preprint arXiv:2502.12170*, 2025.

Z. Xie, Y. Wei, H. Cao, C. Zhao, C. Deng, J. Li, D. Dai, H. Gao, J. Chang, K. Yu, et al. mhc: Manifold-constrained hyper-connections. *arXiv preprint arXiv:2512.24880*, 2025.

A. Yang, A. Li, B. Yang, B. Zhang, B. Hui, B. Zheng, B. Yu, C. Gao, C. Huang, C. Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

S. Yang, J. Kautz, and A. Hatamizadeh. Gated delta networks: Improving mamba2 with delta rule. *arXiv preprint arXiv:2412.06464*, 2024.

Z. Zhou, T. Wu, Z. Jiang, F. Obeid, and Z. Lan. Value residual learning. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 28341–28356, 2025.

D. Zhu, H. Huang, Z. Huang, Y. Zeng, Y. Mao, B. Wu, Q. Min, and X. Zhou. Hyper-connections. *arXiv preprint arXiv:2409.19606*, 2024.

A SATFormer PyTorch Implementation

```python
import torch
import torch.nn as nn
import torch.nn.functional as F

class SATFormerResidual(nn.Module):
    def __init__(self, hidden_size, num_kv_heads, gate='relu'):
        super().__init__()
        self.num_kv_heads = num_kv_heads
        self.gate = gate
        # Lightweight linear projection: hidden_size -> num_kv_heads
        self.alpha_proj = nn.Linear(hidden_size, num_kv_heads, bias=False)

    def _apply_gate(self, logits):
        if self.gate == 'relu':
            return F.relu(logits)
        elif self.gate == 'sigmoid':
            return torch.sigmoid(logits)
        elif self.gate == 'softmax':
            # Competition across heads; scaled to match unit initialization
            return F.softmax(logits, dim=-1) * self.num_kv_heads
        elif self.gate == 'softmax_sigmoid':
            # Combined across-head competition and per-head gating
            return F.softmax(logits, dim=-1) * torch.sigmoid(logits) * self.num_kv_heads
```

```
        elif self.gate == 'tanh':
            return torch.tanh(logits)
        return logits

    def forward(self, v_n, v_1, hidden_states):
        # Compute token-dependent mixing coefficients alpha
        # logits shape: [Batch, Seq, Heads]
        logits = self.alpha_proj(hidden_states.to(v_n.dtype))
        alpha = self._apply_gate(logits).unsqueeze(-1) # [B, T, H, 1]

        # Apply mixing: V'_n = V_n + alpha(h) * V_1
        return v_n + alpha * v_1.to(v_n.dtype)

# --- Integration into Model Forward Pass ---
# In SATFormerModel.forward():
V1 = None
for layer_idx, layer in enumerate(self.layers):
    # Each layer returns (hidden_states, ..., v_base)
    # v_base is the raw V projection from the current layer's Attention module
    outputs = layer(hidden_states, V1=V1)
    hidden_states = outputs[0]
    v_base = outputs[-1]

    # Step 1: Capture V1 from Layer 0
    if layer_idx == 0:
        V1 = v_base  # Store raw projection for all subsequent layers

    # Step 2: Propagation
    # For layers n > 0, the 'V1' variable now contains the stored projection.
    # It is passed into the Attention module of the next layer via 'layer(...)'.
```

