SCALING SPARSE AUTOENCODER CIRCUITS FOR IN-CONTEXT LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Sparse autoencoders (SAEs) are a popular tool for interpreting large language model activations, but their utility in addressing open questions in interpretability remains unclear. In this work, we demonstrate their effectiveness by using SAEs to deepen our understanding of the mechanism behind in-context learning (ICL). We identify abstract SAE features that encode the model's knowledge of which task to execute and whose latent vectors causally induce the task zero-shot. This aligns with prior work showing that ICL is mediated by task vectors. We further demonstrate that these task vectors are well approximated by a sparse sum of SAE latents, including these task-execution features. To explore the ICL mechanism, we adapt the sparse feature circuits methodology of Marks et al. (2024) to work for the much larger Gemma-1 2B model, with 30 times as many parameters, and to the more complex task of ICL. Through circuit finding, we discover task-detecting features with corresponding SAE latents that activate earlier in the prompt, that detect when tasks have been performed. They are causally linked with task-execution features through the attention layer and MLP.

1 INTRODUCTION

Sparse autoencoders (SAEs; Ng (2011); Bricken et al. (2023); Cunningham et al. (2023)) are a promising method for interpreting large language model (LLM) activations. However, the full potential of SAEs in interpretability research remains to be explored, since most recent SAE research either i) interprets a single SAE's features rather than the model's computation as a whole (Bricken et al., 2023), or ii) performs high-level interventions in the model, but does not interpret the effect on the downstream computation caused by the interventions Templeton et al. (2024b). In this work, we address these limitations by interpreting in-context learning (ICL), a widely studied phenomenon in LLMs. In summary, we show that SAEs enable a) the discovery of novel circuit components (**task-detection features**; Section 4.2) and b) making existing interpretations of ICL more precise, by e.g. decomposing task vectors (Todd et al., 2024; Hendel et al., 2023) into **task-execution features** (Section 3).

In-context learning (ICL; Brown et al. (2020)) is a fundamental capability of large language models that allows them to adapt to new tasks without fine-tuning. ICL is a significantly more complex and important task than other behaviors commonly studied in circuit analysis (such as IOI in Wang et al. (2022) and Kissane et al. (2024), or subject-verb agreement and Bias-in-Bios in Marks et al. (2024)). Recent work by Todd et al. (2024) and Hendel et al. (2023) has introduced the concept of task vectors to study ICL in a simple setting, which we follow throughout this paper.[1] In short, task vectors are internal representations of tasks formed by language models during the processing of few-shot prompts, such as "hot → cold, big → small, fast → slow". These vectors can be extracted and added into different LLM forward passes to induce 0-shot task performance, making LLMs predict that "slow" follows "fast →" without explicit context. Section 2.3 provides a full introduction.

To identify **task-execution features**, we decomposed task vectors using SAEs. To achieve this, we needed to go beyond existing methods for solving the classical dictionary problem of decomposing a vector into a sparse sum of dictionary vectors (Elad, 2010). To do this, we developed a bespoke

[1]Task vectors (Hendel et al., 2023) are also called "function vectors" (Todd et al., 2024), but we use "task vectors" throughout this paper for consistency.

method for LLMs we call the TASK VECTOR CLEANING **(TVC)** algorithm. By running the TVC algorithm, we found **task-execution features**: features that can partially replace task vectors taken alone and have highly interpretable max-activating token patterns. We validate the causal relevance of these task features through a series of steering experiments on tasks, spanning several categories like translation or factual recall. The experiments demonstrate that identified task features encode crucial information about task execution, are causally implicated in the model's ICL capabilities, and can play the same role as task vectors.

We adapted the Sparse Feature Circuits (SFC) methodology of Marks et al. (2024) to work on the more complex ICL task and the larger Gemma-1 2B model (Gemma Team, 2024). This adaptation allowed us to discover and analyze the subgraph of key SAE latents involved in ICL, providing a more comprehensive view of the ICL circuit. Using this adaptation, we found **task-detection features** with SFC: features that play a crucial role in identifying the specific task being performed earlier in the prompt. Task-detection features are tightly connected with task-execution features through attention, as part of the whole ICL circuit.

