Revisiting Transformers through the Lens of Low Entropy and Dynamic Sparsity

Anonymous ACL submission

Abstract

Compression has emerged as an important perspective for understanding Transformers, encompassing both *data compression* and *model compression*. However, evaluating either remains challenging in practice: the underlying target distribution is typically unknown, while the exponentially large input space hinders both entropy estimation and sparsity characterization. To address these challenges, we adopt a controlled experimental setting where the target distribution is predefined and the corresponding input space can be exhaustively explored. From the data compression perspective, we find that Transformers exhibit a *low-entropy inductive bias*: beyond fitting the target distribution, they tend to learn lower-entropy distributions, with this tendency becoming increasingly pronounced in larger models. Further analysis shows that the FFN module drives this bias. From the model compression perspective, we observe stronger *dynamic sparsity* in larger Transformers, which increasingly bypass attention computation and activate fewer FFN neurons. Interestingly, we find that neuron sparsity emerges in a jump-like manner during training and is closely associated with loss spikes. Together, our findings shed light on how Transformer compression manifests in both learned distributions and internal computation.

1 Introduction

Recent studies have increasingly viewed the success of Transformers through the lens of compression (Sutskever, 2023; Delétang et al., 2023; Huang et al., 2024; Pan et al., 2025). Depending on the object being compressed, this perspective involves two complementary aspects: *data compression* and *model compression*. Data compression aims to remove redundancy from data. From this perspective, Transformers learn a parameterized distribution $\boldsymbol{p_\theta}$ to efficiently represent large-scale data and model the underlying target distribution $\boldsymbol{p}_{\mathrm{tgt}}$ with limited parameters. Training by maximizing log-likelihood is equivalent to minimizing the expected code length of the data (Delétang et al., 2023). According to Shannon's source coding theorem (Shannon, 1948), the entropy of the target distribution $\mathrm{H}(\boldsymbol{p}_{\mathrm{tgt}})$ determines the theoretical limit of compression performance. Moreover, it has also been argued that compression is closely related to intelligence (Hutter, 2005; Sutskever, 2023; Pan et al., 2025; Huang et al., 2024). Despite this connection, understanding *how well Transformers actually compress* remains challenging. In practice, the target distribution $\boldsymbol{p}_{\mathrm{tgt}}$ is unknown, making it difficult to assess whether the learned distribution $\boldsymbol{p_\theta}$ faithfully approaches $\boldsymbol{p}_{\mathrm{tgt}}$ or how their entropies relate during learning. Moreover, computing the entropy of an auto-regressive model is itself generally intractable: for a sequence of length n, evaluating $\mathrm{H}(\boldsymbol{p_\theta})$ requires marginalizing over $|\mathcal{V}|^n$ discrete outcomes while modern LLMs typically employ very large vocabularies.

On the other hand, compression also arises within the models themselves. While data compression removes redundancy in observations, model compression concerns redundancy in parameters, which is often characterized through dynamic sparsity (Martins et al., 2024). Such sparsity primarily manifests in the attention and FFN modules (Voita et al., 2023; Child et al., 2019; Peters et al., 2019; Correia et al., 2019). Despite recent progress in understanding dynamic sparsity, answering *how well Transformers can actually be compressed* also remains challenging. Similar to data compression, the difficulty also stems from the same factors. First, the underlying target distribution $\boldsymbol{p}_{\mathrm{tgt}}$ is unknown, while existing studies typically identify sparsity using naturally collected corpora, leaving open whether these sparse computation patterns generalize across the full input space induced by $\boldsymbol{p}_{\mathrm{tgt}}$. Second, exhaustively traversing all possible inputs is computationally infeasible under modern

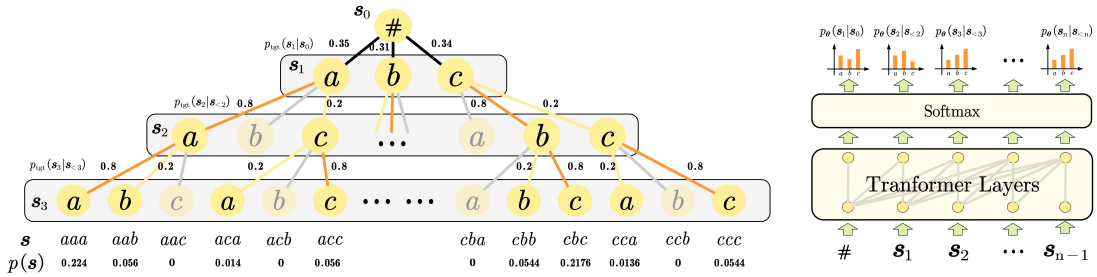

Figure 1: **Left:** An example of generating a target distribution $\boldsymbol{p}_{\text{tgt}}$ when $|\mathcal{V}| = 3$ and length $n = 3$. We choose the vocabulary $V = \{a, b, c\}$. We produce conditional probabilities to generate the final $p(\boldsymbol{s})$. We introduce "#" as the the start symbol $\boldsymbol{s}_0$ and we define $p(\boldsymbol{s}_0) = p(\#) = 1$. First, $\boldsymbol{p}(\boldsymbol{s}_1|\boldsymbol{s}_0)$ is generated from a uniform distribution. Then, when generating the subsequent $\boldsymbol{p}(\boldsymbol{s}_i|\boldsymbol{s}_{<i})$ for $i \geq 2$, we randomly select two characters and assign them transition probabilities of 0.8 and 0.2, while the remaining characters have a probability of 0. **Right:** The sequence $\boldsymbol{s}_{<i} = \boldsymbol{s}_0\boldsymbol{s}_1 \ldots \boldsymbol{s}_{i-1}$ is fed into the model to generate $p(\boldsymbol{s}_i|\boldsymbol{s}_{<i})$ and $p_{\boldsymbol{\theta}}(\boldsymbol{s})$ can be calculated as $p_{\boldsymbol{\theta}}(\boldsymbol{s}) = \prod_{i=1}^{n} p_{\boldsymbol{\theta}}(\boldsymbol{s}_i|\boldsymbol{s}_{<i})$.

vocabularies due to the exponentially large input space. Consequently, the extent of Transformer sparsity and the true compression potential of the model remain difficult to characterize.

To address these challenges from both data and model compression perspectives, we adopt a controlled experimental setting where the target distribution $\boldsymbol{p}_{\text{tgt}}$ is predefined and used to generate data. This setup not only enables computation of $\mathrm{H}(\boldsymbol{p}_{\boldsymbol{\theta}})$ and $\mathrm{KL}(\boldsymbol{p}_{\text{tgt}}|\boldsymbol{p}_{\boldsymbol{\theta}})$, but also allows exhaustive traversal of the input space beyond naturally observed corpora, allowing us to directly study how accurately Transformers compress data distributions and how well the models can be compressed.

From the data compression perspective, our experiments reveal a surprising property of Transformer learning. Beyond fitting the target distribution, Transformers exhibit a *low-entropy inductive bias*: they preferentially learn lower-entropy distributions, and this tendency becomes increasingly pronounced in larger models (Section 3.1). Due to this preference, Transformers often fail to perfectly match the target distribution, instead learning $\boldsymbol{p}_{\boldsymbol{\theta}}$ with entropy even lower than the true $\mathrm{H}(\boldsymbol{p}_{\text{tgt}})$. Further analysis shows that this preference is primarily driven by FFN modules (Section 3.2).

From the model compression perspective, we investigate dynamic sparsity in both attention and FFN modules[1] under our controlled setting. For attention, we study sparsity at the level of computational paths and find that Transformers favor residual routes that bypass attention computation (Section 4.1). For FFNs, we find that neuron sparsity is pervasive across the full input space: larger Transformers activate a smaller proportion of neurons and may even contain neurons that are never activated (Section 4.2). Moreover, this sparsity emerges in a jump-like manner during training, and loss spikes are strongly correlated with sudden decreases in neuronal activity. Together, these findings shed light on how Transformer compression manifests in both learned distributions and internal computation.