B Ablation Study

B.1 Gating Analysis

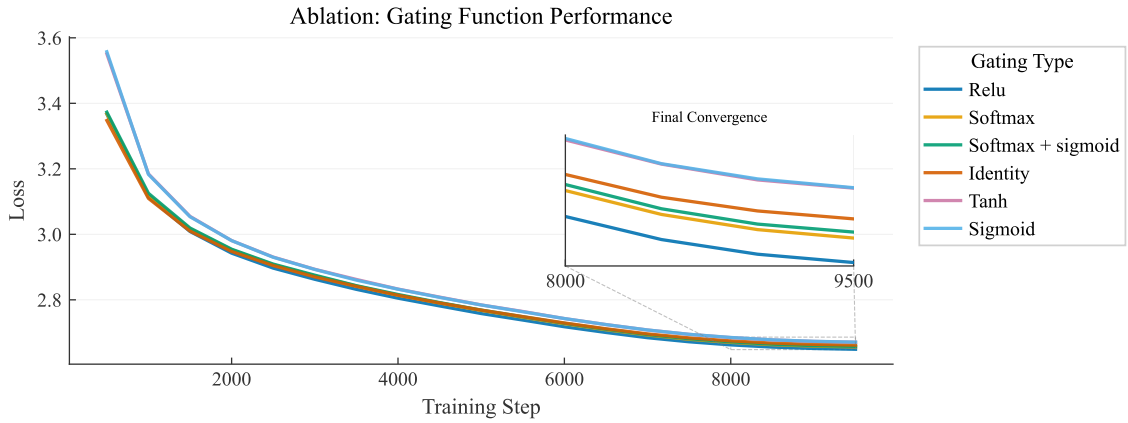

Figure 7: **Impact of Gating Functions on Convergence.** Several gating mechanisms used to compute the token-wise mixing coefficient α at the smallest scale (130M parameters and 5B tokens).

To identify the optimal mechanism for token-dependent value mixing, we evaluate several candidate gating functions for the coefficient $\alpha_{t,j}^{(n)}$.We tested six specific gating behaviors (Note: we use the standard PyTorch Kaiming uniform initialization for all linear layers):

1. **ReLU (Default)**: Produces sparse, non-negative weights, allowing the model to completely zero out the connection if/when V_1 is redundant.
2. **Sigmoid**: Provides a smooth, bounded $[0, 1]$ probability-like weight for each head.

3. **Softmax**: Computes a convex combination across the head dimension, forcing heads to "compete" for a shared V_1 budget (scaled by N_{kv} to maintain signal parity).

4. **Softmax + Sigmoid**: A hybrid approach combining across-head competition with an individual per-head on/off gate.

5. **Tanh**: Allows for subtractive mixing ($[-1, 1]$), enabling the model to suppress features in the current layer using the original input.

6. **Identity**: A linear baseline (Wh) used to verify the necessity of non-linear activation.

As shown in Figure 7, the ReLU and Softmax gates demonstrate the most stable convergence and lowest final loss (although ReLU maintains a significant lead over Softmax). Surprisingly, Sigmoid and Tanh perform the worst, possibly suggesting that bounded gating limits the expressivity enabled with value mixing. Overall, ReLU is simultaneously the simplest and strongest performing gating function, exhibiting a favorable inductive bias for enabling deeper expressivity with value mixing.

B.2 Depth vs. Width Study

The main 760M results show that SATFormer improves both validation loss and zero-shot language modeling over ResFormer, however its advantage is less pronounced than at the 130M, 340M, and 1.3B scales. To test whether this behavior reflects the architecture itself or the particular width vs. depth allocation used at 760M, we ran an additional matched-scale experiment using a "thin-and-long" configuration (1536 d_{model}, 24 L, 16 n_{heads}, 4352 d_{ff}). This configuration reallocates capacity toward depth while remaining in the same approximate parameter regime.

While both value-residual architectures benefit from the deeper allocation, SATFormer benefits more strongly. In Table 6, ResFormer improves from 2.2461 to 2.2396, whereas SATFormer improves from 2.2424 to 2.2313, corresponding to approximately a 1.7× larger reduction for SATFormer. The resulting SATFormer-Large-Thin model also outperforms the corresponding ResFormer-Large-Thin model by 0.0083 validation loss.

Table 6: **Depth vs. Width Final Validation Loss**

Model	Validation Loss
ResFormer-Large-Thin	2.2396
ResFormer-Large	2.2461
SATFormer-Large-Thin	**2.2313**
SATFormer-Large	2.2424

Table 7: **Depth vs. Width zero-shot performance across language modeling tasks.**

Model	Wiki. PPL ↓	LMB. PPL ↓	LMB. ACC ↑	PIQA ACC ↑	Hella. ACC_N ↑	Wino. ACC ↑	ARC-e ACC ↑	ARC-c ACC_N ↑	SIQA ACC ↑	BoolQ ACC ↑	Avg. ↑
ResFormer	20.73	**21.81**	**39.01**	67.85	**46.61**	51.54	61.32	31.06	39.10	**59.76**	49.53
SATFormer	**20.42**	22.04	38.19	**68.17**	46.58	**52.64**	**63.30**	**31.74**	**39.36**	58.47	**49.81**

The zero-shot results in Table 7 show a similar reversal: in the original 760M configuration, ResFormer narrowly outperformed SATFormer in average accuracy, but under the thin-and-long allocation SATFormer takes a larger step forward, improving to 49.81 average accuracy compared with 49.53 for ResFormer. These results support the interpretation that SATFormer benefits disproportionately from depth. This is consistent with the main mechanistic analysis, where gate activation is sparse in early layers and becomes stronger and more head-specialized in later layers. The thin-and-long study suggests that the weaker relative 760M endpoint result is not evidence against the selective-access mechanism, but rather reflects sensitivity to the allocation of parameters between depth and width.

