SPARSELY GATED TINY LINEAR EXPERTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Sparsity allows scaling model parameters without proportionally increasing the size of active circuits. While mixture of experts models are made increasingly sparse, individual experts typically remain large and dense. Here, we demonstrate that further increasing sparsity by shrinking each expert to consist of a single neuron and selecting a tiny fraction of many available neurons can improve mechanistic interpretability *and* compute-matched performance. Counterintuitively, the key to achieving both is removing the nonlinearity typically applied to the experts, resulting in a network of sparsely gated *linear* neurons (`sgatlin`). In an isoflop comparison, we find that replacing all transformer feedforward layers with `sgatlin` improves perplexity in language models across different compute budgets. Crucially, the sparsity and linearity of the resulting feedforward circuits simplifies mechanistic analyses. In a small-scale case study, we demonstrate that the sparse feedforward circuits in `sgatlin` can be interpreted without having to train additional replacement models (e.g., sparse autoencoders). We find that feedforward circuits live in a semantically structured metric space and can be selectively intervened on to probe their causal roles. Our findings paint a possible path towards interpretable transformers without sacrificing modelling quality.

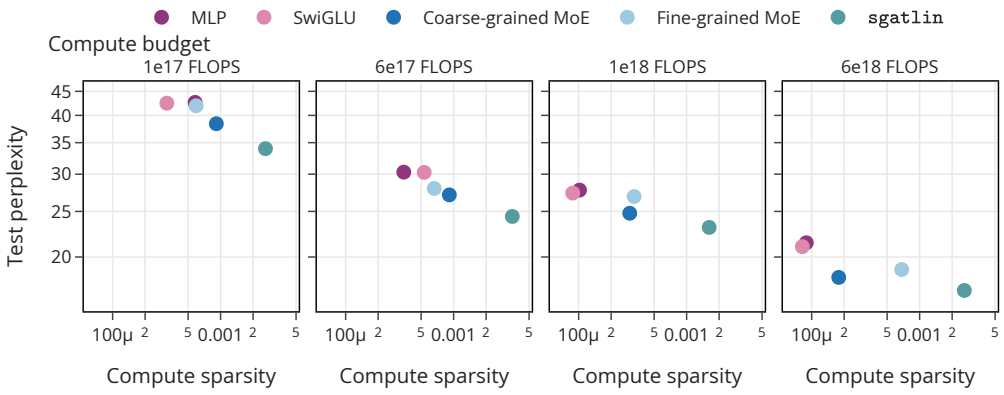

Figure 1: **Feedforward sparsity improves compute-matched perplexity while shrinking active circuits.** Comparing feedforward architectures in transformer-based language models pretrained on SlimPajama 627B in a FLOP-matched setting reveals that increasing compute sparsity – the ratio of parameters per FLOP – improves compute-optimal test perplexity while reducing the size of active circuits. Our sparsely gated linear neuron layer (`sgatlin`) further increases this ratio.

1 INTRODUCTION

Scale is a key ingredient for creating capable transformer-based models and has played an important part in their success. At the same time, it demands large amounts of computational resources and yields complex and opaque models that are difficult to understand. Reconciling scale with compute efficiency and model interpretability is a major challenge.

Feedforward layers in particular consume a considerable amount of the total compute of transformer models as their scale increases. Since feedforward layers are thought to be the main repository of knowledge, increasing their number of parameters is an important scaling axis. However, scaling the number of parameters in classical dense feedforward layers leads to a proportional increase in computational cost and the large number of active parameters presents an obstacle for model

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

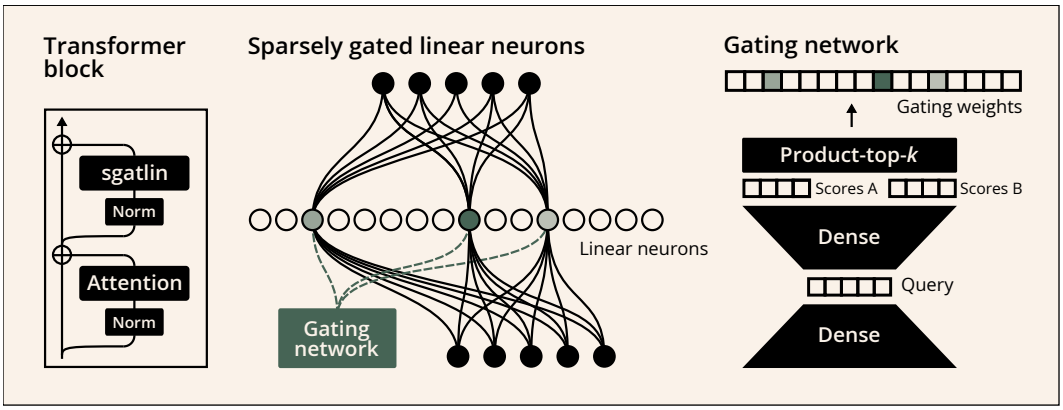

Figure 2: **Sparsely gated linear neurons layer.** ***Left*** We replace the feedforward layer within each transformer block with a `sgatlin` layer. ***Center*** In `sgatlin`, a gating network selects a sparse subset of k linear neurons whose outputs are linearly combined. ***Right*** The gating network can efficiently select among a large number of neurons by combining a bottleneck with an efficient top-k operation over the product set of two sets of scores.

interpretability, a challenge compounded by nonlinearities that preclude analysis using standard linear algebra tools.

Sparsity can lower scale-induced computational cost and complexity: By activating only a subset of parameters, it reduces the compute requirements of parameter scaling and helps localize the subcircuits causally implicated in a given computation (Shazeer et al., 2017; Krajewski et al., 2024; Potapczynski et al., 2024; Arora et al., 2026). However, while mixture of experts (MoE) models are popular and increasingly sparse in current large-scale architectures, they continue to activate billions of parameters per token (Muennighoff et al., 2024; OpenAI et al., 2025; Google, 2026; Team, 2025; GLM-5-Team et al., 2026; DeepSeek-AI, 2026).

Here, we take a step towards scalable feedforward layers that further realize the potential of compute sparsity by drastically increasing the number of parameters per FLOP. In our biggest models, less than 0.1% of feedforward layer parameters are activated per token. We build on prior work by He (2024) that demonstrated how equipping MoEs with an efficient routing mechanism allows scaling parameters at increasing sparsity with fine-grained experts. By removing the element-wise nonlinearity applied to the experts, eliminating expert parameter sharing across channels, and further decreasing the computational cost of the router, we obtain a layer of sparsely gated linear neurons (`sgatlin`) that improves language modelling perplexity in a compute-matched setting and opens avenues for model interpretability that do not rely on training additional replacement models.

We summarize our **main contributions** in the following:

- In Section 2, we present `sgatlin`, a new feedforward layer which sparsely gates a large number of linear neurons and can scale the number of parameters at sublinear time complexity.
- In Section 3, we compare `sgatlin` to standard feedforward architectures in transformer-based language models. In an isoflop comparison with models of up to 4B parameters, we find that `sgatlin` consistently outperforms baselines across compute budgets.
- In Section 4, we conduct an interpretability case study, demonstrating that the feedforward circuits of `sgatlin` form a semantically structured metric space and can be selectively causally intervened on to probe their causal roles.

2 Sparsely gated linear neurons

We now introduce our sparsely gated linear neurons layer (`sgatlin`), which we will use to replace the feedforward layer in each transformer block (see Figure 2 for an overview). It consists of two components, a gating network that creates a sparse vector of gating weights, and a large pool of linear neurons – the experts – that are linearly combined according to the gating weights.

The gating network computes scores for all available neurons and selects the k largest ones,

$$\boldsymbol{g} = \sigma(\texttt{product-top-k}(\boldsymbol{W}^{\text{key}}\boldsymbol{W}^{\text{query}}\boldsymbol{z})), \tag{1}$$

where $\boldsymbol{W}^{\text{query}} \in \mathbb{R}^{d_{\text{key}} \times d_{\text{model}}}$ and $\boldsymbol{W}^{\text{key}} \in \mathbb{R}^{2\sqrt{d_{\text{ffw}}} \times d_{\text{key}}}$. The $\texttt{product-top-k}(\boldsymbol{x}) : \mathbb{R}^{2\sqrt{d_{\text{ffw}}}} \to \mathbb{R}^{d_{\text{ffw}}}$ operation (Lample et al., 2019) selects the k largest entries of the product set of two sets of scalars, obtained by slicing the input vector, $\boldsymbol{x} \in \mathbb{R}^{2\sqrt{d_{\text{ffw}}}}$, into two parts of equal size, and returns a sparse vector of gating weights with exactly k non-zero entries. Applying the gating function $\sigma(\cdot)$ yields the gating vector $\boldsymbol{g} \in \mathbb{R}^{d_{\text{ffw}}}$ with $\|\boldsymbol{g}\|_0 = k$. $\boldsymbol{g} \in \mathbb{R}^{d_{\text{ffw}}}$ with $\|\boldsymbol{g}\|_0 = k$.
The gating weights are then used to linearly combine the output of a sparse subset of k linear neurons,

$$\texttt{sgatlin}(\boldsymbol{z}) = \sum_{i:g_i \neq 0} g_i \boldsymbol{w}_i^{\text{out}} \boldsymbol{w}_i^{\text{in}\top} \boldsymbol{z}, \tag{2}$$

where $\boldsymbol{w}_i^{\text{in}}, \boldsymbol{w}_i^{\text{out}} \in \mathbb{R}^{d_{\text{model}}}$. The full `sgatlin` layer sums over C parallel channels, each of which has distinct sets of parameters except for a shared query projection. To keep the exposition in the main text concise, we defer the full equations for multiple channels to Appendix A. The resulting layer has several notable properties:

Given the gating weights, the effective circuit processing the layer input is linear. We do not apply a nonlinear activation to the neurons (see Section 3.2 for an ablation study). Given the gating weights, the effective circuit processing the layer input is therefore purely linear. Specifically, each neuron i consists of a rank-one matrix, $\boldsymbol{W}_i^{\text{neuron}} := \boldsymbol{w}_i^{\text{out}} \boldsymbol{w}_i^{\text{in}\top}$, and each token is effectively processed by a (at most) rank-k matrix, $\boldsymbol{W}^{\text{circuit}}(\boldsymbol{z}) = \sum_{i:g_i \neq 0} g_i \boldsymbol{w}_i^{\text{out}} \boldsymbol{w}_i^{\text{in}\top}$.