2 Preliminary

Language Modeling: We use $\boldsymbol{s} = [\boldsymbol{s}_i]_{i=1}^{n}$ to denote a sequence of length n, where $\boldsymbol{s}_{<i}$ denotes the prefix before position i. Given a vocabulary $\mathcal{V}$, let $\mathcal{S}_n$ denote the set of all sequences of length n, with size $|\mathcal{S}_n| = |\mathcal{V}|^n$. The target distribution over $\mathcal{S}_n$ is denoted by the vector $\boldsymbol{p}_{\text{tgt}}$, where each element $p_{\text{tgt}}(\boldsymbol{s}) = \prod_{i=1}^{n} p_{\text{tgt}}(\boldsymbol{s}_i \mid \boldsymbol{s}_{<i})$ represents the probability of a specific sequence $\boldsymbol{s}$ and satisfies $\sum_{\boldsymbol{s}\in\mathcal{S}_n} p_{\text{tgt}}(\boldsymbol{s}) = 1$. We introduce an additional character "#" as the beginning of the sequence ($\boldsymbol{s}_0$ is "#") and we define $p(\boldsymbol{s}_0) = p(\#) = 1$. A language model learns a parameterized distribution $\boldsymbol{p}_{\boldsymbol{\theta}}$ through next-token prediction $\mathcal{L}_{\text{NTP}}(\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{s}\sim\boldsymbol{p}_{\text{tgt}}}\left[-\frac{1}{n}\sum_{i=1}^{n}\log p_{\boldsymbol{\theta}}(\boldsymbol{s}_i|\boldsymbol{s}_{<i})\right]$. We study both the entropy $\mathrm{H}(\boldsymbol{p}_{\boldsymbol{\theta}})$ and the divergence $\mathrm{KL}(\boldsymbol{p}_{\text{tgt}}\|\boldsymbol{p}_{\boldsymbol{\theta}})$. However, in real-world settings, $\boldsymbol{p}_{\text{tgt}}$ is unknown and the sequence space grows exponentially with the large vocabulary size $|\mathcal{V}|$, making these quantities generally intractable to compute. To address this issue, we adopt a controlled experimental setup where $\boldsymbol{p}_{\text{tgt}}$ is known and the vocabulary size is controllable.

[1] Here the attention module contains multi-head attention computation along with residual connection (similarly for the FFN module).

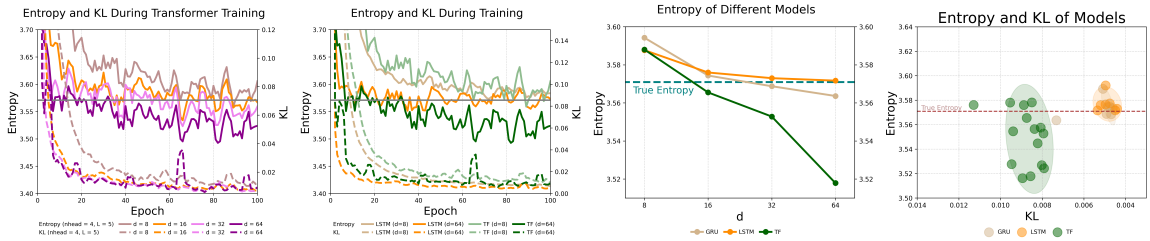

Figure 2: **Left:** Entropy and KL During Training for Transformers of Different Sizes. The value of $\mathrm{H}(\boldsymbol{p}_{\mathrm{tgt}})$ denoted by the horizontal gray line. **Center Left:** Entropy and KL During Training for Transformer and LSTM when $d = 8$ and $d = 64$. **Center Right:** The change in entropy with model size for GRU, LSTM and Transformer, averaged over the last 15 epochs. **Right:** The relationship between entropy and KL for different model configurations (with each configuration averaged over the last 15 epochs).

Controlled Experiments: To obtain a known $\boldsymbol{p}_{\mathrm{tgt}}$, we fix the vocabulary size $|\mathcal{V}|$ and sequence length n, and then generate the conditional distributions $\boldsymbol{p}(\boldsymbol{s}_i|\boldsymbol{s}_{<i})$ through random sampling. Specifically, we first sample $\boldsymbol{p}(\boldsymbol{s}_1|\boldsymbol{s}_0)$ from a uniform distribution as the beginning and then follow a sparsity principle to generate subsequent $\boldsymbol{p}(\boldsymbol{s}_i|\boldsymbol{s}_{<i})$, meaning that only a few characters are assigned high transition probabilities while the remaining ones are assigned zero probability. This sparsity principle is intuitive: in natural language, when the prefix is fixed, only a few words from the vocabulary are likely to follow. We can alter $\mathrm{H}(\boldsymbol{p}_{\mathrm{tgt}})$ by controlling the degree of sparsity. For example, when generating the conditional probabilities, we can randomly select the transition probabilities of two characters to be 0.8 and 0.2, with the remaining characters having a probability of 0 as illustrated in Figure 1. The final $\boldsymbol{p}_{\mathrm{tgt}}$ can be obtained from these generated conditional distributions. Since the generated $\boldsymbol{p}_{\mathrm{tgt}}$ is known, we can directly compute $\mathrm{H}(\boldsymbol{p}_{\mathrm{tgt}})$ and $\mathrm{KL}(\boldsymbol{p}_{\mathrm{tgt}}|\boldsymbol{p}_{\boldsymbol{\theta}})$ to evaluate the model's behavior. Additional details and alternative distributions are provided in Appendix B.

3 Transformers Prefer Low-entropy Distributions

From the perspective of data compression, a language model aims to approximate the target distribution $\boldsymbol{p}_{\mathrm{tgt}}$ as accurately as possible, thereby also approaching its entropy $\mathrm{H}(\boldsymbol{p}_{\mathrm{tgt}})$. However, in real-world settings, it remains difficult to precisely assess *how well language models achieve compression*: the target distribution is typically unknown, while computing the entropy of learned distributions is generally intractable due to the exponentially large sequence space. Under the controlled setting introduced in Section 2, we can directly track both the entropy $\mathrm{H}(\boldsymbol{p}_{\boldsymbol{\theta}})$ of the learned distribution and its divergence from $\boldsymbol{p}_{\mathrm{tgt}}$. In this section, we investigate how these quantities evolve during training and vary across different architectures and model scales. Our main observation is that Transformers exhibit a distinct low-entropy bias: rather than merely approaching $\boldsymbol{p}_{\mathrm{tgt}}$, larger Transformers tend to learn distributions whose entropy falls below the true entropy $\mathrm{H}(\boldsymbol{p}_{\mathrm{tgt}})$.

3.1 Transformers Exhibit a Low-entropy Bias

We generate the target distribution $\boldsymbol{p}_{\mathrm{tgt}}$ following Section 2, where the resulting entropy is $\mathrm{H}(\boldsymbol{p}_{\mathrm{tgt}}) = 3.571$[2]. We then sample sequences from $\boldsymbol{p}_{\mathrm{tgt}}$ and train autoregressive language models on the generated data. We compare Transformers with two classical RNN architectures, GRU (Chung et al., 2014) and LSTM (Hochreiter and Schmidhuber, 1997), across multiple model sizes. For fair comparison, model sizes are chosen to be comparable across architectures under the same embedding dimension. During training, we monitor both the entropy $\mathrm{H}(\boldsymbol{p}_{\boldsymbol{\theta}})$ of the learned distribution and the divergence $\mathrm{KL}(\boldsymbol{p}_{\mathrm{tgt}}|\boldsymbol{p}_{\boldsymbol{\theta}})$. Additional details are provided in Appendix B. In the following, instead of using $\boldsymbol{p}_{\boldsymbol{\theta}}$ directly, we use $\boldsymbol{p}_{\mathrm{tf}}$ and $\boldsymbol{p}_{\mathrm{rnn}}$ to denote the distributions learned by Transformers and RNNs respectively.

Larger Transformers can learn distributions with lower entropy: Firstly, for Transformers, we monitor the changes in $\mathrm{H}(\boldsymbol{p}_{\mathrm{tf}})$ and $\mathrm{KL}(\boldsymbol{p}_{\mathrm{tgt}}|\boldsymbol{p}_{\mathrm{tf}})$ with different model sizes during training as shown

[2] In practice, there may exist slight discrepancies between the sampled empirical distribution and the generated target distribution even with sufficient samples. Therefore, we treat the sampled distribution as the target distribution $\boldsymbol{p}_{\mathrm{tgt}}$ for evaluation.

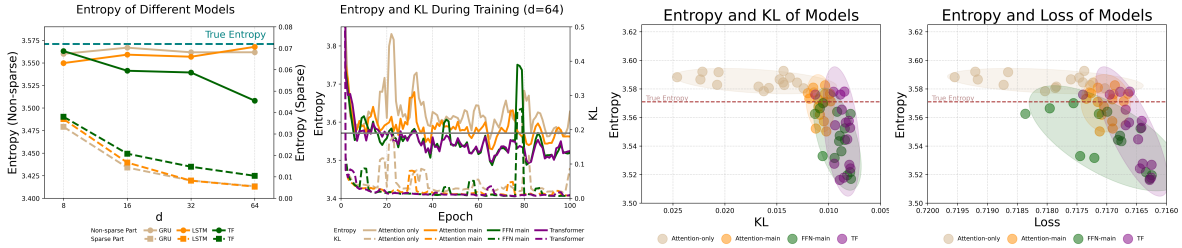

Figure 3: **Left:** The change in entropy of the sparse and non-sparse parts with model size. **Center Left:** Entropy and KL during training for different Transformer variants. **Center Right and Right:** Relationship between KL/Loss and Entropy for different variants. FFN-main can achieve lower entropy compared to Attention-only and Attention-main.

in the left part of Figure 2. It can be seen that as the model size increases, $\mathrm{H}(\boldsymbol{p}_{\mathrm{tf}})$ generally decreases. At first glance, this is not surprising: larger models often have stronger expressive power, making them more likely to achieve a distribution with lower entropy and the reduction in $\mathrm{H}(\boldsymbol{p}_{\mathrm{tf}})$ may be due to $\boldsymbol{p}_{\mathrm{tf}} \to \boldsymbol{p}_{\mathrm{tgt}}$ when enlarging the model size. However, we notice that in the later stages of training, $\mathrm{H}(\boldsymbol{p}_{\mathrm{tf}})$ with larger models is often smaller than $\mathrm{H}(\boldsymbol{p}_{\mathrm{tgt}}) = 3.571$ (denoted by a horizontal gray line). This raises our speculation that: Larger Transformers prefer to learn $\boldsymbol{p}_{\mathrm{tf}}$ with lower entropy when approaching the target distribution $\boldsymbol{p}_{\mathrm{tgt}}$.