C Additional Mechanistic Interpretability

This appendix provides additional details for the interpretability analysis in Section 5. The main paper uses three pieces of evidence: sparse depth-dependent gating, logit-lens behavior, and category-specific gate activation. Here, we describe how the token-feature categories were constructed and provide supplementary analyses showing that the broad content of V_1 is similar across models, while SATFormer changes how those early features are accessed.

C.1 SAE Training and Feature Labeling

To obtain coarse categories for V_1 features, we train Sparse Autoencoders (SAEs) (Cunningham et al., 2023) on the first-layer value activations of the 760M-parameter Transformer and SATFormer models. The SAEs use a Top-K activation function with $k = 32$ and an expansion factor of 16. We train both models for 100M tokens using Adam with learning rate 5×10^{-4} and a cosine annealing schedule. The resulting SAEs achieve Fraction of Variance Explained above 99%, indicating that the sparse codes preserve most of the variance in the underlying activation space.

We use the SAEs only to construct a coarse probe of which types of early features receive stronger gate activation. The goal is not to provide a complete semantic decomposition of V_1, but to distinguish broad classes of early representations that may be accessed differently by SATFormer.

After training, we identify and categorize high-confidence features using a three-stage automated pipeline:

1. **Density filtering and MonoScore ranking.** We remove dead or ultra-sparse features with fewer than 10 activations, as well as overly dense features with activation density above 0.5%. For each remaining feature, we compute a MonoScore based on the similarity structure of its top activating tokens.
2. **LLM consistency scoring.** We select the top 20% of features by MonoScore for qualitative evaluation. Using Qwen3-30B-A3B-Instruct-2507-FP8 (Yang et al., 2025) served with vLLM (Kwon et al., 2023), we perform a consistency check over two non-overlapping sets of highly activating tokens. A feature is retained only if both subsets receive a feature-purity score of at least 7 out of 10.
3. **Taxonomic categorization.** Retained features are assigned to a predefined taxonomy of linguistic and structural categories. The taxonomy is designed to capture broad distinctions relevant to early-token representations, including semantic fragments, structural tokens, code-related tokens, and tokenizer artifacts.

For visual clarity, we aggregate the fine-grained SAE feature labels in Table 8 into three broader categories used in the main-text analysis:

1. **Semantic:** Semantic Root Fragments, Entity Marker Fragments, Logic Quantifier Keywords, Morphological Suffixes, and Functional Leading Space.
2. **Functional:** Multilingual Unicode, Byte-Level Fallbacks, and Numerical Digit Fragments.
3. **Structural:** Structural Punctuation and Code Formatting Symbols.

C.2 Supplementary Gate Statistics

The heatmap in the main paper shows that large gate activations are concentrated in a subset of later-layer heads. Figure 8 provides a complementary aggregate view of the same behavior. Mean gate activation remains low in early layers and increases in later layers, while the fraction of exactly zero gate values decreases. At the same time, the coefficient of variation across heads increases, indicating that later-layer access is not only stronger but also more head-specialized.

Table 8: Taxonomy of SAE feature categories for V_1 activations.

Feature Category	Description	Examples
Semantic Root Fragments	Core meaning-bearing subwords	*struct, trans, ology*
Entity Marker Fragments	Capitalized fragments often used in proper names	*Mc, San, Al*
Morphological Suffixes	Subwords that modify parts of speech	*ing, ed, s, ly*
Logic Quantifier Keywords	Connectives and quantifiers that direct sentence flow	*if, however, all, but*
Functional Leading Space	Tokens that represent the start of a new word	*The, Apple, Run*
Numerical Digit Fragments	Numbers, decimals, or ordinal indicators	*12, 4, .5, th*
Structural Punctuation	Symbols used for syntax, grouping, or boundaries	*., :, (, ”*
Multilingual Unicode	Valid non-ASCII scripts	*α, é, ñ, ß*
Code Formatting Symbols	Programming syntax or whitespace/indentation	*{, \n, ++*
Byte-Level Fallbacks	Raw UTF-8 bytes or non-standard characters	*<0x0A>*