Scaling the number of available neurons only incurs a sublinear increase in time complexity. Since the effective circuit uses a fixed number of neurons (we use $k = 8$ in all experiments), increasing the number of available neurons only affects the time complexity of the gating network. By keeping the key dimension small and constant (we use $d_{\text{key}} = 128$ in all experiments), the time complexity of the gating network scales sublinearly in the number of available neurons, $\mathcal{O}(\sqrt{d_{\text{ffw}}})$. We compare the time and memory complexities of `sgatlin` to other feedforward layer architectures in Table 2.

3 Language modelling

In this section, we evaluate decoder-only transformer models in a standard autoregressive language modelling task where we replace the feedforward layers in all transformer blocks. We will begin by comparing different feedforward layer choices in a compute optimal setting, followed by an ablation study on various architectural components in `sgatlin`.

3.1 IsoFLOP model comparison

We evaluate how the choice of transformer feedforward layer affects language modelling performance in a compute-matched (isoflop) comparison where we systematically vary model size and the number of training tokens as a function of the compute budget. We consider four different compute budgets, $\{1 \times 10^{17}, 6 \times 10^{17}, 1 \times 10^{18}, 6 \times 10^{18}\}$ FLOPs, for each of which we train models of multiple sizes on the SlimPajama text dataset consisting of 627B tokens (Soboleva et al., 2023) using a context window size of 2048 tokens. We compare `sgatlin` to both dense feedforward layers, a standard GeLU MLP (Hendrycks & Gimpel, 2023) and a SwiGLU layer (Shazeer, 2020), as well as two types of mixture of experts layers, the coarse-grained MoE layer used in GPT-OSS (OpenAI et al., 2025) and the fine-grained PEER MoE layer (He, 2024).
We use standard causal, decoder-only transformers and vary model size using the scaling ladder defined in Appendix B.2. We optimize the cross-entropy loss using the AdamW optimizer (Loshchilov & Hutter, 2019) with a warmup-stable-decay learning rate scheduler (Hu et al., 2024; Hägele et al., 2024; Wen et al., 2025) that linearly warms the learning rate from zero for 1000 steps, followed by a stable learning rate of 1×10^{-3} and a final square-root decay over the last 20% of training tokens back to zero. The full set of hyperparameters and the procedure for selecting them are provided in Appendix B.1.
Figure 3 compares the test perplexities of the different feedforward layer types after training on a given compute budget and model size. Consistent with prior work (e.g., Fedus et al., 2022; Artetxe et al., 2022; Muennighoff et al., 2024; Lin et al., 2024), we find that MoE models are more compute optimal than dense architectures. The relationship between feedforward compute sparsity and performance

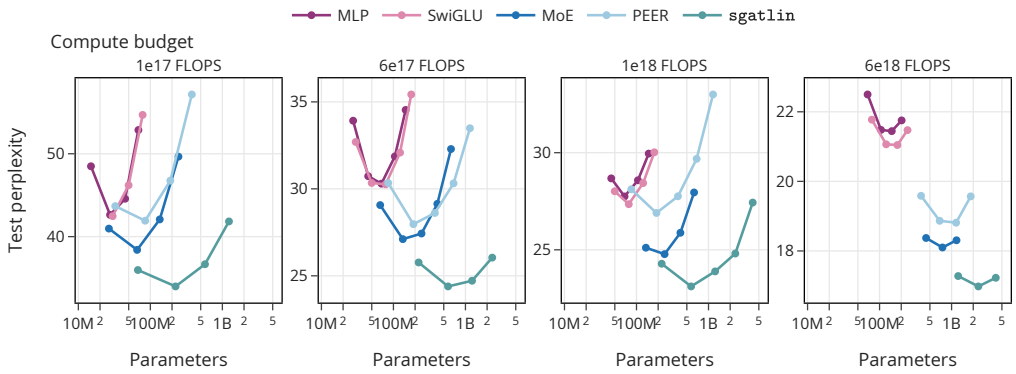

Figure 3: **Sparsely gated linear neurons improve language modelling performance.** Isoflop comparison of transformer-based language models trained on SlimPajama across four different compute budgets. We compare the choice of transformer feedforward layer type by training models of varying size for a given compute budget and report the resulting perplexities on the test set.

becomes evident in Figure 1, where we plot the test perplexity of the best performing model size per compute budget and feedforward type as a function of the number of parameters per FLOP. `sgatlin` has a particularly high compute sparsity and consistently achieves the best test perplexity across compute budgets.

3.2 Ablation Study

The absence of a nonlinear activation function in `sgatlin` might appear counterintuitive given that it is a standard component in transformer feedforward layers. For instance, MoE layers typically apply an elementwise nonlinear activation function to the experts. We run an ablation study to verify how adding such an activation function in `sgatlin` affects language modelling performance in the 6×10^{18} FLOPs compute budget using 1.2B-parameter models. In Table 1, we report the test perplexity of adding either a ReLU, GeLU, or Swish activation function to `sgatlin` and find that each choice leads to a decrease in modelling performance. These findings suggest that the top-k gating by itself suffices to enable nonlinear processing in `sgatlin`. We additionally ablate our gating network, swapping it with the router used in PEER and varying the gating function. Despite our gating networks' reduced time and memory complexity, it leads to comparable performance. We find that Softmax gating outperforms the identity gating function we used in Section 3.1.

Table 1: **Model ablations.** Effect of adding a nonlinearity to the neurons, replacing the gating network with the router from PEER (He, 2024), or varying the gating function on language modelling performance of `sgatlin`.

	Ablation	Test perplexity
$+$	ReLU activation	21.0880
$+$	GeLU activation	20.8445
$+$	Swish activation	20.6424
$\leftrightarrow$	PEER router	18.2964
$\varnothing$	Identity gating	18.1002
$+$	Sigmoid gating	18.0210
$+$	Softmax gating	**17.8226**

3.3 Downstream Performance

We next verify whether our findings generalize to a different training distribution and translate into downstream performance. For this purpose, we replicate our experimental setup in the `nanochat` codebase (Karpathy, 2025), pretraining on the ClimbMix (Diao et al., 2025) dataset, and evaluating on the Core Evals benchmark suite (Li et al., 2024). To allow us to increase the compute budget to 10^{19} FLOPs per model, we fix network depth to 14 layers and scale the remaining architectural parameters according to our scaling ladder in Appendix B.2. As shown in Figure 4, the modelling improvements of `sgatlin` previously observed on SlimPajama carry over to this new setting, yielding corresponding gains in downstream task accuracy. Furthermore, practical prefill and decoding throughput benchmarks (Figure 4, right) demonstrate that the compute sparsity of `sgatlin` enables running larger models at speeds comparable to smaller dense baselines. Notably, the throughput advantage of `sgatlin` over PEER empirically confirms the theoretical time complexity benefits of our gating network.

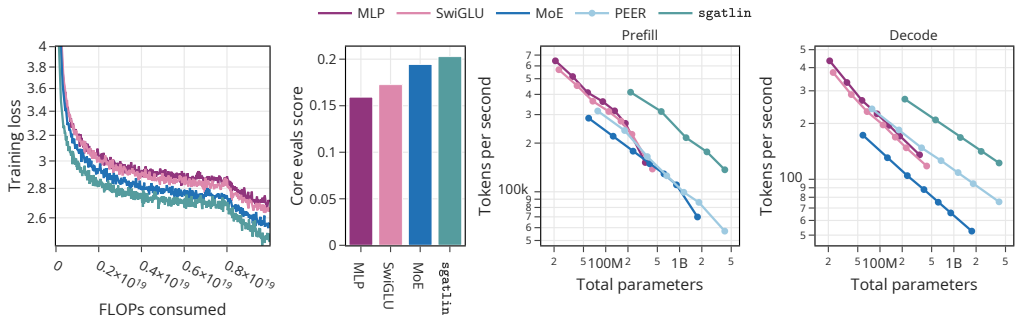

Figure 4: **Sparsely gated linear neurons improve downstream performance.** ***Left*** Pretraining loss on ClimbMix and downstream *Core Evals* performance given a compute budget of 10^{19} FLOPs comparing different transformer feedforward architectures. ***Right*** Inference prefill and decoding throughput on an NVIDIA H100 GPU (1792 prompt tokens, 128 decoded tokens).

4 Interpretability

The structure of `sgatlin` has interesting implications for model interpretability that we will explore in the following section:

> For each position in the sequence `sgatlin` linearly combines a small number of *linear* neurons, effectively applying a **sparse, linear circuit**. Are these feedforward circuits interpretable?

To keep this analysis tractable, we will consider a small, `sgatlin`-based language model trained on the TinyStories corpus (Eldan & Li, 2023), a collection of short stories for small children which use simple language. Despite the simplicity of this dataset, models trained on it have been shown to produce coherent and diverse stories with proper grammatical and semantic structure (Eldan & Li, 2023). We use the same training pipeline as for our larger models above, albeit with a small, custom tokenizer with a vocabulary size of 8192, to train a language model with four layers and a context window size of 1024 tokens. The full training details can be found in Appendix B.1.3.

4.1 Neuron subpopulations form reusable feedforward circuits

We begin by exploring whether the feedforward circuits activated across novel sequences are reused in semantically similar contexts. For this purpose, we collect the gating weights at each layer as well as the corresponding input tokens for a random subset of 128 unseen stories from the TinyStories validation set. Each gating weight is a d_{ffw}-dimensional vector with k non-zero entries. To visualize the collection of gating weights, we embed them into a two-dimensional space using UMAP (Sainburg et al., 2021) equipped with the cosine distance. The result of this procedure is shown in Figure 5 for the penultimate feedforward layer, filtered by the 100 most frequent input tokens in the random subset of unseen stories. A corresponding plot for all four feedforward layers of the model is shown in Figure 10 in the appendix. Intriguingly, the resulting embeddings have discernible structure with distinct clusters of varying size. Some – but not all – of the structure is related to the input tokens that activated the corresponding circuits. To illustrate this point, we partially colour points in the two-dimensional embedding according to the input tokens. The colours are chosen to group semantically related input tokens. This reveals, for instance, that the larger clusters in the outer perimeter of the plot are made up of circuits that activate for different punctuation tokens. The different tenses of the verb "to be" are similarly grouped together. Particularly interesting is that names of the characters in the stories group together in the vicinity of pronouns. If we track these groups across the different layers in Figure 10, we find that after the penultimate layer, pronouns and names are processed with similar circuits, suggesting that the corresponding circuits operate on the semantic role of these tokens. Overall, these findings indicate that the gating network systematically combines neuron subpopulations into reusable circuits that are semantically organized.