Our findings not only advance our understanding of ICL mechanisms but also demonstrate the potential of SAEs as a powerful tool for interpretability research on larger language models. By unifying the task vectors view with SAEs and uncovering two of the most important causally implicated feature families behind ICL, we pave the way for future work to control and monitor ICL further, to improve either the safety or capabilities of models.

Our main contributions are as follows:

1. We demonstrate that SAEs can be effectively used to explain mechanisms behind a complex set of ICL tasks in a Gemma-1 2B, which has 10-35x more parameters than prior models typically studied at this depth in comparable, circuits-style mechanistic interpretability research (Wang et al., 2022; Marks et al., 2024). We show that causal circuit finding algorithms and SFC specifically straightforwardly scale up to larger models and SAEs with different architectures (Appendix B).
2. We identify two core bottlenecks in the ICL circuit – **task-detection features** and **task-execution features** (see Appendix C, F, G) – and study their interactions (Section 3.2). This provides new insights into how LLMs process and execute ICL tasks. Specifically, we discover task-detection features that identify the task being performed earlier in the prompt, which are then moved by attention heads to trigger task-execution features (Figure 8).
3. We present a novel transformer-specific sparse linear decomposition algorithm (Section 3.1) that decomposes task vectors (Hendel et al., 2023) into a small set of mostly task-relevant features, enabling more precise analysis of ICL mechanisms.

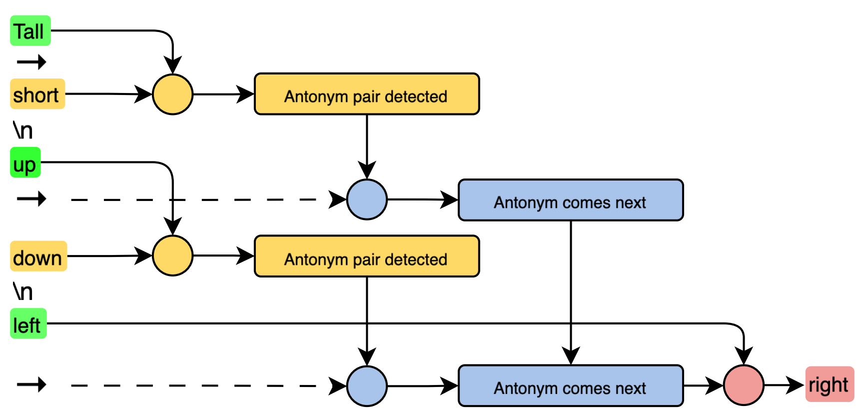


Figure 1: A diagram of the in-context learning circuit, showing task detection features (yellow) causing task execution features (blue) which cause the model to output the antonym (left → right). A more concrete circuit, along with texts these features activate on, can be seen in Figure 9.

2 Background

2.1 Sparse Autoencoders (SAEs)

Sparse autoencoders (SAEs) are neural networks designed to learn efficient representations of data by enforcing sparsity in the hidden layer activations (Elad, 2010). In the context of language model interpretability, SAEs are used to decompose the high-dimensional activations of language models into more interpretable features (Cunningham et al., 2023; Bricken et al., 2023). The basic idea behind SAEs is to train a neural network to reconstruct its input while constraining the hidden layer to have sparse activations. This process typically involves an encoder that maps the input to a sparse hidden representation, a decoder that reconstructs the input from this sparse representation, and loss task that balances reconstruction accuracy with sparsity [2]. The encoding step is as follows, with $\mathbf{f}$ denoting the pre-activation features and $\mathbf{W}_{\text{enc}}$ and $\mathbf{b}_{\text{enc}}$ the encoder weights and biases respectively:

$$\mathbf{f}(\mathbf{x}) = \sigma(\mathbf{W}_{\text{enc}}\mathbf{x} + \mathbf{b}_{\text{enc}}) \tag{1}$$

For JumpReLU SAEs (Rajamanoharan et al., 2024b), the activation function and decoder are (with H being the Heaviside step function, θ the threshold parameter and $\mathbf{W}_{\text{dec}}$/$\mathbf{b}_{\text{dec}}$ the decoder affine parameters):

$$\hat{\mathbf{x}}(\mathbf{f}) = \mathbf{W}_{\text{dec}}(\mathbf{f} \odot H(\mathbf{f} - \theta)) + \mathbf{b}_{\text{dec}} \tag{2}$$

In our work, we train SAEs on residual stream activations and attention outputs, and also train transcoders[3] on MLP layers, all of which use the improved Gated SAE architecture (Rajamanoharan et al., 2024a).