Transformers prefer low-entropy landscape compared to RNNs: Does RNNs exhibit similar behaviors when increasing the model size? Here we provide a negative answer. As shown in the center-left part of Figure 2, the entropy of RNNs gradually approaches the true entropy $\mathrm{H}(\boldsymbol{p}_{\mathrm{tgt}})$ as model size increases, rather than continuing to decrease below $\mathrm{H}(\boldsymbol{p}_{\mathrm{tgt}})$ as observed in Transformers. Meanwhile, the KL divergence of RNNs also decreases during training, indicating that larger RNNs progressively approach the target distribution. This difference becomes clearer in the center-right part of Figure 2, where we report the average entropy and KL divergence over the last 15 epochs across different model sizes. To further verify the robustness of this phenomenon, we additionally vary architectural settings including model depth and hidden dimensions (Appendix B) and compute these quantities. The results are summarized in the right part of Figure 2. Overall, Transformers consistently learn distributions with lower entropy and larger KL divergence compared to RNNs. In contrast, RNNs tend to converge near the entropy of the target distribution ($\mathrm{H}(\boldsymbol{p}_{\mathrm{rnn}}) \approx \mathrm{H}(\boldsymbol{p}_{\mathrm{tgt}})$), causing the scatter points to eventually converge in a small region, without exhibiting the significantly lower entropy ($\mathrm{H}(\boldsymbol{p}_{\mathrm{tf}}) < \mathrm{H}(\boldsymbol{p}_{\mathrm{tgt}})$) as in Transformers. Therefore, we present our first observation:

Observation 1. Transformers exhibit a preference toward lower-entropy distributions when fitting the target distribution.

Our observations suggest that Transformers may possess an implicit preference toward lower-entropy solutions beyond simply minimizing the divergence to the target distribution. As an intuitive interpretation, this behavior may resemble an entropy-related implicit regularization effect: $\min_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}) = \mathcal{L}_{\mathrm{NTP}}(\boldsymbol{\theta}) + \alpha \mathrm{H}(\boldsymbol{p}_{\boldsymbol{\theta}})$ where $\mathcal{L}_{\mathrm{NTP}}(\boldsymbol{\theta})$ is the next-token prediction loss, $\mathrm{H}(\boldsymbol{p}_{\boldsymbol{\theta}}) = -\mathbb{E}_{\boldsymbol{s}\sim\boldsymbol{p}_{\boldsymbol{\theta}}}[\log p_{\boldsymbol{\theta}}(\boldsymbol{s})]$, and α may depend on model scale. Under this interpretation, larger Transformers not only approach the target distribution but also tend to further reduce the entropy of the learned distribution, which may explain the larger KL divergence relative to RNNs shown in Figure 2. However, as previously discussed, such behaviors may be difficult to directly observe in practice due to the unknown target distribution and the intractability of entropy computation in large sequence spaces. Although this formulation is only an intuitive interpretation of the observed behavior, it may provide a direction for understanding entropy-related inductive biases in Transformers.

Larger Transformers carry less from the seen data. Further, we measure the contribution of two parts of the sequences to the total entropy: *Sparse part* $\mathcal{S}_{\mathrm{spa}} = \{\boldsymbol{s} \in \mathcal{S}_n \mid p_{\mathrm{tgt}}(\boldsymbol{s}) = 0\}$ and *Non-sparse part* $\mathcal{S}_{\mathrm{non}} = \mathcal{S}_n / \mathcal{S}_{\mathrm{spa}} = \{\boldsymbol{s} \in \mathcal{S}_n \mid p_{\mathrm{tgt}}(\boldsymbol{s}) > 0\}$. Note that models only see sequences from $\mathcal{S}_{\mathrm{non}}$ during training and should learn to approximate their true probabilities, while assigning nearly zero probability to sequences $\boldsymbol{s} \in \mathcal{S}_{\mathrm{spa}}$. The

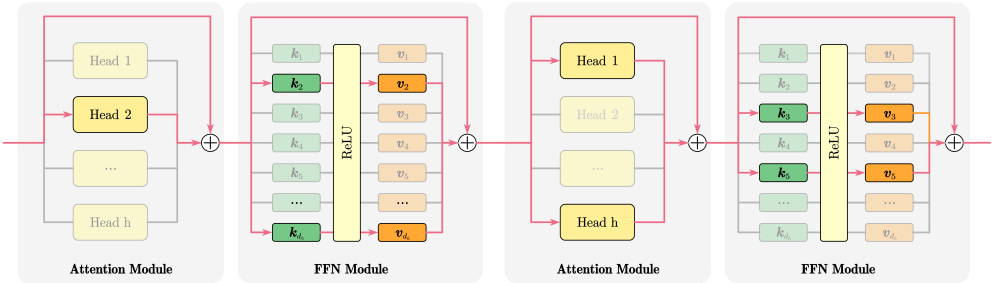

Figure 4: Explanation of dynamic sparsity in the attention and FFN modules of Transformers. As for the attention modules, each attention head or residual connection can be viewed as a forward path andlarger Transformers prefer residual connections to bypass attention head computations. While for the FFN modules, the parameters of the first and second layer can be viewed as key and value vectors respectively and each neuron is a key-value pair. Larger Transformers will have a lower proportion of active neurons (i.e., key-value pairs).

contribution to the total entropy can be computed as $-\sum_{\boldsymbol{s}\in\mathcal{S}} p_{\boldsymbol{\theta}}(\boldsymbol{s}) \log p_{\boldsymbol{\theta}}$ where $\mathcal{S}$ is $\mathcal{S}_{\text{spa}}$ or $\mathcal{S}_{\text{non}}$. The main results can be seen in the left part Figure 3. As model size increases, the entropy contribution from the sparse part decreases, indicating that larger models increasingly suppress probabilities assigned to unseen sequences. At the same time, RNNs gradually approach the true entropy over the non-sparse part. In contrast, the entropy of Transformers over the non-sparse region continues to decrease below the true entropy as model size grows. These results suggest that the low-entropy behavior of Transformers mainly arises from learning sharper distributions over observed sequences, rather than solely from suppressing unseen ones.

3.2 FFNs Drive the Low-entropy Tendency

In the previous section, we showed that Transformers exhibit a preference toward lower-entropy distributions. A natural question is: *which components of Transformers are primarily responsible for this behavior?* To investigate this, we perform architectural ablations on the attention and FFN modules: (1) *Attention-only:* all FFN modules are removed; (2) *Attention-main:* only the FFN in the final layer is retained; (3) *FFN-main:* only the attention module in the first layer is retained. We do not consider an FFN-only setting since stacked FFN layers alone cannot effectively model sequential dependencies. Additional details are provided in Appendix B.

FFN modules are strongly associated with the low-entropy bias: We first compare the entropy dynamics of different architectures for the case of $d = 64$, shown in the left part of Figure 3 (additional results are provided in Figure 10). Compared to FFN-main and the original Transformer, Attention-only maintains noticeably higher entropy throughout training and generally remains above the true entropy $\mathrm{H}(\boldsymbol{p}_{\text{tgt}})$. After introducing a single FFN layer in Attention-main, the entropy decreases and fluctuates around $\mathrm{H}(\boldsymbol{p}_{\text{tgt}})$. In contrast, FFN-main exhibits entropy behavior much closer to the original Transformer, with entropy frequently falling below $\mathrm{H}(\boldsymbol{p}_{\text{tgt}})$ during later training stages. To further verify this phenomenon, we report the average entropy, KL divergence, and loss over the final 15 epochs across multiple settings in the center and right parts of Figure 3. Overall, architectures retaining most FFN modules consistently exhibit lower entropy, whereas Attention-only remains substantially closer to or above the target entropy. Therefore, we summarize our second observation as follows:

Observation 2. FFN modules primarily drive the low-entropy tendency of Transformers.