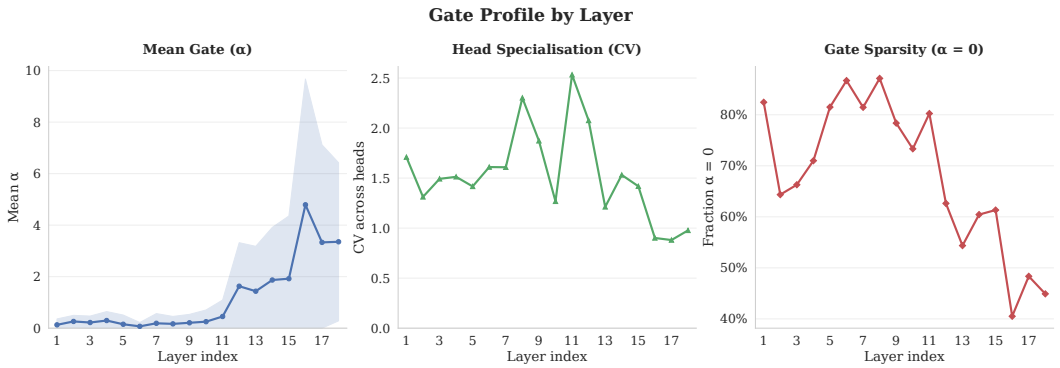

Figure 8: **Gate profile by layer.** Mean gate activation increases in later layers, while gate sparsity decreases and head-level variation grows. This provides an aggregate view of the sparse, depth-dependent access pattern shown in the main-text heatmap.

This supports the interpretation in Section 5: SATFormer does not use V_1 as a uniform shortcut. Instead, the model learns a sparse access pattern that becomes stronger and more differentiated across heads in deeper layers.

C.3 Feature Distribution in V_1

We also compare the distribution of automatically labeled V_1 features across the baseline Transformer and SATFormer. SATFormer produces a larger number of automatically verified monosemantic features under our pipeline, with 3,939 retained features compared to 3,261 for the Transformer. We interpret this as evidence that the accessible feature structure differs between the models, but not as a direct measure of representational richness. This metric can be affected by SAE training dynamics, activation geometry, tokenizer artifacts, and the automated labeling procedure.

Figure 9 shows that the broad category distribution of labeled V_1 features is similar across the two architectures. This is important for the main analysis: SATFormer's category-specific gate behavior does not appear to arise from a large shift in the coarse content of V_1. Instead, the evidence is more consistent with the view that SATFormer primarily changes how early features are accessed through depth.

Limitations. The SAE-based categories should be interpreted as a coarse analysis tool rather than a definitive semantic labeling of early representations. The Llama-2 tokenizer produces many subword fragments and byte-level artifacts, which limits the granularity of linguistic categories.

In addition, the labeling pipeline relies on an LLM consistency check and therefore may inherit model-driven biases. For this reason, we use the SAE labels only to support the qualitative conclusion that SATFormer gate activation varies across broad feature categories.

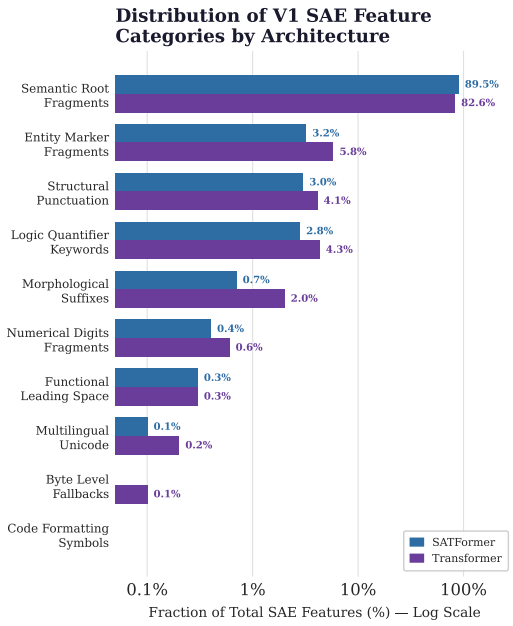

Figure 9: Transformer and SATFormer have similar category distributions, suggesting that SATFormer does not substantially change the content of V_1.