2.2 Sparse Feature Circuits

Sparse Feature Circuits (SFCs) are a methodology introduced by Marks et al. (2024) to identify and analyze causal subgraphs of sparse autoencoder features that explain specific model behaviors. This approach combines the interpretability benefits of SAEs with causal analysis to uncover the mechanisms underlying language model behavior. The SFC methodology involves several key steps:

1. Decomposing model activations into sparse features using SAEs
2. Calculating the Indirect Effect (IE, Pearl (2001) of each feature on the target behavior
3. Identifying a set of causally relevant features based on IE thresholds
4. Constructing a circuit by analyzing the connections between these features

The IE of a model component is measured by intervening on that component and observing the change in the model's output. For a component a and a metric m, the IE is defined using do-calculus (Pearl, 2009) as in Marks et al. (2024) as:

$$\text{IE}(m; a) = m(x|\text{do}(a = a')) - m(x) \tag{3}$$

Where $m(x|\text{do}(a = a'))$ represents the value of the metric when we intervene to set the value of component a to a', and $m(x)$ is the original value of the metric. In practice, attribution patching (Syed et al., 2023) is used to approximate IE, allowing for efficient computation across many components simultaneously.

SFC is described in detail in (Marks et al., 2024). We describe our modifications in Appendix E.

[2]Typically, the L_1 penalty on activations is used (Bricken et al., 2023) with some modifications (Rajamanoharan et al., 2024a; Conerly et al., 2024), although there are alternatives: Rajamanoharan et al., 2024b; Farrell, 2024; Riggs & Brinkman, 2024.

[3]Transcoders are a modification of SAEs that take MLP input and convert it into MLP output instead of trying to reconstruct the residual stream.

2.3 Task Vectors

Continuing from the high-level description in Section 1, task vectors were independently discovered by Hendel et al. (2023) and Todd et al. (2024). The key idea behind task vectors is that they capture the essence of a task demonstrated in a few-shot prompt, allowing the model to apply this learned task to new inputs without explicit fine-tuning. Task vectors have several important properties:

1. They can be extracted from the model's hidden states given ICL prompts as inputs.
2. When added to the model's activations in a zero-shot setting, they can induce task performance without explicit context.
3. They appear to encode abstract task information, independent of specific input-output examples.

To illustrate the concept, consider the following simple prompt for an antonym task in the Example 1, where boxes represent distinct tokens:

BOS	Follow	the	pattern	:	\n	
hot	→	cold	\n			
big	→	small	\n			
fast	→	slow				

Example 1: All token types in an example input: prompt, input, arrow, output, newline (target tokens for calculating the loss on included)

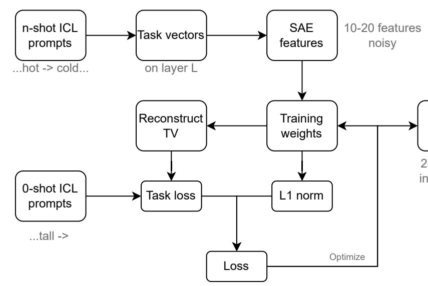


Figure 2: Overview of the task vector cleaning algorithm (see Figure 10; TV stands for task vector).

In this case, the task vector would encode the abstract notion of "finding the antonym" rather than specific word pairs. Task vectors are typically collected by averaging the residual stream of "→" tokens at a specific layer across multiple ICL prompts for a given task. This averaged representation can then be used to study the model's internal task representations and to manipulate its behavior in zero-shot settings. We perform our analysis on the datasets derived from Todd et al. (2024). Details can be found in Appendix A.