4 Dynamic Sparsity in Transformers

In the previous section, we revealed the low-entropy preference exhibited by Transformers during data compression. While compressing redundancy in data distributions, Transformers themselves may also contain redundancy in their internal computation, which can be characterized through dynamic sparsity from the perspective of model compression. However, similar to data compression, it remains difficult to precisely assess *how well Transformers can actually be compressed in real-world settings*. This is because sparsity patterns are typically observed only on finite corpora, while their behaviors across the full input space induced by the unknown $\boldsymbol{p}_{\text{tgt}}$ remain unclear. At the same time, exhaustively traversing such a space is computationally infeasible due to its exponen-

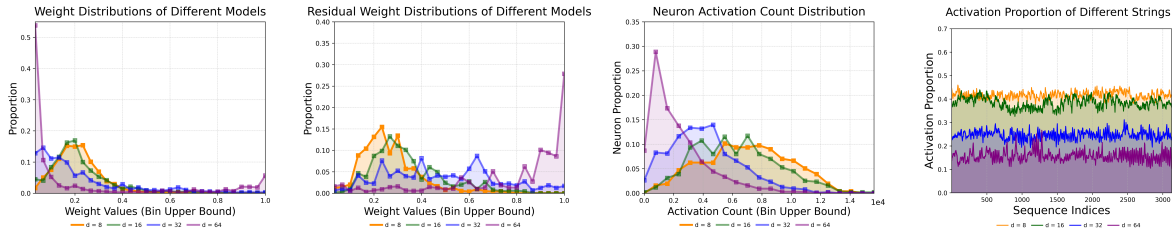

Figure 5: **Left:** The weight distribution of all paths for different model sizes. **Center Left:** The weight distribution of only residual paths for different model sizes. **Center Right:** The distribution of neuron activation counts for Transformers of different sizes. **Right:** The proportion of neurons activated across different input sequences.

tial size. Under our controlled setting, we can directly analyze sparsity patterns across the full input space. In this section, we investigate dynamic sparsity in the attention and FFN modules and we will show that larger Transformers exhibit increasingly sparse computation patterns in both modules. An overview of this section is provided in Figure 4.

4.1 Dynamic Sparsity in the Attention Module

Residual connections are considered essential to Transformer optimization and expressivity (Dong et al., 2021; He et al., 2023; He and Hofmann, 2023). Inspired by Dong et al. (2021), we view attention heads and residual connections as alternative computational paths within each layer. Rather than studying token-level sparse attention, we focus on sparsity at the level of computational paths. A natural question is whether Transformers implicitly prefer residual shortcuts over attention computation, particularly as model size increases. To investigate this, we slightly modify the original attention module by incorporating a dynamic routing mechanism. For the i-th token $\boldsymbol{x}_i$, the modified attention output is defined as

$$\mathrm{Atten}(\boldsymbol{x}_i) = w_0 \boldsymbol{x}_i + \sum_{h=1}^{H} w_h \mathrm{Head}^{(h)}(\boldsymbol{x}_i),$$

where H is the number of heads and $[w_0, w_1, \ldots, w_H] \in \mathbb{R}^{H+1}$ is a routing distribution produced by a input-dependent gating network and then normalized by softmax. Here, w_h denotes the routing weight assigned to the h-th attention head, while w_0 corresponds to the residual connection path. After training the model as in Section 3.1, we exhaustively traverse the full input space and collect the routing weights of all paths. Each layer contains five paths in total (four attention heads and one residual path). We divide the interval $[0, 1]$ into 30 bins and compute the distribution of routing weights across all paths.

Larger Transformers Favor Residual Paths: We first vary the model size and present the routing-weight distributions in the left part of Figure 5. When the model size is small, the routing weights are concentrated around 0.2, suggesting that all five paths contribute relatively evenly to the forward computation. However, as the model size increases, the routing distribution becomes increasingly polarized, with more paths receiving weights close to zero. For example, when $d = 64$, more than half of the routing weights are nearly zero, indicating that many computational paths are effectively ignored. We further analyze the routing weights assigned to residual connections in the center-left part of Figure 5. As model size increases, the proportion of large residual weights rises substantially, and nearly 30% of residual routing weights become close to 1 when $d = 64$. This suggests that larger Transformers increasingly favor residual paths over attention computation. Therefore, we summarize our observation as follows:

Observation 3. Larger Transformers increasingly favor residual paths over attention computation, leading to sparser routing patterns in the attention module.

4.2 Dynamic Sparsity in the FFN Module

FFN modules are another core component of Transformers. Previous work (Geva et al., 2020; Voita et al., 2023; Shen et al., 2023) interprets FFN neurons as input-dependent key-value memories. Specifically, denoting $\mathrm{FFN}(\boldsymbol{x}) = \boldsymbol{W}_2 \mathrm{ReLU}(\boldsymbol{W}_1 \boldsymbol{x})$, the first-layer parameters $\boldsymbol{W}_1 \in \mathbb{R}^{d_h \times d}$ can be viewed as d_h key vectors, while $\boldsymbol{W}_2$ corresponds to d_h value vectors. For a given input $\boldsymbol{x}$, the FFN selectively activates neurons

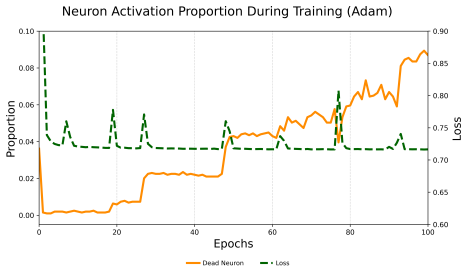

Figure 6: Loss and proportion of dead neurons during training when $d = 64$.

according to the similarity between $\boldsymbol{x}$ and the key vectors, which can be written as $\mathrm{FFN}(\boldsymbol{x}) = \sum_{i=1}^{d_h} \max(\boldsymbol{x}^\top \boldsymbol{k}_i, 0)\boldsymbol{v}_i$. As a result, only a subset of neurons contributes to the output for each input, naturally inducing dynamic sparsity in FFN computation. Recent studies (Voita et al., 2023) have observed substantial neuron sparsity in LLMs on collected corpora. However, it remains unclear whether such sparsity persists across the full input space beyond observed data. Moreover, the emergence of these sparse patterns during training is still insufficiently understood, while monitoring neuron activity on large-scale corpora can itself be computationally expensive.

Under our controlled setting, we can further analyze this phenomenon across the full input space. Our experimental setup is basically consistent with that in Section 3.1 and more details can be found in Appendix C. Here, all possible inputs consist of $|\mathcal{V}|^n$ sequences, allowing us to examine the activation of each FFN neuron across $N_{\max} = n|\mathcal{V}|^n$ tokens, which is also the maximum possible activation count. We uniformly divide the interval $(0, N_{\max}]$ into 20 bins and count the proportion of neurons whose activation counts fall into each one. Additionally, we specially count the proportion of dead neurons (i.e., never activated).

Larger Transformers exhibit stronger FFN sparsity: We first analyze the distribution of neuron activation counts across different model sizes, shown in the center-right part of Figure 5. From a global perspective across the full input space, the activation-count distribution becomes increasingly concentrated near zero as model size increases, indicating that larger Transformers contain substantially more inactive neurons. For example, when $d = 64$, more than 25% of neurons are activated on fewer than $5\% N_{\max}$ tokens, while nearly 10% of neurons are never activated across the full input space. These observations extend previous findings on neuron sparsity in LLMs (Voita et al., 2023) by showing that such sparse behaviors persist even under exhaustive traversal of all possible input sequences, which is typically infeasible in real-world settings due to the exponentially large sequence space. We further analyze the proportion of activated neurons for each input sequence in the right part of Figure 5. From a local per-input perspective, larger models activate progressively smaller fractions of neurons for each input, suggesting increasingly sparse computation at the individual-input level. For example, when $d = 64$, only about 15% of neurons are activated on average for each input sequence, compared to more than 40% when $d = 8$. Together, these results suggest that larger Transformers exhibit increasingly sparse FFN computation both globally across the input space and locally for individual inputs.

Observation 4. Larger Transformers exhibit increasingly sparse FFN computation, both globally across the input space and locally for individual inputs.

The FFN module emerges sparsity in a jump-like manner. The above results reveal increasingly sparse FFN computation in larger Transformers. A natural follow-up question is how such sparsity emerges during training. To further investigate this, we monitor neuron activations throughout the training process. In Figure 6, we present the proportion of dead neurons during training. Surprisingly, we find that this dynamic sparsity enhances in a jump-like manner rather than continuously. Moreover, the enhancement timing closely coincides with the occurrence of loss spikes, where the loss rapidly increases and then returns to its previous level. This suggests that loss spikes are highly correlated with the sudden decrease in neuronal activity. We intuitively interpret this behavior as the model progressively suppressing subsets of neurons during training and subsequently re-adapting the remaining active neurons, gradually forming sparse FFN computation patterns. Interestingly, loss spikes have also been widely observed during LLM training (Chowdhery et al., 2023; Molybog et al., 2023). Under our controlled setting, we further observe that such spikes occur substantially less frequently in smaller models (see Figure 12 in Appendix), which is also consistent with their weaker FFN sparsity patterns observed above.