4.2 Feedforward circuits live in a semantically structured metric space

How pervasive is the semantic organization of circuits observed above? In this next part, we attempt to quantify the semantic regularity of the metric space induced by the gating weights. Specifically,

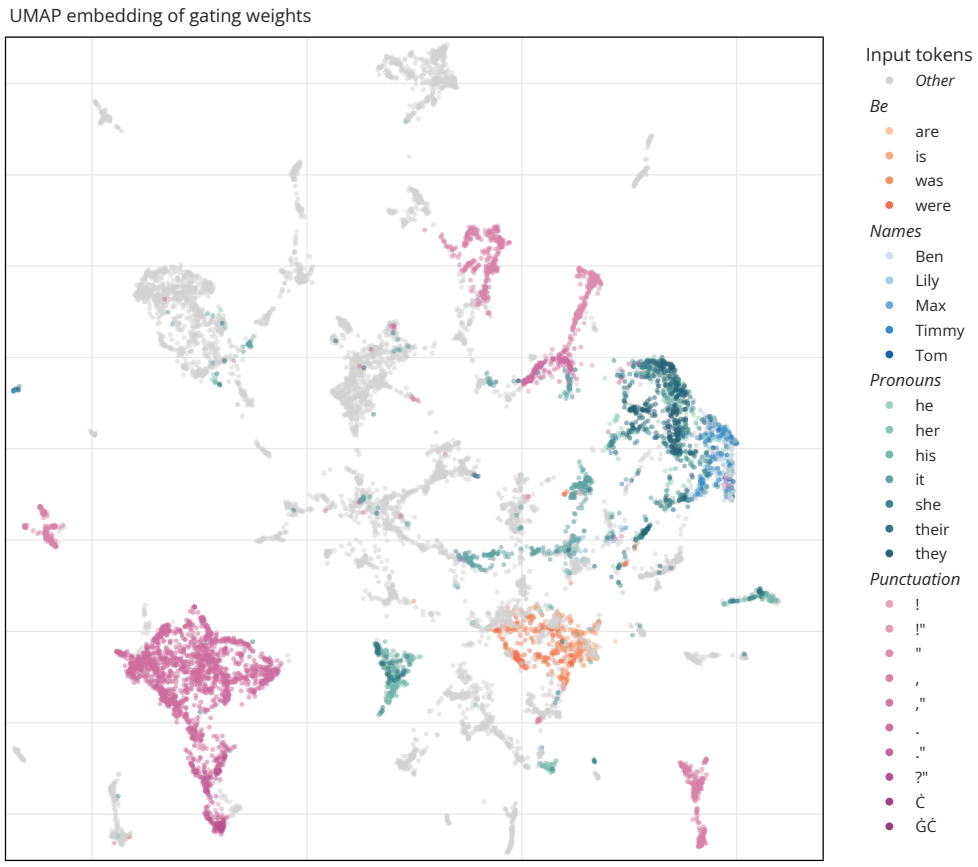

Figure 5: **Feedforward circuits form semantic clusters.** We collect the gating weights of `sgatlin` of the penultimate layer of a language model trained on TinyStories for the 100 most frequent tokens and embed them into 2D space using UMAP. The resulting embedding forms clusters that are partially explained by the semantics of the corresponding input tokens. For instance, names used in the stories cluster together and pronouns form clusters in their vicinity. An extended version of this figure showing the remaining layers can be found in Figure 10.

we will evaluate how predictive a feedforward circuit's local gating neighbourhood is of the context within which it activates. For this purpose, we collect the gating weights corresponding to the input tokens from unseen sequences in the TinyStories validation set and store them in a per-layer database alongside the corresponding input sequence and top output token predictions, as outlined in Figure 6. This circuit catalogue forms a reference set to which we can compare a novel circuit to obtain a candidate explanation of its function. We then use the pipeline shown in Figure 7 to verify whether the closest circuit neighbours for a new target circuit allow to create a discriminative circuit explanation. We run this pipeline for 100 target circuits from held-out sequences for each of the four layers, retrieving the 8 nearest neighbours per target circuit and using Gemini 3.8 Flash through the official API to generate circuit explanations and solve the circuit discrimination quizzes. This results in discrimination accuracies of 0.99, 0.98, 0.99 and 0.99 for the four layers, respectively. This finding confirms that the circuit space is indeed highly semantically regular, making it possible to infer a novel circuit's behavioural signature by comparing it to a reference circuit catalogue.

4.3 Feedforward circuits are causally implicated in factual recall

So far we have focused on observational behavioural signatures of the feedforward circuits. Next, we will examine how causal interventions on the feedforward circuits impact model predictions. Generally speaking, feedforward layers in transformers are believed to function as the primary store of a model's knowledge about the world. We can use feedforward circuit interventions to assess where such factual knowledge is represented in the feedforward circuits. Notably, since `sgatlin`

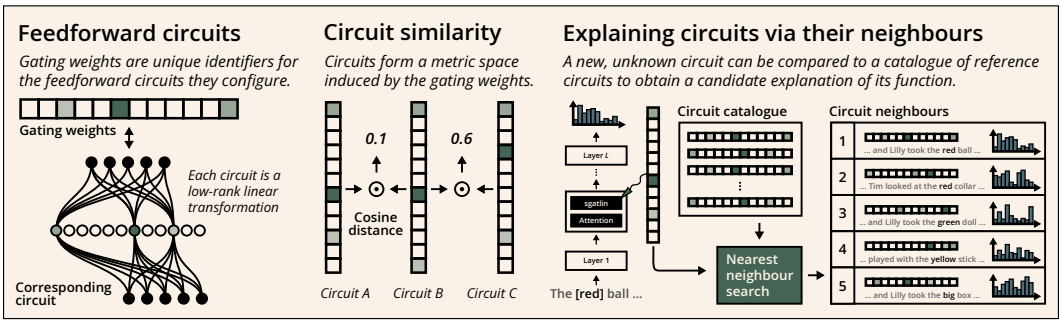

Figure 6: **Feedforward circuit analysis.** The effective feedforward circuits that process information at a given position in `sgatlin` are uniquely determined by their sparse gating weights and we can calculate circuit similarity as the distance between them. We use this property to analyse a given feedforward circuit by comparing it to similar circuits. For this purpose, we collect a circuit catalogue over a set of reference sequences and retrieve the nearest neighbours from this catalogue to obtain a candidate explanation for unseen circuits.

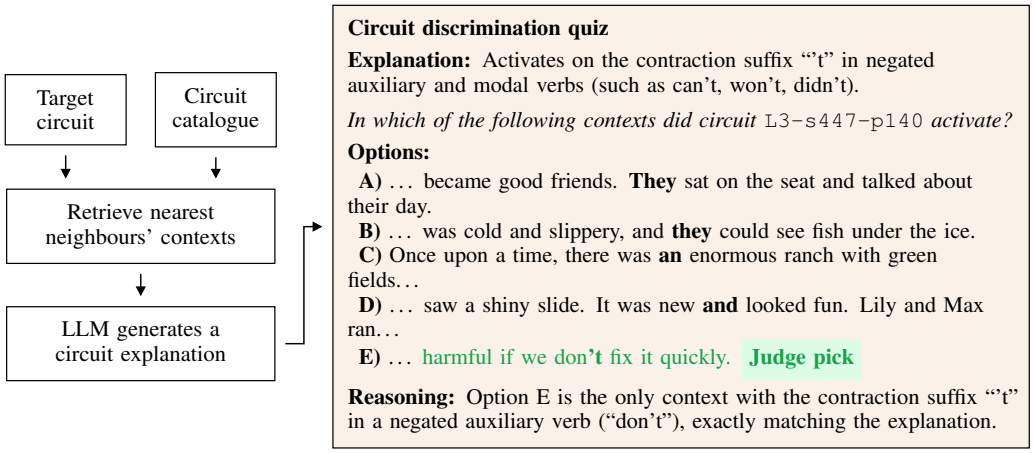

Figure 7: **Autointerpretation pipeline to evaluate semantic regularity of the circuit space.** Given a target circuit, we retrieve its nearest neighbours from a reference circuit catalogue and instruct an LLM to generate an explanation of when it activates soley based on the neighbours' contexts. We then evaluate whether the explanation allows an LLM judge to determine the target circuit's context in a multiple-choice quiz.

explicitly separates the circuit formation from its application, we can selectively intervene on a circuit without altering the intermediate data it produces.

To ensure the knowledge we probe for exists in TinyStories and we can reasonably expect the model to have learned it, we extract frequently co-occurring noun-adjective pairs that match the syntactic template "`The [noun] was [adjective]`" (e.g., "*The turtle was slow*") from the TinyStories training set. We select the largest set of noun-adjective pairs that ensures no two nouns have the same target adjective. For the causal intervention, we then create pairs of prompts "`The [noun] was`" to serve as clean and counterfactual inputs $(x_{\text{clean}}, x_{\text{patch}})$ with divergent nouns and target adjectives. We can intervene on the forward pass of the clean input by patching the feedforward circuit with the gating weights from the counterfactual input at different token positions and layers in the model.

We quantify the causal influence of a feedforward circuit, identified by the gating weights $\boldsymbol{g}$, with respect to a pair of input sequences, x_{clean} and x_{patch}, using the normalized indirect effect (Pearl, 2001). Given the corresponding clean and counterfactual target tokens, y_{clean} and y_{patch}, we measure the logit difference for an input x as $m(x) = \text{logit}(y_{\text{clean}} \mid x) - \text{logit}(y_{\text{patch}} \mid x)$. The normalized indirect effect can then be computed as

$$\text{NIE}(m; \boldsymbol{g}; x_{\text{clean}}, x_{\text{patch}}) = \frac{m(x_{\text{clean}} \mid \text{do}(\boldsymbol{g} = \boldsymbol{g}_{\text{patch}})) - m(x_{\text{clean}})}{m(x_{\text{patch}}) - m(x_{\text{clean}})},$$

where $\text{do}(\boldsymbol{g} = \boldsymbol{g}_{\text{patch}})$ denotes the do-intervention, corresponding to clamping the gating weights at specific layers and token positions to the counterfactual gating weights.

We report the normalized indirect effect on the logit difference between the clean and counterfactual adjectives in Figure 8. The overall effects we observe are of moderate magnitude, potentially reflecting that the factoids probed for are not only encoded in the feedforward layers but also in other parts of the network such as the input/output embeddings. Nevertheless, we find that patching the feedforward circuits at the position of the noun has a stronger impact on the normalized indirect effect than patching at the final token prior to the prediction, suggesting that associated factoids are retrieved right when the noun is encountered. When patching individual layers, the effect is strongest for the second layer of the network, although the outsized effect of patching all layers indicates that feedforward circuits across layers work in conjunction to retrieve a particular factoid.