3 Discovering Task-Execution Features

3.1 Decomposing Task Vectors

To gain a deeper understanding of task vectors, we attempted to decompose them using sparse autoencoders (SAEs). However, several of our initial naive approaches faced significant challenges. Firstly, direct SAE reconstruction, i.e. passing the task vector as input to the SAE, produced noisy results with more than 10 nonzero SAE features on average on layers of interest[4], most of which were irrelevant to the task. Moreover, this reconstruction noticeably reduced the vector's performance. These issues arose partly because task vectors are out-of-distribution inputs for SAEs, as they aggregate information from different residual streams rather than representing a single one.

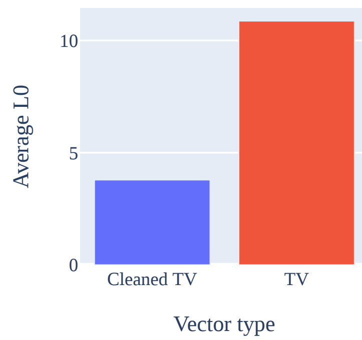
We then explored inference-time optimization (ITO) (Smith, 2024) as an alternative. However, this method also failed to reconstruct task vectors using a low number of SAE features while maintaining high performance.

Given these observations, we developed a novel method called **task vector cleaning**. It produces optimized SAE decomposition weights $\theta \in \mathbb{R}^{d_{SAE}}$ for a task vector v_{tv}. At a high level, the method:

1. Initializes θ with weights from SAE decomposition of v_{tv}.

[4]Layers where steering with task vectors decreased loss significantly (Figure 3a). We found 3-5 interpretable features. Our cleaning algorithm can usually reduce the number to 2-4. The usual residual SAE L0 is around 44. as highlighted in the Figure 3b

(a) The effect on the model's loss by steering with different kinds of reconstructed task vectors, at each layer. We see that cleaning performs similarly to the original task vector until layer 14.



(b) Average L0 for cleaned task vectors vs. original task vectors at layer 12 (which corresponds to the elbow in Figure 3a).

2. Reconstructs a new task vector v_θ from θ; steers the model with v_θ on a batch of zero-shot prompts and computes negative log-likelihood loss $\mathcal{L}_{NLL}(\theta)$ on them.

3. Optimizes θ to minimize $\mathcal{L} = \mathcal{L}_{NLL}(\theta) + l\|\theta\|_1$, where l is the L_1 regularization coefficient.

This approach allows us to maintain or even improve the performance of task vectors while reducing the amount of active SAE features to less than 4 on average (Figure 3b) for Gemma 1 2B. The algorithm overview can be found in Figure 10. Further details are in Appendix D.

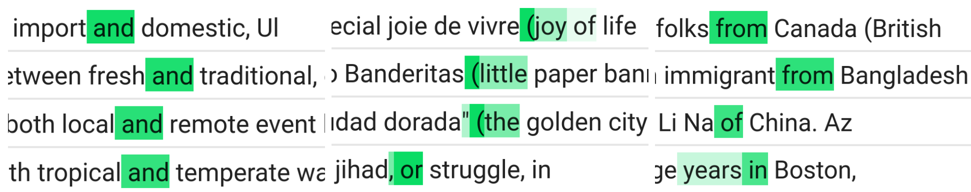
We compare it with the four baselines: original task vectors, naive SAE reconstruction, ITO with target L0 norm set to 5, and ITO with target L0 set to 20. To compare them, we steer the zero-shot prompt using the reconstructed task vector and calculate relative log-likelihood loss improvement. We then average it across all tasks. Layer-wise comparison results can be found in Figure 3a. We have also conducted sweeps for L_1 regularization coefficient l across several models and SAEs, including multiple widths and target sparsities for Gemma 2 2B and 9B. Their results are included in Appendix D.1 and show that the method can consistently reduce the amount of active SAE features by 50-80% while preserving the performance of task vectors. They also suggest that the method benefits from SAEs with higher target L0.