Adaptive second-moment estimation influences sparsity emergence: Previous work (Moly-

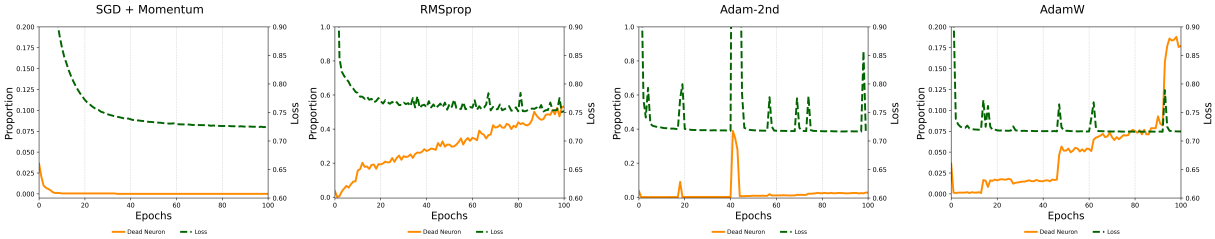

Figure 7: Loss and the proportion of dead neurons during training under different optimizers. **SGD with momentum** does not cause loss spikes or an increase in the proportion of dead neurons. **RMSprop** exhibits unstable training and a higher proportion of dead neurons. **Adam primarily relying on second-order gradients** causes more severe loss spikes and greater fluctuations in the dead neurons proportion compared to regular Adam while **AdamW** shows behavior similar to Adam.

bog et al., 2023) has linked loss spikes to particular optimization states of Adam-like optimizers. To further investigate the role of optimization dynamics in sparsity emergence, we compare several optimizers with different moment estimation behaviors. Noting that our default Adam setting is $\beta_1 = 0.9, \beta_2 = 0.999$ where β_1 controls the smoothing of first-moment gradients and β_2 controls adaptive second-moment estimation, to align as closely possible with Adam and ensure training performance, we set the following optimizers: (a) SGD + Momentum (Robbins and Monro, 1951) ($\beta_1 = 0.9$); (b) RMSprop (Zeiler, 2012) ($\beta_2 = 0.99$); (c) Adam-2nd ($\beta_1 = 0.01, \beta_2 = 0.999$); (d) AdamW (Loshchilov and Hutter, 2017). Among them, SGD + Momentum mainly relies on first-moment momentum, while RMSprop and Adam-2nd depend more heavily on adaptive second-moment statistics. AdamW is similar to Adam but decouples weight decay from gradient updates.

We present the main results in Figure 7. It can be observed that SGD + Momentum exhibits neither obvious loss spikes nor significant increases in dead neurons during training. In contrast, optimizers relying more heavily on adaptive second-moment information (RMSprop and Adam-2nd) show substantially stronger training instability and sparsity emergence. Interestingly, Adam-2nd exhibits unstable sparsity transitions when first-moment momentum is largely removed, suggesting that the interaction between first- and second-moment statistics may both contribute to the formation of sparse FFN patterns. anwhile, AdamW behaves similarly to Adam, indicating that weight decay plays a relatively limited role in this phenomenon. Therefore, we summarize the observation as follows:

Observation 5 The emergence of FFN sparsity is closely associated with loss spikes during training and is strongly influenced by optimizer dynamics, particularly adaptive second-moment estimation.

Discussions: Although we have shown the correlation between dynamic sparsity in FFNs and loss spikes, it remains to be further explored how Transformers guide the gradual enhancement of this dynamic sparsity. Moreover, FFN modules are unlikely to be the sole factor contributing to loss spikes, since similar spikes can also be observed in Attention-only models (see Figure 11). In addition, we find that larger Transformers are more prone to loss spikes than RNNs (see Figure 9). Together, these observations suggest that loss spikes may be influenced not only by adaptive second-moment optimization dynamics, but also by Transformer architectural properties themselves. We leave these aspects for our future work.

5 Conclusion

In this paper, we revisited Transformers from the perspectives of both data compression and model compression under a controlled setting that enables direct analysis of entropy and sparsity. From the data compression perspective, we show that Transformers exhibit an intrinsic preference toward lower-entropy solutions, particularly in larger models, and that this behavior is closely connected to FFN modules. As for model compression, we find that larger Transformers exhibit increasingly sparse computation in both attention and FFN modules, while FFN sparsity is closely associated with loss spikes and optimizer dynamics. Overall, our findings provide further insights into Transformer learned distributions and internal computation.

Limitations

Our work is conducted under a controlled experimental setup that enables direct computation of entropy, KL divergence, and sparsity across the full input space, which are generally intractable in real-world language modeling due to the exponentially large sequence space. While this controllability allows us to isolate and analyze Transformer behaviors with high precision, developing efficient approaches for estimating these quantities under realistic vocabularies and data distributions remains an important open problem. Such advances may help bridge controlled analyses and real-world language models. We hope our study can provide useful insights and experimental foundations for future work in this direction.

References

Mikel Artetxe, Shruti Bhosale, Naman Goyal, Todor Mihaylov, Myle Ott, Sam Shleifer, Xi Victoria Lin, Jingfei Du, Srinivasan Iyer, Ramakanth Pasunuru, and 1 others. 2021. Efficient large scale language modeling with mixtures of experts. *arXiv preprint arXiv:2112.10684*.

Weilin Cai, Juyong Jiang, Fan Wang, Jing Tang, Sunghun Kim, and Jiayi Huang. 2024. A survey on mixture of experts. *arXiv preprint arXiv:2407.06204*.

Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. 2019. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, and 1 others. 2023. Palm: Scaling language modeling with pathways. *Journal of Machine Learning Research*, 24(240):1–113.

Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*.

Gonçalo M Correia, Vlad Niculae, and André FT Martins. 2019. Adaptively sparse transformers. *arXiv preprint arXiv:1909.00015*.

Soham De and Sam Smith. 2020. Batch normalization biases residual blocks towards the identity function in deep networks. *Advances in Neural Information Processing Systems*, 33:19964–19975.

Grégoire Delétang, Anian Ruoss, Paul-Ambroise Duquenne, Elliot Catt, Tim Genewein, Christopher Mattern, Jordi Grau-Moya, Li Kevin Wenliang, Matthew Aitchison, Laurent Orseau, and 1 others. 2023. Language modeling is compression. *arXiv preprint arXiv:2309.10668*.

Yihe Dong, Jean-Baptiste Cordonnier, and Andreas Loukas. 2021. Attention is not all you need: Pure attention loses rank doubly exponentially with depth. In *International conference on machine learning*, pages 2793–2803. PMLR.

Maha Elbayad, Jiatao Gu, Edouard Grave, and Michael Auli. 2019. Depth-adaptive transformer. *arXiv preprint arXiv:1910.10073*.

William Fedus, Barret Zoph, and Noam Shazeer. 2022. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(120):1–39.

Mor Geva, Avi Caciularu, Kevin Ro Wang, and Yoav Goldberg. 2022. Transformer feed-forward layers build predictions by promoting concepts in the vocabulary space. *arXiv preprint arXiv:2203.14680*.

Mor Geva, Roei Schuster, Jonathan Berant, and Omer Levy. 2020. Transformer feed-forward layers are key-value memories. *arXiv preprint arXiv:2012.14913*.

Bobby He and Thomas Hofmann. 2023. Simplifying transformer blocks. *arXiv preprint arXiv:2311.01906*.

Bobby He, James Martens, Guodong Zhang, Aleksandar Botev, Andrew Brock, Samuel L Smith, and Yee Whye Teh. 2023. Deep transformers without shortcuts: Modifying self-attention for faithful signal propagation. *arXiv preprint arXiv:2302.10322*.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.

Yuzhen Huang, Jinghan Zhang, Zifei Shan, and Junxian He. 2024. Compression represents intelligence linearly. *arXiv preprint arXiv:2404.09937*.

Marcus Hutter. 2005. *Universal artificial intelligence: Sequential decisions based on algorithmic probability*. Springer Science & Business Media.

Marcus Hutter. 2006. The hutter prize. http://prize.hutter1.net.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Jakub Krajewski, Jan Ludziejewski, Kamil Adamczewski, Maciej Pióro, Michał Krutul, Szymon Antoniak, Kamil Ciebiera, Krystian Król, Tomasz Odrzygóźdź, Piotr Sankowski, and 1 others. 2024. Scaling laws for fine-grained mixture of experts. *arXiv preprint arXiv:2402.07871*.

Ilya Loshchilov and Frank Hutter. 2017. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*.

André Martins, Edoardo Ponti, Duarte Alves, Piotr Nawrot, and Saul Santos. 2024. Dynamic sparsity in the brain and machines routing information through neural pathways. https://dynamic-sparsity.github.io/. NeurIPS 2024 Tutorial.