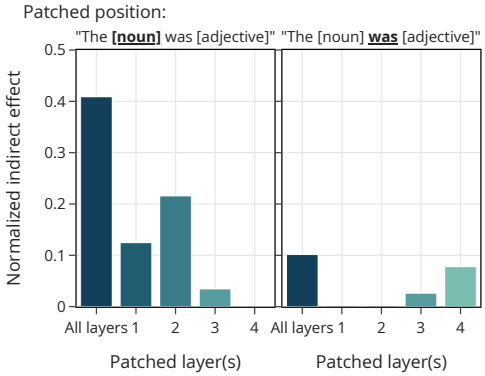

Figure 8: **Causal intervention on feedforward circuits.** Normalized indirect effect of patching layers at different depth and position in the sequence on counterfactual knowledge pairs extracted from the TinyStories dataset.

5 Related work

5.1 Feedforward layers

The feedforward network architectures we have considered here can be categorized as dense, where all parameters are activated in each forward pass, or mixture of experts (MoE), where only a subset of parameters are active per token. Interestingly, `sgatlin` can be conceptually related to both.

The original transformer uses dense ReLU multilayer perceptrons (Vaswani et al., 2023), which implicitly gate the subset of non-negative neurons. While many transformers replace the ReLU activation with a GeLU due to better modelling performance (Radford et al., 2018), the activation sparsity implicitly induced by ReLU might be useful to improve inference efficiency (Mirzadeh et al., 2023). We can think of `sgatlin` as a multilayer perceptron with a special kind of input-dependent nonlinearity that guarantees a fixed activation sparsity. A similar comparison can be drawn to the dense SwiGLU feedforward layer (Shazeer, 2020), in which a second linear input transformation with an ensuing nonlinearity (e.g. sigmoid) is used to softly gate the intermediate neurons via a component-wise product (Dauphin et al., 2017). In a similar vein, `sgatlin` can be interpreted as hard gating via a component-wise binary mask of fixed sparsity.

As outlined in Section 2, `sgatlin` can be equally motivated from the perspective of mixture of experts (MoE) layers that select a subset of experts, each of which is a full feedforward network. Classically, such MoE layers were coarse-grained with large experts, replicating dense layers (Fedus et al., 2022). More recently, there has been a trend towards fine-grained MoEs with many, smaller experts (Krajewski et al., 2024). PEER takes an extreme stance in terms of granularity, making experts as small as possible, with each expert consisting of a single neuron (He, 2024). `sgatlin` is closely related to PEER, albeit with several important architectural changes: In contrast to PEER, `sgatlin` uses distinct expert parameters for each channel and removes the activation function applied to the experts as well as the softmax nonlinearity applied to the gating scores.

5.2 Interpretability

Understanding the internal mechanisms of large-scale transformer-based models is a pressing area of active research given their widespread deployment. The prevailing approach attempts to automatically extract sparse, interpretable circuits that faithfully characterize a particular behaviour (Wang et al., 2022; Marks et al., 2024; Ameisen et al., 2025). Typically, this involves training replacement models that substitute components of the original model with more interpretable counterparts. Replacement models such as sparse autoencoders, transcoders, or crosscoders emulate the original model using high-dimensional and sparse representations (Cunningham et al., 2023; Dunefsky et al., 2024; Lindsey et al., 2024). Their shared goal is to decompose the likely polysemantic features of the original model into more easily interpretable, monosemantic features (Bricken et al., 2023). However, in practice, training such models is costly, surrogate features often only imperfectly approximate the original

model (Jacovi & Goldberg, 2020; Gurnee, 2024; Engels et al., 2024), and sparsity constraints can be hard to reconcile with hierarchical structure, leading to uninterpretable features (Bricken et al., 2023; Chanin et al., 2025).

An alternative line of work therefore attempts to identify interpretable structure directly in the model under investigation without relying on a replacement model (Gao et al., 2026). Arora et al. (2026) demonstrate that gradient-based attribution can in several instances successfully extract sparse circuits directly from dense feedforward layers. Herbst et al. (2026) show that as MoEs become more sparsely activated, expert neurons tend to become less polysemantic. These findings suggest that it may be possible to modify the model architecture itself in order to improve its interpretability. While the idea of modifying the architecture to improve interpretability is not new, and several attempts have been made to this extent (e.g., Elhage et al., 2022a; Sharkey, 2023), such modifications typically lead to a loss in modelling performance or scalability and thus struggle to be adopted. Our approach is conceptually very similar to building a sparse autoencoder into the architecture from the outset (trained with fixed-sparsity routing rather than sparsity-inducing regularization), all the while maintaining good modelling quality.

6 Discussion

We have introduced `sgatlin`, a new feedforward architecture that can serve as a simple drop-in replacement for the feedforward layer in transformers to improve interpretability through high levels of sparsity. In an isoflop comparison, we found improved language modelling performance across scales and validated key architectural decisions in an ablation study. The sparse and linear feedforward circuits configured by `sgatlin` promise to simplify model interpretability by removing the need to train separate replacement models such as sparse autoencoders. We analyse the feedforward circuits in a small-scle case study, finding that feedforward circuits form a semantically structured metric space and can be selectively intervened on to investigate their causal implication in factual recall.

Beyond interpreting feedforward circuits, replacing the feedforward layers with `sgatlin` might enable new approaches for understanding the inner workings of transformers as a whole. Similar to how the attention mechanism can be decomposed into a key-query circuit that computes the attention pattern and an output-value circuit that applies a linear transformation to the inputs (Elhage et al., 2021), `sgatlin` can be decomposed into the gating circuit that configures a linear feedforward circuit, which in turn processes the inputs. This means that conditioned on a particular input's fixed gating weights, attention patterns and layer normalization statistics, the full transformer forward pass becomes effectively a single, large affine transformation followed by the final softmax (a log-linear model). Linear networks and linearized transformer models in particular have served as important analytical devices and advanced deep learning theory, but ultimately have to concede with simplifying assumptions (Saxe et al., 2014; 2022; Elhage et al., 2022b). In contrast, the input-conditioned linear decomposition of the transformer enabled by `sgatlin` is exact and might open new directions for bridging the gap between theory and practice.

Limitations

Here, we focused on the implications of compute sparsity in just one component of the transformer, namely the feedforward layers. Investigating how compute sparsity could improve interpretability of other dense components, such as the attention mechanism, is an important avenue for future work. We dedicated much of our effort on investigating how `sgatlin` performance scales and compares to other feedforward architectures, in order to establish whether it is a viable replacement in terms of modelling quality for informative interpretability studies. An exciting next step is to develop whole-transformer interpretability tools that make use of the sparsity and linearity of `sgatlin` to analyse full transformer circuits and scale these efforts to larger models.

Reproducibility statement

To ensure the reproducibility of our work, we are providing the code for all our experiments as part of the supplementary material. We further expand upon the experimental details explained in the main text in Appendix B.1.

AI use statement

In this work, we used generative AI tools to design or provide feedback on research methodology or experiments, implement methods, and assist with translation. We have not used generative AI tools to help develop theoretical models or conceptual frameworks, formulate mathematical claims,

propose or refine hypotheses, support qualitative and thematic data analysis, and interpret results. ”Generate synthetic data sets, assist in the writing of proofs, provide critical ingredients for proving mathematical claims, and clean and reformat dataset” are not applicable to this work. Additionally, we used generative AI tools to create or modify scientific figures or images, suggest experimental parameters, create or edit software code, summarize or analyse existing literature, discover research topics or identify gaps, brainstorming, sourcing/searching for information, edit a research paper to improve readability, and identify relevant literature. We have reviewed all AI-assisted work: All code is carefully tested and every single line of code written is reviewed by a human before being committed. Any literature suggested via AI is manually retrieved from the source and any claims thereof validated in the primary text. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Emmanuel Ameisen, Jack Lindsey, Adam Pearce, Wes Gurnee, Nicholas L. Turner, Brian Chen, Craig Citro, David Abrahams, Shan Carter, Basil Hosmer, Jonathan Marcus, Michael Sklar, Adly Templeton, Trenton Bricken, Callum McDougall, Hoagy Cunningham, Thomas Henighan, Adam Jermyn, Andy Jones, Andrew Persic, Zhenyi Qi, T. Ben Thompson, Sam Zimmerman, Kelley Rivoire, Thomas Conerly, Chris Olah, and Joshua Batson. Circuit Tracing: Revealing Computational Graphs in Language Models, 2025.

Aryaman Arora, Zhengxuan Wu, Jacob Steinhardt, and Sarah Schwettmann. Language Model Circuits Are Sparse in the Neuron Basis, January 2026. URL http://arxiv.org/abs/2601.22594. arXiv:2601.22594 [cs].

Mikel Artetxe, Shruti Bhosale, Naman Goyal, Todor Mihaylov, Myle Ott, Sam Shleifer, Xi Victoria Lin, Jingfei Du, Srinivasan Iyer, Ramakanth Pasunuru, Giridharan Anantharaman, Xian Li, Shuohui Chen, Halil Akin, Mandeep Baines, Louis Martin, Xing Zhou, Punit Singh Koura, Brian O'Horo, Jeffrey Wang, Luke Zettlemoyer, Mona Diab, Zornitsa Kozareva, and Veselin Stoyanov. Efficient Large Scale Language Modeling with Mixtures of Experts. In Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang (eds.), *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 11699–11732, Abu Dhabi, United Arab Emirates, December 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.emnlp-main.804. URL https://aclanthology.org/2022.emnlp-main.804/.

Igor Babuschkin, Kate Baumli, Alison Bell, Surya Bhupatiraju, Jake Bruce, Peter Buchlovsky, David Budden, Trevor Cai, Aidan Clark, Ivo Danihelka, Antoine Dedieu, Claudio Fantacci, Jonathan Godwin, Chris Jones, Ross Hemsley, Tom Hennigan, Matteo Hessel, Shaobo Hou, Steven Kapturowski, Thomas Keck, Iurii Kemaev, Michael King, Markus Kunesch, Lena Martens, Hamza Merzic, Vladimir Mikulik, Tamara Norman, George Papamakarios, John Quan, Roman Ring, Francisco Ruiz, Alvaro Sanchez, Rosalia Schneider, Eren Sezener, Stephen Spencer, Srivatsan Srinivasan, Wojciech Stokowiec, Luyu Wang, Guangyao Zhou, and Fabio Viola. The DeepMind JAX Ecosystem, 2020. URL http://github.com/deepmind.

Lukas Biewald. Experiment Tracking with Weights and Biases, 2020. URL https://www.wandb.com/.

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL http://github.com/google/jax.