Using this method, we broke down task vectors into a small set of features. Many of these features were easy to interpret and clearly related to the task at hand. We found a particularly interesting group of features, which we called "task-execution features" (or "executor features"). These features have two key characteristics:

1. They activate when the model encounters examples of the relevant task in normal text.
2. In these encounters they activate on the token just before the task is completed.

For instance, imagine an antonym task feature processing the phrase " hot and cold ." It would activate on the token " and ," suggesting that the model expects an antonym to follow. This tells us that the model recognizes it's dealing with an antonym pair before seeing the complete pair. See Figure 4 for examples of such features. Appendix I contains more examples of such features with their max activating examples on SAE training data, which show that the features often have task-related max activating patterns.

To analyze the activation patterns of executor features, we split all ICL prompt tokens into several types (highlighted in Example 1 and discussed later in Section 4.1.1). For each executor feature, we calculate its token type *activation masses*: the sum of all its activations on tokens of a particular type across a batch of ICL prompts. Table 1 shows the percentages of total mass split among different token types for executor features. We can see that executors activate largely on arrow tokens.



(a) Antonyms executor feature 11618.

(b) Translation to English executor feature 5579.

(c) Prediction of city/country feature 850.

Figure 4: A subset of max activating examples for executor features from Appendix I.

3.2 STEERING EXPERIMENTS

To validate the causal relevance of our decomposed task features, we conducted a series of steering experiments. To observe the features' impact on task performance across different contexts and model layers.

The experiments were performed on the dataset of diverse tasks taken from Todd et al. (2024). We first extracted relevant task features using our cleaning algorithm. Then steered the zero-shot prompt using them and calculated relative loss improvement, normalizing and clipping it after that. Further details and additional experiments that include other models can be found in Appendix F.

Figure 5 shows a heatmap of steering results for each pair of tasks and task-relevant features. Higher values indicate greater improvement in the loss after steering. It can be seen that most tasks have a single feature with a high effect on them, and this feature generally does not significantly affect unrelated tasks. Another notable detail is that features from related tasks (like the translation group) at least partially affect all tasks within the group.

We have manually examined the features with the highest effect and found that their maximum activating dataset examples tend to align with their hypothesized role in the ICL circuit. Interestingly, we observed that translation-to-English tasks all share a generic English-to-foreign task execution feature, thus requiring an additional language encoding feature for complete task encoding. This shared feature suggests a common mechanism for translation tasks, with language-specific information encoded separately. Max activating examples of the most interpretable features are present in Appendix I.

Token Type	Mass (%)
arrow	89.80
output	6.46
input	3.2
newline	0.54
prompt	0.00

Table 1: Activation masses for **executor** features across different token types, averaged across all tasks. We can notice they activate largely on arrow tokens.

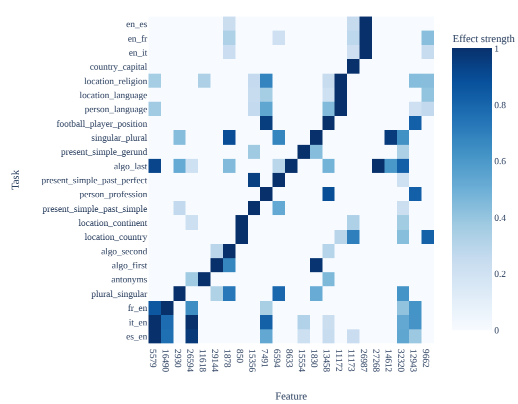


Figure 5: Heatmap showing the effect of steering with individual task-execution features for each task. Most features boost exactly one task, with a few exceptions for similar tasks like translating to English. Full and unfiltered versions of the heatmap are available in Appendix F.

4 Applying SFC to ICL

After identifying task-execution features through our task vector analysis, we sought to expand our understanding of the in-context learning (ICL) circuit. To this end, we apply the Sparse Feature Circuits (SFC) methodology (Marks et al., 2024) to the Gemma-1 2B model. However, due to the increased complexity of ICL tasks and the larger model size, the original SFC approach did not work out of the box. We had to implement several key modifications to address the challenges we encountered.