Igor Molybog, Peter Albert, Moya Chen, Zachary DeVito, David Esiobu, Naman Goyal, Punit Singh Koura, Sharan Narang, Andrew Poulton, Ruan Silva, and 1 others. 2023. A theory on adam instability in large-scale machine learning. *arXiv preprint arXiv:2304.09871*.

Lorenzo Noci, Sotiris Anagnostidis, Luca Biggio, Antonio Orvieto, Sidak Pal Singh, and Aurelien Lucchi. 2022. Signal propagation in transformers: Theoretical perspectives and the role of rank collapse. *Advances in Neural Information Processing Systems*, 35:27198–27211.

Matanel Oren, Michael Hassid, Nir Yarden, Yossi Adi, and Roy Schwartz. 2024. Transformers are multi-state rnns. *arXiv preprint arXiv:2401.06104*.

Zhixuan Pan, Shaowen Wang, and Jian Li. 2025. Understanding llm behaviors via compression: Data generation, knowledge acquisition and scaling laws. *arXiv preprint arXiv:2504.09597*.

Ben Peters, Vlad Niculae, and André FT Martins. 2019. Sparse sequence-to-sequence models. *arXiv preprint arXiv:1905.05702*.

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

Herbert Robbins and Sutton Monro. 1951. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407.

Claude E Shannon. 1948. A mathematical theory of communication. *The Bell system technical journal*, 27(3):379–423.

Kai Shen, Junliang Guo, Xu Tan, Siliang Tang, Rui Wang, and Jiang Bian. 2023. A study on relu and softmax in transformer. *arXiv preprint arXiv:2302.06461*.

Ilya Sutskever. 2023. An observation on generalization. *Large Language Models and Transformers Workshop, Simons Institute*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Elena Voita, Javier Ferrando, and Christoforos Nalmpantis. 2023. Neurons in large language models: Dead, n-gram, positional. *arXiv preprint arXiv:2309.04827*.

Peng Wang, Yifu Lu, Yaodong Yu, Druv Pai, Qing Qu, and Yi Ma. 2025. Attention-only transformers via unrolled subspace denoising.

Jinrui Yang, Xianhang Li, Druv Pai, Yuyin Zhou, Yi Ma, Yaodong Yu, and Cihang Xie. 2024. Scaling white-box transformers for vision. *Advances in Neural Information Processing Systems*, 37:36995–37019.

Yaodong Yu, Sam Buchanan, Druv Pai, Tianzhe Chu, Ziyang Wu, Shengbang Tong, Benjamin Haeffele, and Yi Ma. 2023. White-box transformers via sparse rate reduction. *Advances in Neural Information Processing Systems*, 36:9422–9457.

Matthew D Zeiler. 2012. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*.

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

A Related Work

Data Compression and Transformers: Recently, data compression has emerged as an important perspective for understanding the success behind Transformer-based LLMs. Delétang et al. (2023) establish a connection between the maximum likelihood training objective of LLMs and arithmetic coding, proposing that LLMs are powerful lossless compressors. Moreover, a prevailing belief is that compression can lead to intelligence (Hutter, 2006; Sutskever, 2023; Pan et al., 2025). To quantitatively explore this, Huang et al. (2024) take knowledge and commonsense, coding, and mathematical reasoning as indicators of intelligence and observe the strong linear relationship between compression performance and downstream task performance. Previous works focus more on evaluating compression quality by measuring the discrepancy between the model's learned distribution and the target distribution. In contrast, our work lifts the constraint of the target distribution and instead focuses on the variation in the information content (entropy) of the learned distribution itself. Another line of work orthogonal to our study is those focusing more on understanding the architecture of Transformers from a compression perspective (Yu et al., 2023; Yang et al., 2024; Wang et al., 2025), whose core idea is to establish an equivalence between the forward process of Transformers and iterative algorithms related to data compression objective. Yu et al. (2023); Yang et al. (2024) view Transformers as an iterative process toward a objective called sparse rate reduction and propose an interpretable, white-box Transformer architecture. Wang et al. (2025) interprets attention layers as compressing noisy token representations into low-dimensional subspaces.

Mechanisms of Attention and FFN Modules: Previous work advances our understanding of the interplay between attention modules and residual connections. Dong et al. (2021) show that stacking pure self-attention layers will lead to the collapse of output representations into a rank-1 matrix, while residual connections effectively prevent this. Noci et al. (2022) analyze rank collapse from the perspective of signal propagation and propose mitigating it by scaling the residual branch which are adopted by De and Smith (2020); He et al. (2023); He and Hofmann (2023). Following Dong et al. (2021), our work views each attention head or skip connection as a path and introduces a routing mechanism to explore the dynamic sparsity of attention modules. For the FFN module, Geva et al. (2020) interpret the feed-forward layer as a key-value store and empirically show that the keys capture input patterns while the values contribute to the output distribution(Geva et al., 2022). The work most closely related to ours is Voita et al. (2023), which investigates neuron activations in the FFNs of the OPT(Zhang et al., 2022) family by collecting a large corpus, finding that many neurons in the early part of models are inactive (dead). However, despite the large-scale test corpus, it remains unclear whether such sparsity pattern holds on all underlying unseen inputs, and how the pattern forms is still an open question. Our work further explore these problems under a controlled experimental setup.

Dynamic Sparsity of Transformers: Dynamic sparsity is widely present in the structure of foundation models(Martins et al., 2024), including attention computation(Peters et al., 2019; Correia et al., 2019; Oren et al., 2024), MoE routing (Fedus et al., 2022; Artetxe et al., 2021; Krajewski et al., 2024), and sparse layers(Elbayad et al., 2019; Raposo et al., 2024), among others. For the attention module, previous work primarily focuses on sparsity in the attention map, that is, a token attends to only a small set of key tokens. However, inspired by Dong et al. (2021), our work focuses on sparsity in the path selection between attention heads and residual connections. As for the FFN module, current foundation models often adopt the MoE structure (Fedus et al., 2022; Cai et al., 2024), where different inputs typically activate only a limited number of experts. Thus, studies on dynamic sparsity are usually conducted at the granularity of expert networks. In contrast to MoE-based approaches, our work emphasizes the dynamic sparsity within the FFN module itself, exploring sparsity at the granularity of individual neurons similar to Voita et al. (2023).

B More details of Experiment Setting

More details of data generating: To trigger the model to output the probability of the first character, we introduce a special symbol "#" as the start of the sequence, just similar to <BOS>. We define $\boldsymbol{s}_0 =$"#" and $\boldsymbol{p}_{\boldsymbol{s}_0} = 1$. Then, we sample from a uniform distribution to obtain $\boldsymbol{p}(\boldsymbol{s}_1|\boldsymbol{s}_0)$ for generating the first character. When generating the subsequent conditional probabilities $\boldsymbol{p}(\boldsymbol{s}_i|\boldsymbol{s}_{<i})$, we follow the sparsity principle: that

is, we randomly select two characters from the vocabulary and assign them probabilities of 0.8 and 0.2 respectively, while the probabilities of other characters are all set to 0. Additionally, it should be noted that the complexity of calculating the learned $\boldsymbol{p_\theta}$ is exponential, that is, $|\mathcal{V}|^n$. Therefore, the vocabulary size and sequence length cannot be set too large. In the main body of this paper, we fix the vocabulary size $|\mathcal{V}| = 5$ (not including "#") and the sequence length $n = 5$, which results in a total of $|\mathcal{V}|^n = 3125$ possible sequences. Then, under the approach of the sparsity principle, only $|\mathcal{V}| \cdot 2^{n-1} = 80$ sequences will have non-zero probabilities. These sequences form the non-sparse part, while the remaining constitute the sparse part as in Section 3.1. It should be noted that there may be slight differences between the sampled true distribution and the generated target distribution. Therefore, to better compare with the real distribution, we use the sampled distribution as the target distribution $\boldsymbol{p}_{\text{tgt}}$ when calculating the entropy $\mathrm{H}(\boldsymbol{p}_{\text{tgt}})$ and KL divergence $\mathrm{KL}(\boldsymbol{p}_{\text{tgt}}|\boldsymbol{p_\theta})$.

In the main body, we have $\mathrm{H}(\boldsymbol{p}_{\text{tgt}}) = 3.571$. In addition to generating the target distribution as described above, we also test different distributions by adjusting the values of conditional probabilities: **(a) Lower-entropy distribution:** we still sample from a uniform distribution to generate $\boldsymbol{p}(\boldsymbol{s}_1|\boldsymbol{s}_0)$ while when generating subsequent conditional distributions, we randomly select two characters and assigned them probabilities of 0.9 and 0.1 while the remaining are set to 0, resulting in $\boldsymbol{p}_{\text{tgt}} = 2.864$; **(b) Higher-entropy distribution:** we randomly selected three characters and assigned them probabilities of 0.6, 0.3, and 0.1 respectively, which ultimately resulted in $\boldsymbol{p}_{\text{tgt}} = 5.160$. Our conclusions still hold for these cases which can be seen in Appendix D.1 and D.2 respectively.