Trenton Bricken, Adly Templeton, Joshua Batson, Brian Chen, Adam Jermyn, Tom Conerly, Nicholas L. Turner, Cem Anil, Carson Denison, Amanda Askell, Robert Lasenby, Yifan Wu, Shauna Kravec, Nicholas Schiefer, Tim Maxwell, Nicholas Joseph, Alex Tamkin, Karina Nguyen, Brayden McLean, Josiah E. Burke, Tristan Hume, Shan Carter, Tom Henighan, and Chris Olah. Towards Monosemanticity: Decomposing Language Models With Dictionary Learning, 2023. URL https://transformer-circuits.pub/2023/monosemantic-features.

David Chanin, James Wilken-Smith, Tomáš Dulka, Hardik Bhatnagar, Satvik Golechha, and Joseph Isaac Bloom. A is for Absorption: Studying Feature Splitting and Absorption in Sparse Autoencoders. October 2025. URL https://openreview.net/forum?id=R73ybUciQF.

Hoagy Cunningham, Aidan Ewart, Logan Riggs, Robert Huben, and Lee Sharkey. Sparse Autoencoders Find Highly Interpretable Features in Language Models, October 2023. URL http://arxiv.org/abs/2309.08600. arXiv:2309.08600 [cs].

Yann N. Dauphin, Angela Fan, Michael Auli, and David Grangier. Language Modeling with Gated Convolutional Networks. In *Proceedings of the 34th International Conference on Machine Learning*, pp. 933–941. PMLR, July 2017. URL https://proceedings.mlr.press/v70/dauphin17a.html.

DeepSeek-AI. DeepSeek-V4: Towards Highly Efficient Million-Token Context Intelligence, 2026. URL https://huggingface.co/deepseek-ai/DeepSeek-V4-Flash.

Shizhe Diao, Yu Yang, Yonggan Fu, Xin Dong, Dan SU, Markus Kliegl, ZIJIA CHEN, Peter Belcak, Yoshi Suhara, Hongxu Yin, Mostofa Patwary, Yingyan (Celine) Lin, Jan Kautz, and Pavlo Molchanov. Nemotron-CLIMB: Clustering-based Iterative Data Mixture Bootstrapping for Language Model Pre-training. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, Main Conference. Curran Associates, Inc., 2025. doi: 10.52202/085713-0865. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/24a8968affe71ffe4067d022b9d16566-Paper-Datasets_and_Benchmarks_Track.pdf.

Jacob Dunefsky, Philippe Chlenski, and Neel Nanda. Transcoders Find Interpretable LLM Feature Circuits, November 2024. URL http://arxiv.org/abs/2406.11944. arXiv:2406.11944 [cs].

Ronen Eldan and Yuanzhi Li. TinyStories: How Small Can Language Models Be and Still Speak Coherent English?, May 2023. URL http://arxiv.org/abs/2305.07759. arXiv:2305.07759 [cs].

Nelson Elhage, Neel Nanda, Catherine Olsson, Tom Henighan, Nicholas Joseph, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, Nova DasSarma, Dawn Drain, Deep Ganguli, Zac Hatfield-Dodds, Danny Hernandez, Andy Jones, Jackson Kernion, Liane Lovitt, Kamal Ndousse, Dario Amodei, Tom Brown, Jack Clark, Jared Kaplan, Sam McCandlish, and Chris Olah. A Mathematical Framework for Transformer Circuits, 2021. URL https://transformer-circuits.pub/2021/framework/index.html.

Nelson Elhage, Tristan Hume, Catherine Olsson, Neel Nanda, Tom Henighan, Scott Johnston, Sheer El Showk, Nicholas Joseph, Nova DasSarma, Ben Mann, Danny Hernandez, Amanda Askell, Kamal Ndousse, Andy Jones, Dawn Drain, Anna Chen, Yuntao Bai, Deep Ganguli, Liane Lovitt, Zac Hatfield-Dodds, Jackson Kernion, Tom Conerly, Shauna Kravec, Stanislav Fort, Saurav Kadavath, Josh Jacobson, Eli Tran-Johnson, Jared Kaplan, Jack Clark, Tom Brown, Sam McCandlish, Dario Amodei, and Christopher Olah‡ AFFILIATION Anthropic PUBLISHED June 27. Softmax Linear Units, 2022a. URL https://transformer-circuits.pub/2022/solu/index.html.

Nelson Elhage, Tristan Hume, Catherine Olsson, Nicholas Schiefer, Tom Henighan, Shauna Kravec, Zac Hatfield-Dodds, Robert Lasenby, Dawn Drain, Carol Chen, Roger Grosse, Sam McCandlish, Jared Kaplan, Dario Amodei, Martin Wattenberg, and Christopher Olah. Toy Models of Superposition, 2022b. URL https://transformer-circuits.pub/2022/toy_model/index.html.

Joshua Engels, Logan Riggs Smith, and Max Tegmark. Decomposing The Dark Matter of Sparse Autoencoders. *Transactions on Machine Learning Research*, December 2024. ISSN 2835-8856. URL https://openreview.net/forum?id=sXq3Wb3vef.

William Fedus, Barret Zoph, and Noam Shazeer. Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity. *Journal of Machine Learning Research*, 23(120):1–39, 2022. ISSN 1533-7928. URL http://jmlr.org/papers/v23/21-0998.html.

Yutong Gao, Qinglin Meng, Yuan Zhou, and Liangming Pan. Towards Intrinsic Interpretability of Large Language Models:A Survey of Design Principles and Architectures, April 2026. URL http://arxiv.org/abs/2604.16042. arXiv:2604.16042 [cs].

GLM-5-Team, Aohan Zeng, Xin Lv, Zhenyu Hou, Zhengxiao Du, Qinkai Zheng, Bin Chen, Da Yin, Chendi Ge, Chenghua Huang, Chengxing Xie, Chenzheng Zhu, Congfeng Yin, Cunxiang Wang, Gengzheng Pan, Hao Zeng, Haoke Zhang, Haoran Wang, Huilong Chen, Jiajie Zhang, Jian Jiao, Jiaqi Guo, Jingsen Wang, Jingzhao Du, Jinzhu Wu, Kedong Wang, Lei Li, Lin Fan, Lucen Zhong, Mingdao Liu, Mingming Zhao, Pengfan Du, Qian Dong, Rui Lu, Shuang-Li, Shulin Cao, Song Liu, Ting Jiang, Xiaodong Chen, Xiaohan Zhang, Xuancheng Huang, Xuezhen Dong, Yabo Xu, Yao Wei, Yifan An, Yilin Niu, Yitong Zhu, Yuanhao Wen, Yukuo Cen, Yushi Bai, Zhongpei Qiao, Zihan Wang, Zikang Wang, Zilin Zhu, Ziqiang Liu, Zixuan Li, Bojie Wang, Bosi Wen, Can Huang, Changpeng Cai, Chao Yu, Chen Li, Chengwei Hu, Chenhui Zhang, Dan Zhang, Daoyan Lin, Dayong Yang, Di Wang, Ding Ai, Erle Zhu, Fangzhou Yi, Feiyu Chen, Guohong Wen, Hailong

Sun, Haisha Zhao, Haiyi Hu, Hanchen Zhang, Hanrui Liu, Hanyu Zhang, Hao Peng, Hao Tai, Haobo Zhang, He Liu, Hongwei Wang, Hongxi Yan, Hongyu Ge, Huan Liu, Huanpeng Chu, Jia'ni Zhao, Jiachen Wang, Jiajing Zhao, Jiamin Ren, Jiapeng Wang, Jiaxin Zhang, Jiayi Gui, Jiayue Zhao, Jijie Li, Jing An, Jing Li, Jingwei Yuan, Jinhua Du, Jinxin Liu, Junkai Zhi, Junwen Duan, Kaiyue Zhou, Kangjian Wei, Ke Wang, Keyun Luo, Laiqiang Zhang, Leigang Sha, Liang Xu, Lindong Wu, Lintao Ding, Lu Chen, Minghao Li, Nianyi Lin, Pan Ta, Qiang Zou, Rongjun Song, Ruiqi Yang, Shangqing Tu, Shangtong Yang, Shaoxiang Wu, Shengyan Zhang, Shijie Li, Shuang Li, Shuyi Fan, Wei Qin, Wei Tian, Weining Zhang, Wenbo Yu, Wenjie Liang, Xiang Kuang, Xiangmeng Cheng, Xiangyang Li, Xiaoquan Yan, Xiaowei Hu, Xiaoying Ling, Xing Fan, Xingye Xia, Xinyuan Zhang, Xinze Zhang, Xirui Pan, Xu Zou, Xunkai Zhang, Yadi Liu, Yandong Wu, Yanfu Li, Yidong Wang, Yifan Zhu, Yijun Tan, Yilin Zhou, Yiming Pan, Ying Zhang, Yinpei Su, Yipeng Geng, Yong Yan, Yonglin Tan, Yuean Bi, Yuhan Shen, Yuhao Yang, Yujiang Li, Yunan Liu, Yunqing Wang, Yuntao Li, Yurong Wu, Yutao Zhang, Yuxi Duan, Yuxuan Zhang, Zezhen Liu, Zhengtao Jiang, Zhenhe Yan, Zheyu Zhang, Zhixiang Wei, Zhuo Chen, Zhuoer Feng, Zijun Yao, Ziwei Chai, Ziyuan Wang, Zuzhou Zhang, Bin Xu, Minlie Huang, Hongning Wang, Juanzi Li, Yuxiao Dong, and Jie Tang. GLM-5: from Vibe Coding to Agentic Engineering, February 2026. URL http://arxiv.org/abs/2602.15763. arXiv:2602.15763 [cs].

Google. Welcome Gemma 4: Frontier multimodal intelligence on device, April 2026. URL https://huggingface.co/blog/gemma4.

Wes Gurnee. SAE reconstruction errors are (empirically) pathological — AI Alignment Forum, March 2024. URL https://www.alignmentforum.org/posts/rZPiuFxESMxCDHe4B/sae-reconstruction-errors-are-empirically-pathological.

Xu Owen He. Mixture of A Million Experts, July 2024. URL http://arxiv.org/abs/2407.04153. arXiv:2407.04153.

Jonathan Heek, Anselm Levskaya, Avital Oliver, Marvin Ritter, Bertrand Rondepierre, Andreas Steiner, and Marc van Zee. Flax: A neural network library and ecosystem for JAX, 2023. URL http://github.com/google/flax.

Dan Hendrycks and Kevin Gimpel. Gaussian Error Linear Units (GELUs), 2023. URL https://arxiv.org/abs/1606.08415. _eprint: 1606.08415.