4.1 Our Modifications

4.1.1 Token Position Categorization and Feature Aggregation

We modified the SFC approach to better handle the structured nature of ICL prompts. Instead of treating each SAE feature as a separate node, we categorized token positions into the following groups:

- Prompt: The initial instruction tokens (e.g., "Follow the pattern:")
- Input: The last token before each arrow in an example pair
- Arrow: The arrow token itself ("→")
- Output: The last token before each newline in an example pair
- Newline: The newline token
- Extra: Any tokens not covered by the above categories (e.g., in multi-token inputs or outputs)

Each pair of an SAE feature and a token type was assigned its own graph node. The effects of the feature were aggregated across all tokens of the corresponding type. This categorization allowed us to evaluate how features affect all tokens within the same category, separating features based on their role in the ICL circuit. It also enabled us to selectively disable parts of the circuit for one task while testing another, verifying the task specificity of the identified circuits.

4.1.2 Loss Function Modification

An ICL prompt can be viewed as an (x, y) pair, where x represents the entire prompt except for the last pair's output, and y represents this output. The original SFC paper suggested using the log probabilities of y conditioned on x for such datasets. However, this approach often resulted in task-relevant features having high negative IEs on other example pairs in the prompt. This was likely due to the circuit's effect on those pairs being lost to either diminishing gradients in backpropagation or because copying circuits were much more relevant to predicting the last pair. By considering all pairs except the first one, we amplified the effect of the task-solving circuit relative to the numerous cloning circuits that activate due to the repetitive nature of ICL prompts.

4.1.3 SFC Evaluation

To evaluate the quality of our SFC modification, we conducted a series of ablation experiments across the same dataset of ICL tasks. Our primary metric for evaluation was faithfulness, which measures how much of the original task performance is maintained after ablating specific features. We calculated faithfulness using the following formula:

$$F(M) = \frac{M - M_a}{M_n - M_a} \tag{4}$$

Where M is the current metric (loss), M_a is the fully ablated model metric, and M_n is the non-ablated model metric.

We evaluated the impact of ablating features for one task on the performance of all other tasks. Specifically, we ablated the nodes with highest Indirect Effects (IEs) first, continuing until we reached a faithfulness of 0.5 for the target task. Faithfulness of 0.5 corresponds to half of the original performance, i.e. a significantly destructive ablation for the target task. This approach allowed us to assess both the specificity of the circuits discovered and their impact on related tasks. Our analysis

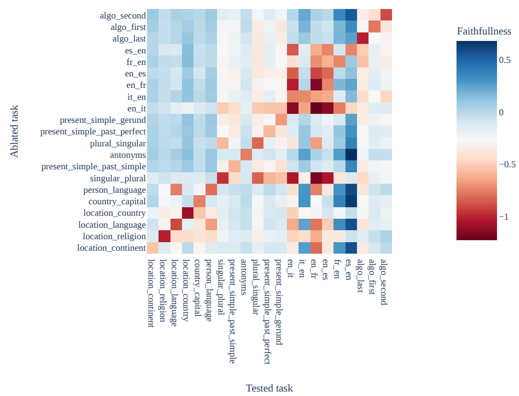


Figure 6: We study how useful the most important nodes on task A are for performance on task B. Specifically, we ablate the most important features for task A (the ablated task on the y-axis) so that faithfulness reduces by 0.5, and measure how much faithful reduces on another task B (the tested task on the x-axis).

revealed that it is possible to significantly reduce faithfulness by disabling just a few hundred nodes. Furthermore, we found that we could reduce the number of active nodes to less than a thousand while keeping the performance almost intact. Extra details and faithfulness/completeness charts can be found in Appendix E.