More details of model settings: We select two classic RNN models: GRU (Chung et al., 2014) and LSTM (Hochreiter and Schmidhuber, 1997), both implemented using PyTorch's default modules (torch.nn.GRU and torch.nn.LSTM). For RNNs, we find that a single layer is sufficient while stacking too many layers may lead to optimization issues under our controlled experimental setup. Therefore, we use a single-layer configuration for all RNNs by default. In addition, the hidden size is set to be $d_h = 4d$. For Transformers, we use the decoder-only architecture similar in Vaswani et al. (2017). More specifically, we use the cosine absolute position encoding and add it to the token embeddings. In the attention module, we fix the number of attention heads at 4. In the FFN module, we use ReLU as the activation function, and the hidden size d_h is set to four times the size of model dimension, that is, $d_h = 4d$. In both the attention module and the FFN module, we apply layer normalization after the residual connection. The input size to the embedding layer is $|\mathcal{V}| + 1$ as there is an additional symbol "#" while for the final projection head, the output size of the linear layer is exactly $|\mathcal{V}|$, which is followed by Softmax to output the probabilities over $|\mathcal{V}|$ characters.

When a comparison is needed, noting that the scale of the task is relatively small, the embedding size is chosen from $\{8, 16, 32, 64\}$ meanwhile we adjust the number of layers of Transformer ($L = 5$) to make the model size comparable to RNNs under the same model dimension d (in center left and center right part of Figure 2 and Figure 3 (Left part) in section 3.1. In addition, when more model settings are required, we change the model dimension and the number of layers of the model. More specifically, in the right part of Figure 2, for RNNs, the model dimension d is selected from $\{16, 20, 24, 28, 32, 64\}$ while for Transformers, the model dimension d is selected from $\{16, 32, 64\}$ and the number of layers is from $\{3, 4, 5, 6, 8\}$. The settings of Transformer variants in the center left and right parts of Figure 3 also follow the above. The cases $d = 8$ are all excluded above for clear presentation as the learned distributions are relatively far from the target distribution when models are small and these scatter points often become outliers relative to the main region. This does not affect our conclusions because the low-entropy preference tends to gradually emerge in larger Transformers. However, when the model is small, there is still a strong linear relationship between KL and Loss. Therefore, we retained the case of $d = 8$ in the right part of Figure 3. Moreover, in Section 4.1 and 4.2, we set the layers $L = 8$ to better observe the role of residual connections and ensure a sufficient number of FFN neurons. However, we find that our conclusions still hold when $L = 5$.

More details of training settings: We sample 65,536 sequences $\boldsymbol{s}$ from the generated target distribution $\boldsymbol{p}_{\text{tgt}}$ as our training data. During training, we set the batch size to 512 and set the dropout rate to 0.1 for RNNs and the FFN modules in Transformers. All models are trained for 100 epochs using Adam (Kingma and Ba, 2014), where we set the learning rate $\mathrm{lr} = 0.001$, $\beta_1 = 0.9$ and $\beta_2 = 0.999$. As for settings of

Figure 7 in Section 4.2, the parameters of each optimizer are as follows: (a) SGD + Momentum: lr = 0.001 and Momentum = 0.9; (b) RMSprop: lr = 0.0001, $\alpha(\beta_2)$ = 0.99 and weight decay = 0.01; (c) Adam-2nd: lr = 0.0005, β_1 = 0.01 and β_2 = 0.999; (d) AdamW: lr = 0.001, β_1 = 0.01, β_2 = 0.999 and weight decay = 0.01. We observe that during training, entropy and KL exhibit more pronounced fluctuations compared to loss. To more clearly present the trends and the relative levels between different models, we smooth the curves of entropy and KL changes during training by taking an average with a window size of 3. However, to better illustrate the occurrence of loss spikes, we do not apply similar smoothing to the loss curve. Additionally, when drawing scatter plots in Figure 2 and 3, for each model configuration, we average the values of the last 15 epochs where we exclude outliers caused by the occurrence of loss spikes especially for larger models. Furthermore, our experiments were conducted on two 24GB NVIDIA GeForce RTX 3090 GPUs and were completed within two days.

C More Experiment Details

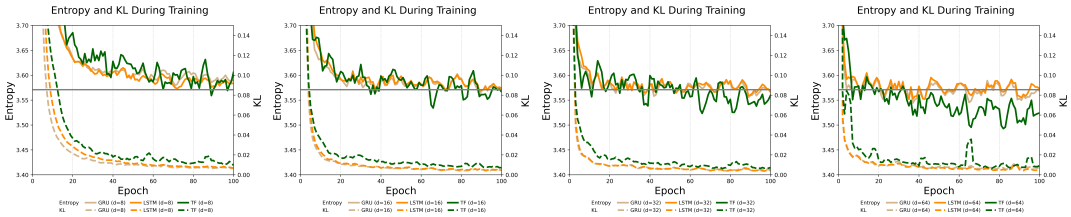

Figure 8: Entropy and KL during training for GRU, LSTM and Transformer when $d = 8, 16, 32, 64$.

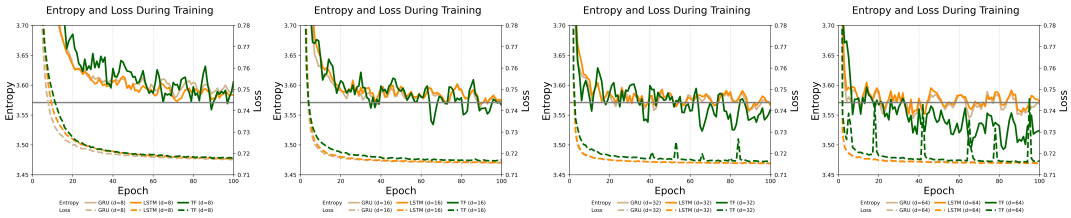

Figure 9: Entropy and Loss during training for GRU, LSTM and Transformer when $d = 8, 16, 32, 64$.

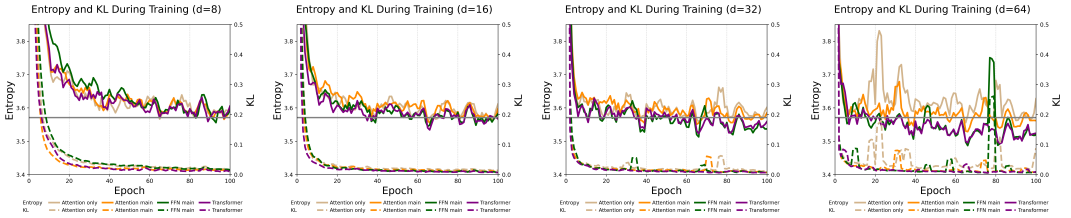

Figure 10: Entropy and KL during training for different Transformer variants when $d = 8, 16, 32, 64$.

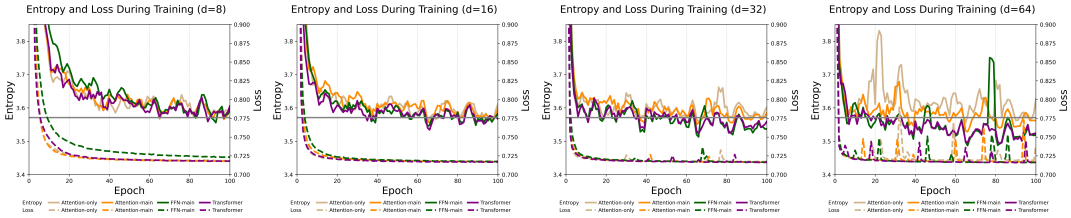

Figure 11: Entropy and Loss during training for different Transformer variants when $d = 8, 16, 32, 64$.

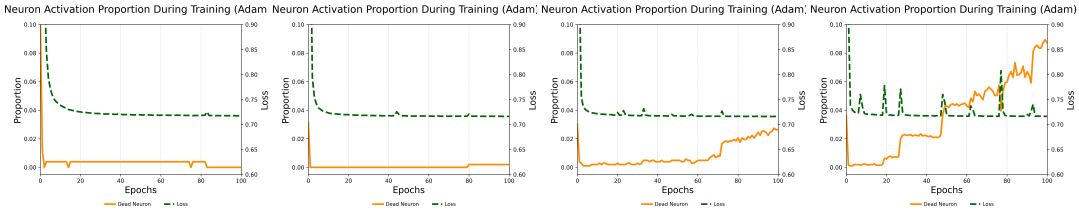

Figure 12: Loss and proportion of dead neurons during training when $d = 8, 16, 32, 64$ (left to right).