Jeremy Herbst, Jae Hee Lee, and Stefan Wermter. The Expert Strikes Back: Interpreting Mixture-of-Experts Language Models at Expert Level, April 2026. URL http://arxiv.org/abs/2604.02178. arXiv:2604.02178 [cs].

Shengding Hu, Yuge Tu, Xu Han, Ganqu Cui, Chaoqun He, Weilin Zhao, Xiang Long, Zhi Zheng, Yewei Fang, Yuxiang Huang, Xinrong Zhang, Zhen Leng Thai, Chongyi Wang, Yuan Yao, Chenyang Zhao, Jie Zhou, Jie Cai, Zhongwu Zhai, Ning Ding, Chao Jia, Guoyang Zeng, Dahai Li, Zhiyuan Liu, and Maosong Sun. MiniCPM: Unveiling the Potential of Small Language Models with Scalable Training Strategies. August 2024. URL https://openreview.net/forum?id=3X2L2TFr0f.

Alexander Hägele, Elie Bakouch, Atli Kosson, Loubna Ben Allal, Leandro Von Werra, and Martin Jaggi. Scaling Laws and Compute-Optimal Training Beyond Fixed Training Durations. November 2024. URL https://openreview.net/forum?id=Y13gSfTjGr.

Plotly Technologies Inc. Collaborative data science, 2015. URL https://plot.ly. Place: Montreal, QC.

Alon Jacovi and Yoav Goldberg. Towards Faithfully Interpretable NLP Systems: How Should We Define and Evaluate Faithfulness? In Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel Tetreault (eds.), *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 4198–4205, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.386. URL https://aclanthology.org/2020.acl-main.386/.

Andrej Karpathy. nanochat: The best ChatGPT that $100 can buy, 2025. URL https://github.com/karpathy/nanochat.

Jakub Krajewski, Jan Ludziejewski, Kamil Adamczewski, Maciej Pióro, Michał Krutul, Szymon Antoniak, Kamil Ciebiera, Krystian Król, Tomasz Odrzygóźdź, Piotr Sankowski, Marek Cygan, and Sebastian Jaszczur. Scaling Laws for Fine-Grained Mixture of Experts, February 2024. URL http://arxiv.org/abs/2402.07871. arXiv:2402.07871 [cs].

Guillaume Lample, Alexandre Sablayrolles, Marc'Aurelio Ranzato, Ludovic Denoyer, and Hervé Jégou. Large Memory Layers with Product Keys, December 2019. URL http://arxiv.org/abs/1907.05242. arXiv:1907.05242.

Quentin Lhoest, Albert Villanova del Moral, Yacine Jernite, Abhishek Thakur, Patrick von Platen, Suraj Patil, Julien Chaumond, Mariama Drame, Julien Plu, Lewis Tunstall, Joe Davison, Mario Šaško, Gunjan Chhablani, Bhavitvya Malik, Simon Brandeis, Teven Le Scao, Victor Sanh, Canwen Xu, Nicolas Patry, Angelina McMillan-Major, Philipp Schmid, Sylvain Gugger, Clément Delangue, Théo Matussière, Lysandre Debut, Stas Bekman, Pierric Cistac, Thibault Goehringer, Victor Mustar, François Lagunas, Alexander Rush, and Thomas Wolf. Datasets: A Community Library for Natural Language Processing. In Heike Adel and Shuming Shi (eds.), *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 175–184, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-demo.21. URL https://aclanthology.org/2021.emnlp-demo.21/.

Jeffrey Li, Alex Fang, Georgios Smyrnis, Maor Ivgi, Matt Jordan, Samir Gadre, Hritik Bansal, Etash Guha, Sedrick Keh, Kushal Arora, Saurabh Garg, Rui Xin, Niklas Muennighoff, Reinhard Heckel, Jean Mercat, Mayee Chen, Suchin Gururangan, Mitchell Wortsman, Alon Albalak, Yonatan Bitton, Marianna Nezhurina, Amro Abbas, Cheng-Yu Hsieh, Dhruba Ghosh, Josh Gardner, Maciej Kilian, Hanlin Zhang, Rulin Shao, Sarah Pratt, Sunny Sanyal, Gabriel Ilharco, Giannis Daras, Kalyani Marathe, Aaron Gokaslan, Jieyu Zhang, Khyathi Chandu, Thao Nguyen, Igor Vasiljevic, Sham Kakade, Shuran Song, Sujay Sanghavi, Fartash Faghri, Sewoong Oh, Luke Zettlemoyer, Kyle Lo, Alaaeldin El-Nouby, Hadi Pouransari, Alexander Toshev, Stephanie Wang, Dirk Groeneveld, Luca Soldaini, Pang Wei Koh, Jenia Jitsev, Thomas Kollar, Alexandros G. Dimakis, Yair Carmon, Achal Dave, Ludwig Schmidt, and Vaishaal Shankar. DataComp-LM: In search of the next generation of training sets for language models. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 14200–14282. Curran Associates, Inc., 2024. doi: 10.52202/079017-0455. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/19e4ea30dded58259665db375885e412-Paper-Datasets_and_Benchmarks_Track.pdf.

Xi Victoria Lin, Akshat Shrivastava, Liang Luo, Srinivasan Iyer, Mike Lewis, Gargi Ghosh, Luke Zettlemoyer, and Armen Aghajanyan. MoMa: Efficient Early-Fusion Pre-training with Mixture of Modality-Aware Experts, July 2024. URL https://arxiv.org/abs/2407.21770v3.

J. Lindsey, A. Templeton, J. Marcus, T. Conerly, J. Batson, and C. Olah. Sparse Crosscoders for Cross-Layer Features and Model Diffing, 2024. URL https://transformer-circuits.pub/2024/crosscoders/.

Ilya Loshchilov and Frank Hutter. Decoupled Weight Decay Regularization, January 2019. URL http://arxiv.org/abs/1711.05101. arXiv:1711.05101 [cs, math].

Samuel Marks, Can Rager, Eric J. Michaud, Yonatan Belinkov, David Bau, and Aaron Mueller. Sparse Feature Circuits: Discovering and Editing Interpretable Causal Graphs in Language Models. October 2024. URL https://openreview.net/forum?id=I4e82CIDxv.

Charlie Marsh. uv: An extremely fast Python package and project manager, written in Rust., 2024. URL https://pypi.org/project/uv/.

Iman Mirzadeh, Keivan Alizadeh, Sachin Mehta, Carlo C. Del Mundo, Oncel Tuzel, Golnoosh Samei, Mohammad Rastegari, and Mehrdad Farajtabar. ReLU Strikes Back: Exploiting Activation Sparsity in Large Language Models, 2023. URL https://arxiv.org/abs/2310.04564.

Niklas Muennighoff, Luca Soldaini, Dirk Groeneveld, Kyle Lo, Jacob Morrison, Sewon Min, Weijia Shi, Pete Walsh, Oyvind Tafjord, Nathan Lambert, Yuling Gu, Shane Arora, Akshita Bhagia, Dustin Schwenk, David Wadden, Alexander Wettig, Binyuan Hui, Tim Dettmers, Douwe Kiela, Ali Farhadi, Noah A. Smith, Pang Wei Koh, Amanpreet Singh, and Hannaneh Hajishirzi. OLMoE: Open Mixture-of-Experts Language Models, September 2024. URL https://arxiv.org/abs/2409.02060v2.

OpenAI, Haiming Bao, Lama Ahmad, Jason Ai, Sam Altman, Andy Applebaum, Edwin Arbus, Rahul K. Arora, Yu Bai, Bowen Baker, Haiming Bao, Boaz Barak, Ally Bennett, Tyler Bertao, Nivedita Brett, Eugene Brevdo, Greg Brockman, Sebastien Bubeck, Che Chang, Kai Chen, Mark Chen, Enoch Cheung, Aidan Clark, Dan Cook, Marat Dukhan, Casey Dvorak, Kevin Fives, Vlad Fomenko, Timur Garipov, Kristian Georgiev, Mia Glaese, Tarun Gogineni, Adam Goucher, Lukas Gross, Katia Gil Guzman, John Hallman, Jackie Hehir, Johannes Heidecke, Alec Helyar, Haitang Hu, Romain Huet, Jacob Huh, Saachi Jain, Zach Johnson, Chris Koch, Irina Kofman, Dominik Kundel, Jason Kwon, Volodymyr Kyrylov, Elaine Ya Le, Guillaume Leclerc, James Park Lennon, Scott Lessans, Mario Lezcano-Casado, Yuanzhi Li, Zhuohan Li, Ji Lin, Jordan Liss, Lily, Liu, Jiancheng Liu, Kevin Lu, Chris Lu, Zoran Martinovic, Lindsay McCallum, Josh McGrath, Scott McKinney, Aidan McLaughlin, Song Mei, Steve Mostovoy, Tong Mu, Gideon Myles, Alexander Neitz, Alex Nichol, Jakub Pachocki, Alex Paino, Dana Palmie, Ashley Pantuliano, Giambattista Parascandolo, Jongsoo Park, Leher Pathak, Carolina Paz, Ludovic Peran, Dmitry Pimenov, Michelle Pokrass, Elizabeth Proehl, Huida Qiu, Gaby Raila, Filippo Raso, Hongyu Ren, Kimmy Richardson, David Robinson, Bob Rotsted, Hadi Salman, Suvansh Sanjeev, Max Schwarzer, D. Sculley, Harshit Sikchi, Kendal Simon, Karan Singhal, Yang Song, Dane Stuckey, Zhiqing Sun, Philippe Tillet, Sam Toizer, Foivos Tsimpourlas, Nikhil Vyas, Eric Wallace, Xin Wang, Miles Wang, Olivia Watkins, Kevin Weil, Amy Wendling, Kevin Whinnery, Cedric Whitney, Hannah Wong, Lin Yang, Yu Yang, Michihiro Yasunaga, Kristen Ying, Wojciech Zaremba, Wenting Zhan, Cyril Zhang, Brian Zhang, Eddie Zhang, and Shengjia Zhao. gpt-oss-120b & gpt-oss-20b Model Card, 2025. URL https://arxiv.org/abs/2508.10925. _eprint: 2508.10925.

Judea Pearl. Direct and indirect effects. In *Proceedings of the Seventeenth conference on Uncertainty in artificial intelligence*, UAI'01, pp. 411–420, San Francisco, CA, USA, August 2001. Morgan Kaufmann Publishers Inc. ISBN 978-1-55860-800-9. URL https://dl.acm.org/doi/10.5555/2074022.2074073.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Andres Potapczynski, Shikai Qiu, Marc Finzi, Christopher Ferri, Zixi Chen, Micah Goldblum, Bayan Bruss, Christopher De Sa, and Andrew Gordon Wilson. Searching for Efficient Linear Layers over a Continuous Space of Structured Matrices, October 2024. URL http://arxiv.org/abs/2410.02117. arXiv:2410.02117 [cs].

Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, and others. Improving language understanding by generative pre-training, 2018.

Marvin Ritter, Ihor Indyk, Aayush Singh, Andrew Audibert, Anoosha Seelam, Camelia Hanes, Eric Lau, Jacek Olesiak, Jiyang Kang, and Xihui Wu. Grain - Feeding JAX Models, 2023. URL http://github.com/google/grain.

Tim Sainburg, Leland McInnes, and Timothy Q Gentner. Parametric UMAP Embeddings for Representation and Semisupervised Learning. *Neural Computation*, 33(11):2881–2907, 2021.

Andrew M. Saxe, James L. McClelland, and Surya Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks, 2014. URL https://arxiv.org/abs/1312.6120. _eprint: 1312.6120.

Andrew M. Saxe, Shagun Sodhani, and Sam Lewallen. The Neural Race Reduction: Dynamics of Abstraction in Gated Networks, July 2022. URL http://arxiv.org/abs/2207.10430. arXiv:2207.10430 [cs].

Lee Sharkey. A technical note on bilinear layers for interpretability, May 2023. URL http://arxiv.org/abs/2305.03452. arXiv:2305.03452 [cs].

Noam Shazeer. GLU Variants Improve Transformer, 2020. URL https://arxiv.org/abs/2002.05202. arXiv: 2002.05202.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer, January 2017. URL http://arxiv.org/abs/1701.06538. arXiv:1701.06538 [cs].

Daria Soboleva, Faisal Al-Khateeb, Robert Myers, Jacob R Steeves, Joel Hestness, and Nolan Dey. SlimPajama: A 627B token cleaned and deduplicated version of RedPajama, June 2023. URL https://huggingface.co/datasets/cerebras/SlimPajama-627B.

Qwen Team. Qwen3-Next: Towards Ultimate Training & Inference Efficiency, September 2025. URL https://huggingface.co/Qwen/Qwen3-Next-80B-A3B-Instruct.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention Is All You Need, August 2023. URL http://arxiv.org/abs/1706.03762. arXiv:1706.03762 [cs].

Kevin Wang, Alexandre Variengien, Arthur Conmy, Buck Shlegeris, and Jacob Steinhardt. Interpretability in the Wild: a Circuit for Indirect Object Identification in GPT-2 small, November 2022. URL http://arxiv.org/abs/2211.00593. arXiv:2211.00593 [cs].

Kaiyue Wen, Zhiyuan Li, Jason S. Wang, David Leo Wright Hall, Percy Liang, and Tengyu Ma. Understanding Warmup-Stable-Decay Learning Rates: A River Valley Loss Landscape View. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=m51BgoqvbP.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Transformers: State-of-the-Art Natural Language Processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 38–45, Online, October 2020. Association for Computational Linguistics. URL https://aclanthology.org/2020.emnlp-demos.6/.

Appendix

A Sparsely gated linear neuron layer **18**
A.1 Parallel channels 18
A.2 Time and memory complexity 18

B Experimental details **19**
B.1 Training details 19
B.1.1 SlimPajama 19
B.1.2 ClimbMix 20
B.1.3 TinyStories 20
B.2 Scaling ladder 20

C Additional results **20**
C.1 Neuron usage and coactivation 20
C.2 Model interpretability 21

D Additional details **21**
D.1 Compute resources 21
D.2 Software and libraries 21

A SPARSELY GATED LINEAR NEURON LAYER

A.1 PARALLEL CHANNELS

The full `sgatlin` layer consists of C parallel *channels* that are summed,

$$\boldsymbol{g}_c = \texttt{product-top-k}(\boldsymbol{W}_c^{\text{key}} \boldsymbol{W}^{\text{query}} \boldsymbol{z}),$$
$$\texttt{sgatlin}(\boldsymbol{z}) = \sum_{c=1}^{C} \sum_{i:g_{c,i}\neq 0} g_{c,i} \boldsymbol{w}_{c,i}^{\text{out}} \boldsymbol{w}_{c,i}^{\text{in}\top} \boldsymbol{z}$$

where $\boldsymbol{W}^{\text{query}} \in \mathbb{R}^{d_{\text{key}} \times d_{\text{model}}}$ is shared across layers, whereas $\boldsymbol{W}_c^{\text{key}} \in \mathbb{R}^{2\sqrt{d_{\text{ffw}}} \times d_{\text{key}}}$, $\boldsymbol{g}_c \in \mathbb{R}^{d_{\text{ffw}}}$, and $\boldsymbol{w}_{c,i}^{\text{in}}, \boldsymbol{w}_{c,i}^{\text{out}} \in \mathbb{R}^{d_{\text{model}}}$ are all channel-specific. $\texttt{product-top-k}(\boldsymbol{x}) : \mathbb{R}^{2\sqrt{d_{\text{ffw}}}} \to \mathbb{R}^{d_{\text{ffw}}}$ is the product top-k operator introduced by Lample et al. (2019) that selects the k largest entries of the product set of two sets of scalars, here obtained by slicing the input vector $\boldsymbol{x}$ into two parts of equal size, and returns a sparse vector of gating scores with k non-zero entries. Pseudocode for the layer is shown in Listing 1.

```
class SparselyGatedLinear(nn.Module):
  d_model: int
  d_ffw: int
  d_key: int
  knn: int
  n_channels: int

  @nn.compact
  def __call__(self, x):

    # Gating network
    n_keys = int(math.sqrt(self.d_ffw))
    queries = nn.Dense(features=self.d_key, use_bias=False)(x)
    similar = nn.DenseGeneral(
      features=(2, self.n_channels, n_keys),
      use_bias=False,
    )(queries)
    scores, indices = product_key_top_k(similar, k=self.knn)

    # Expert computation
    experts = self.param(
      name='experts',
      init_fn=lecun_normal,
      shape=(2, self.n_channels, self.d_ffw, self.d_model),
    )

    w_in, w_out = experts[indices]
    z = jnp.einsum('btd,btckd->btck', x, w_in)
    z = z * scores
    z = jnp.einsum('btck,btckd->btd', z, w_out)

    return z
```

Listing 1: Pseudocode for the `sgatlin` layer using JAX (Bradbury et al., 2018) and Flax-like (Heek et al., 2023) syntax.

A.2 TIME AND MEMORY COMPLEXITY

We compare the time and memory complexities of different transformer feedforward layers in Table 2. In typical dense feedforward layers, the time complexity is dominated by the product of the residual stream dimension and the feedforward dimension. Scaling the number of parameters by increasing the feedforward dimension proportionally increases computational cost. MoEs decouple the total number of feedforward neurons (i.e., the product of the number of experts and the size of each expert) from the number of active feedforward neurons by activating only a subset of experts. This allows

Table 2: **Time and memory complexities of feedforward layers**. We compare the time and memory complexities of different transformer feedforward layers considered in our experiments. d_{model} denotes the residual stream dimension, d_{ffw} is the number of feedforward neurons, n_{experts} the total number of experts, n_{active} the number of active experts routed per token, n_{channel} the number of channels, and d_{key} the bottleneck dimension of the gating network.

Feedforward variant	**Time complexity** ($\mathcal{O}$)	**Memory complexity** ($\mathcal{O}$)
MLP	$d_{\text{model}}\, d_{\text{ffw}}$	$d_{\text{model}}\, d_{\text{ffw}}$
SwiGLU	$d_{\text{model}}\, d_{\text{ffw}}$	$d_{\text{model}}\, d_{\text{ffw}}$
MoE	$d_{\text{model}}\, n_{\text{experts}} + n_{\text{active}}\, d_{\text{model}}\, d_{\text{ffw}}$	$n_{\text{experts}}\, d_{\text{model}}\, d_{\text{ffw}}$
PEER	$n_{\text{channel}}\, d_{\text{key}}(d_{\text{model}} + \sqrt{d_{\text{ffw}}}) + n_{\text{channel}}\, d_{\text{model}}\, n_{\text{active}}$	$n_{\text{channel}}\, d_{\text{key}}(d_{\text{model}} + \sqrt{d_{\text{ffw}}}) + d_{\text{ffw}}\, d_{\text{model}}$
`sgatlin`	$d_{\text{key}}(d_{\text{model}} + n_{\text{channel}}\sqrt{d_{\text{ffw}}}) + n_{\text{channel}}\, d_{\text{model}}\, n_{\text{active}}$	$d_{\text{key}}(d_{\text{model}} + n_{\text{channel}}\sqrt{d_{\text{ffw}}}) + n_{\text{channel}}\, d_{\text{ffw}}\, d_{\text{model}}$

scaling the number of feedforward parameters without proportionally increasing computational cost. However, typical MoE parameterizations are usually comparably coarse-grained, and the size of each expert is large enough for the time complexity to be dominated by the product of active experts and their size. Both PEER (He, 2024) and `sgatlin` shrink the experts to consist of a single neuron and only activate a small, fixed subset of neurons per token. In combination with their respective efficient router/gating network, this allows scaling the number of parameters while keeping the time complexity comparably small. Note that the difference in time and memory complexity between PEER and `sgatlin` is due to the gating network query projection being shared across channels in the latter but not in the former.

B Experimental details

B.1 Training details

In the following, we provide further training details for both the language modelling experiments on the SlimPajama 627B dataset (Soboleva et al., 2023) conducted in Section 3 and the model interpretability experiments based on the TinyStories dataset (Eldan & Li, 2023) conducted in Section 4.

B.1.1 SlimPajama

For our language modelling experiments on the SlimPajama 627B dataset, we use the following training setup.

Tokenizer We tokenize text input using the pretrained GPT-2 byte-pair encoding tokenizer with a vocabulary size of $50\,257$.

Model We use a causal, decoder-only transformer with an RMSNorm applied prior to the attention and feedforward block and rotary positional encoding. We use the MoE layer from GPT-OSS (OpenAI et al., 2025) with 16 experts and top-$k = 2$ routing. Both PEER and `sgatlin` use 16 channels, top-$k = 8$ gating and a gating bottleneck dimension of 128. For the main isoflop comparison, `sgatlin` uses no gating function, i.e. $\sigma(\boldsymbol{x}) = \boldsymbol{x}$, although we later found through the ablation study in Section 3.2 that this choice is suboptimal. All models are trained using mixed precision with `bfloat16` compute and `float32` master weights, upcasting attention softmax and the final output logits to `float32` for numerical stability. In addition, we use activation checkpointing (`flax.linen.remat`) to reduce memory requirements during training.