D More Results on Different Distributions

D.1 More results on the distribution with lower entropy

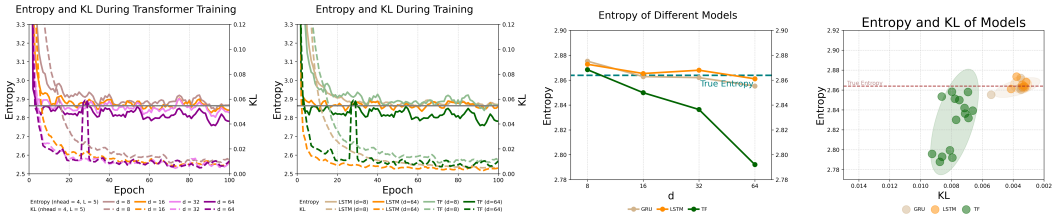

Figure 13: **Left:** Entropy and KL During Training for Transformers of Different Sizes. **Center Left:** Entropy and KL During Training for Transformer and LSTM when $d = 8$ and $d = 64$. **Center Right:** The change in entropy with model size for GRU, LSTM and Transformer, averaged over the last 15 epochs. **Right:** The relationship between entropy and KL for different model configurations.

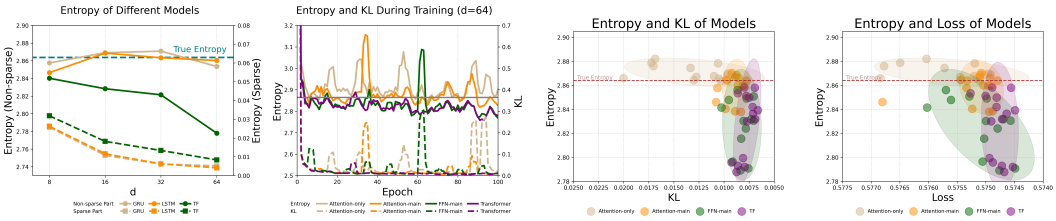

Figure 14: **Left:** The change in entropy of the sparse and non-sparse parts with model size. **Center Left:** Entropy and KL during training for different Transformer variants. **Center Right and Right:** Relationship between KL/Loss and Entropy for different variants. FFN-main can achieve lower entropy compared to Attention-only and Attention-main.

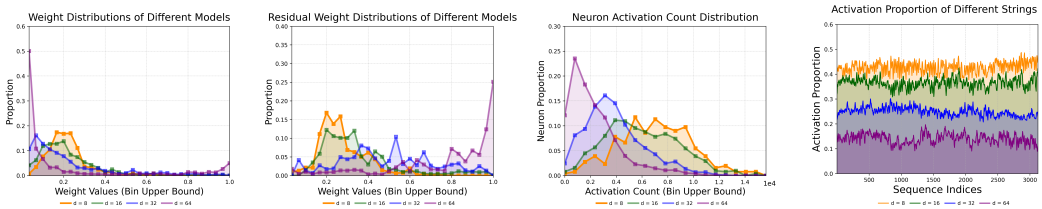

Figure 15: **Left:** The weight distribution of all paths for different model sizes. **Center Left:** The weight distribution of only residual paths for different model sizes. **Center Right:** The distribution of neuron activation counts for Transformers of different sizes. **Right:** The proportion of neurons activated across different input sequences.

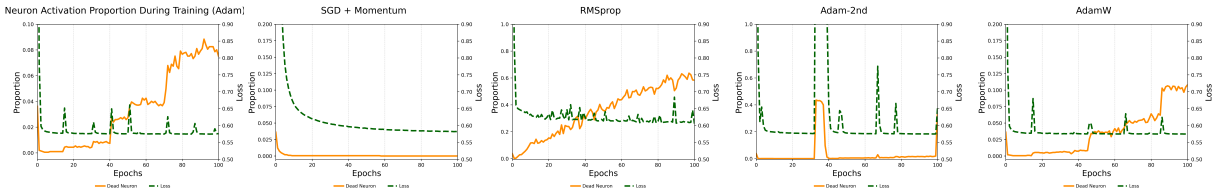

Figure 16: Loss and the proportion of dead neurons during training under different optimizers. We use **Adam** as the optimizer by default. **SGD with momentum** does not cause loss spikes or an increase in the proportion of dead neurons. **RMSprop** exhibits unstable training and a higher proportion of dead neurons. **Adam primarily relying on second-order gradients** causes more severe loss spikes and greater fluctuations in the dead neurons proportion compared to regular Adam while **AdamW** shows behavior similar to Adam.

D.2 More results on the distribution with higher entropy

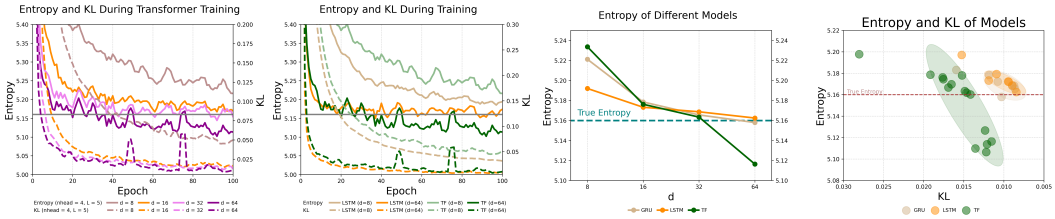

Figure 17: **Left:** Entropy and KL During Training for Transformers of Different Sizes. **Center Left:** Entropy and KL During Training for Transformer and LSTM when $d = 8$ and $d = 64$. **Center Left:** The change in entropy with model size for GRU, LSTM and Transformer, averaged over the last 15 epochs. **Center Right:** The relationship between entropy and KL for different model configurations. For Transformers, **the five scatters at the bottom that are far below the true entropy exactly correspond to cases when** $d = 64$.

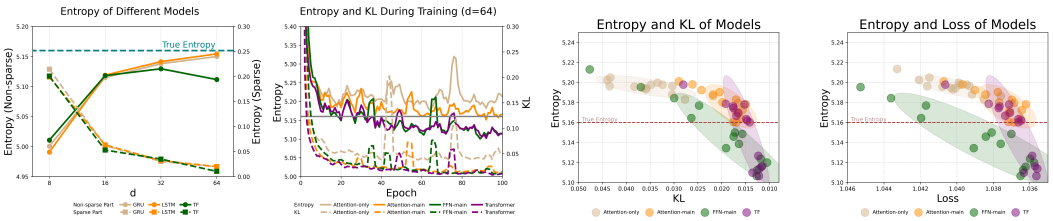

Figure 18: **Left:** The change in entropy of the sparse and non-sparse parts with model size. **Center Left:** Entropy and KL during training for different Transformer variants. **Center Right and Right:** Relationship between KL/Loss and Entropy for different variants. FFN-main can achieve lower entropy compared to Attention-only and Attention-main.

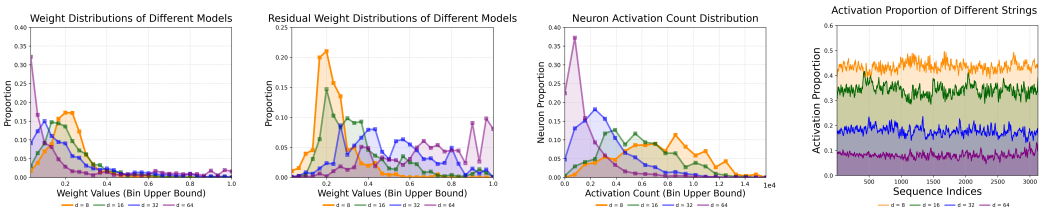

Figure 19: **Left:** The weight distribution of all paths for different model sizes. **Center Left:** The weight distribution of only residual paths for different model sizes. **Center Right:** The distribution of neuron activation counts for Transformers of different sizes. **Right:** The proportion of neurons activated across different input sequences.

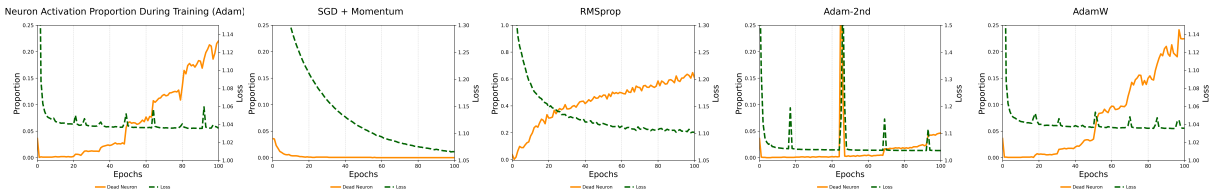

Figure 20: Loss and the proportion of dead neurons during training under different optimizers. We use **Adam** as the optimizer by default. **SGD with momentum** does not cause loss spikes or an increase in the proportion of dead neurons. **RMSprop** exhibits unstable training and a higher proportion of dead neurons. **Adam primarily relying on second-order gradients** causes more severe loss spikes and greater fluctuations in the dead neurons proportion compared to regular Adam while **AdamW** shows behavior similar to Adam.