Optimizer We use a global batch size of 128 and a sequence length of 2048 tokens resulting in 262144 tokens per batch. We optimize the models using AdamW with a peak learning rate of 1×10^{-3} and apply a weight decay of 0.1, excluding scalar parameters. The learning rate follows a warmup stable square root decay schedule (Hu et al., 2024; Hägele et al., 2024) with 1000 warmup steps and a decay fraction of 0.2. Gradients are clipped to a maximum global norm of 1.0. Learning rate, weight decay, and warmup steps were selected after running a grid search over learning rate values of $\{1 \times 10^{-4}, 3 \times 10^{-4}, 1 \times 10^{-3}\}$, weight decay values of $\{0.01, 0.03, 0.1\}$, and warmup steps of $\{100, 1000\}$, training a dense SwiGLU transformer with 12 layers on a compute budget of 1×10^{18} FLOPs for each configuration.

B.1.2 CLIMBMIX

For the language modelling experiments on ClimbMix (Diao et al., 2025), we replicate the setup described above in Appendix B.1.1 in the `nanochat` (Karpathy, 2025) codebase but adapt their custom byte-level byte pair encoding tokenizer with a vocabulary size of 32 768 trained on 2B characters from the ClimbMix pretraining corpus. Following the results of our ablation study in Section 3.2, we use `sgatlin` with a $\sigma(\boldsymbol{x}) = \mathrm{softmax}(x)$ gating function.

B.1.3 TINYSTORIES

For our model interpretability experiments with the TinyStories dataset, we train `sgatlin`-based transformers with 4 layers, a residual stream dimension of 256 and 4 attention heads of dimension 64 with a batch size of 256 sequences and a sequence length of 1024 tokens. We train a custom byte-pair encoding tokenizer on the TinyStories training data with a vocabulary size of 8192. Otherwise, we reuse the same optimizer and scheduler settings as for SlimPajama, only reducing the number of warmup steps to 100 and increasing the gradient clipping norm to 2.0.

B.2 SCALING LADDER

We use the scaling ladder shown in Listing 2 to systematically scale model size as a function of depth.

```python
def scaling_ladder(n_layers: int, feedforward_type: str):
  d_model = 64 * n_layers
  d_heads = 64 if n_layers < 14 else 128

  match feedforward_type:
    case 'swiglu' | 'mlp':
      d_ffw = int(8 / 3 * d_model / 256) * 256
    case 'moe':
      d_ffw = d_model  # following GPT-OSS
    case 'peer':
      d_ffw = (32 + 12 * n_layers) ** 2
    case 'sgatlin':
      d_ffw = (16 + 6 * n_layers) ** 2
    case _:
      raise ValueError(f'{feedforward_type} unknown')

  return (d_model, d_heads, d_ffw)
```

Listing 2: Python code for scaling transformer model size with different feedforward layer types.

C ADDITIONAL RESULTS

C.1 NEURON USAGE AND COACTIVATION

In Figure 9, we show neuron usage and coactivation in the `sgatlin`-based transformer trained on TinyStories used for our model interpretability experiments.

The product top-k mechanism creates neuron subpopulations. The product top-k mechanism used in the gating network of `sgatlin` computes an input-dependent query and calculates the dot-product with a neuron-specific key. It efficiently computes the dot-product for a large number of possible neurons by defining the neuron keys as the product-set of multiple subkeys. As a result, each neuron shares its two subkeys with other neurons. This is illustrated in the left panel of Figure 9. In practice, this key sharing affects the empirically observed coactivation probability of neurons, effectively creating subpopulations of neurons that preferably coactivate with neurons they share a gating key with.

Available neurons are effectively used. We verify that the gating network effectively uses the available feedforward capacity and activates almost all the available feedforward neurons at least once across a batch of 128 randomly chosen sequences from the TinyStories validation set, as shown in the centre panel of Figure 9. The right panel further shows that neuron activation frequency is heterogeneous, with an average Gini coefficient of 0.58 across these sequences.

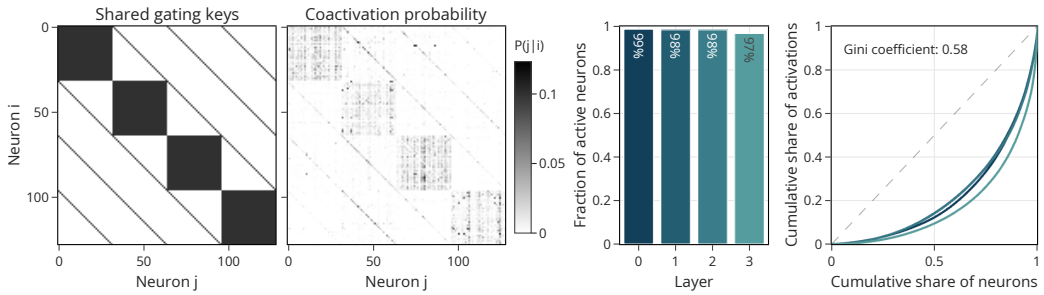

Figure 9: **Neuron usage and coactivation.** ***Left*** As a result of the product top-k mechanism, certain pairs of neurons share gating keys. This affects the empirically observed coactivation probability of neurons, where neurons preferably coactivate with neurons that they share a gating key with. Coactivation probability is computed over 128 random sequences of the TinyStories validation set and shown for the first 128 neurons of the penultimate transformer layer. ***Center*** Almost all neurons activate at least once over 128 random sequences of the TinyStories validation set. ***Right*** Neuron activation frequency is heterogeneous, with an average Gini coefficient of 0.58 across 128 random sequences of the TinyStories validation set. We control for the skewed input token frequency of TinyStories sequences considering each unique input token only once.

C.2 Model interpretability

Figure 10 shows an extended version of Figure 5 in the main text, with gating embeddings for all four layers.

D Additional details

D.1 Compute resources

We ran our experiments on two internal Slurm-based clusters equipped with nodes of 8 Nvidia H100 GPUs with 94GB of high-bandwidth memory per GPU or 8 Nvidia H200 GPUs with 141GB of high-bandwidth memory per GPU respectively. The longest of our training runs takes approximately 12 hours on a node of 8 H200 GPUs to complete. In total we spent the equivalent of around 10 000 H100 GPU hours to run our experiments.

D.2 Software and libraries

For the results obtained in this paper we build on free and open-source software. We implemented our experiments in Python using JAX (Bradbury et al., 2018, Apache License 2.0), Flax (Heek et al., 2023, Apache License 2.0), the Deepmind JAX Ecosystem (Babuschkin et al., 2020, Apache License 2.0), Grain (Ritter et al., 2023, Apache License 2.0), HuggingFace dataset (Lhoest et al., 2021, Apache License 2.0), HuggingFace tokenizers (Wolf et al., 2020, Apache License 2.0) and Scikit-learn (Pedregosa et al., 2011, BSD 3-Clause License). We utilized WandB (Biewald, 2020, MIT license) to monitor the progress and results of experiments, and Plotly (Inc, 2015, MIT license) for generating the plots. We use uv for Python project dependency management (Marsh, 2024, MIT License).

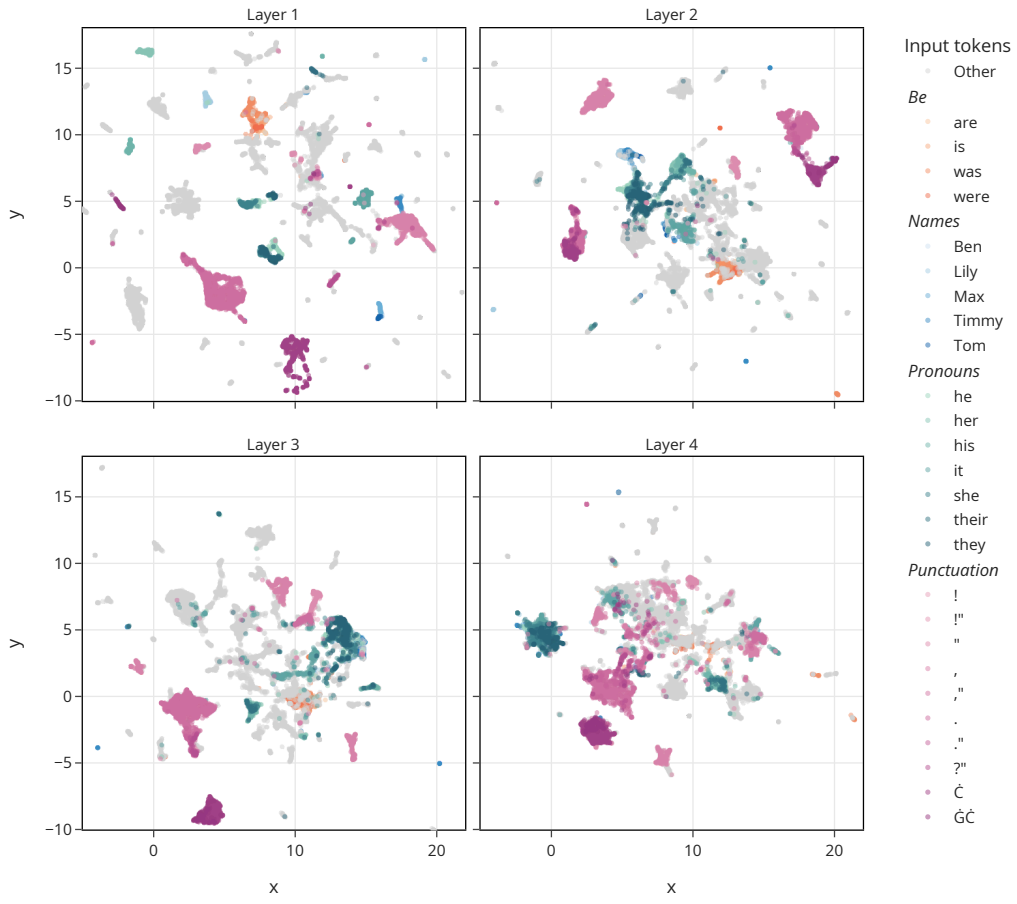

Figure 10: **Feedforward circuits form semantic clusters (extended version of Figure 5).** We collect the gating weights of `sgatlin` of a four layer language model trained on TinyStories for the 100 most frequent tokens and embed them into 2D space using UMAP. The resulting embedding forms clusters that are partially explained by the semantics of the corresponding input tokens. For instance, names used in the stories cluster together and pronouns form clusters in their vicinity.