Figure 6 presents a heatmap showing the change in faithfulness for various tasks when ablating the highest IE nodes for a single task. Several key observations can be made from this visualization:

- **Task Specificity**: Ablating most tasks does not significantly impact the performance of others, indicating that the discovered circuits are largely task-specific. This suggests that there are no common high-IE ICL-specific nodes across tasks.
- **Related Task Effects**: Tasks are grouped into categories, and we observe that ablation of related tasks has a higher effect on all tasks within the same group. This is visible as squares along the diagonal, particularly noticeable in the translation group.
- **Performance Improvement**: For some tasks, we observe that faithfulness rises well above 1.0 after ablation of other tasks. We hypothesize that this occurs because we reduce the confusion of the model by removing irrelevant execution paths.

It is worth noting that we excluded the **person_profession** and **football_player_position** tasks from Figure 6 due to the very small difference between their fully ablated and non-ablated losses. This resulted in highly unstable faithfulness calculations for these tasks. We attribute this small difference partially to our modified loss function, as we found that calculating the loss only from the last pair results in a higher loss difference.

4.2 Task-Detection Features

Our modified SFC analysis revealed a second crucial component of the ICL mechanism: task-detection features. These features activate on instances of a complete task in the training data, specifically on the token that **completes the task**, contrary to executors that activate right before them. Both task-detection and task-execution features showed high Indirect Effects (IEs) in the extracted sparse feature circuits, with task-detection features connected to task execution features through attention output and transcoder nodes. We applied our task vector cleaning algorithm to extract task-detection features, identifying layer 11 as optimal for steering, preceding the layer 12 task-execution features. The details can be found in Appendix G. As with executor features, we present the steering heatmap in Figure 7 and the activation mass percentages in Table 2. We again see the task and token-type specificity of these features.

To evaluate the causal connection between task-detection features and task-execution features, we selected the most relevant detection and execution pairs based on steering effects and confirmed that

Token Type	Mass (%)
output	96.76
input	3.22
newline	0.01
arrow	0.0
prompt	0.0

Table 2: Activation masses for **task-detection** features across different token types, averaged across all tasks. We can notice that they activate almost exclusively on output tokens.

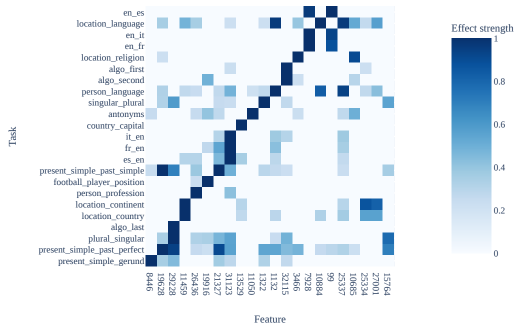


Figure 7: Heatmap showing the effect of steering with the task-detection feature most relevant to each task, on every task. We see that task detection features are typically specific to the task, with exceptions for similar tasks.

their max activating patterns aligned with their hypothesized circuit roles. We then ablated detection directions while fixing attention patterns and measured the decrease in execution activations. Figure 8 presents the results.

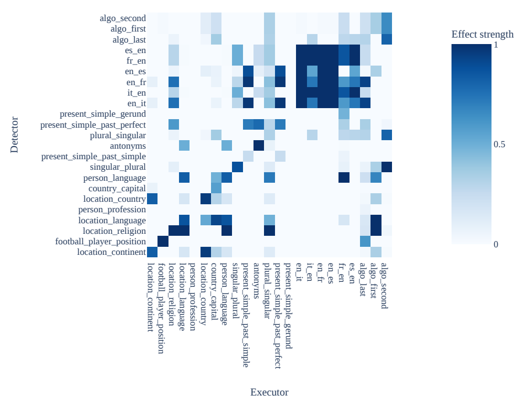


Figure 8: Heatmap showing the causal effect of the top task-detection features of each task, on the activation of the top task-execution features for every task. Averaged across all initial non-zero activations in all tasks.

The results of our causal connection analysis reveal several key insights. First, we observe strong causal connections between most task-detection and their corresponding task-execution features, supporting our hypothesis about their roles in the ICL circuit. Second, we note significant interconnectivity among translation tasks, suggesting shared circuitry for this group of related tasks. Interestingly, two tasks (**person_profession** and **present_simple_gerund**) showed unexpectedly weak connections between their detection and execution features, warranting further investigation.

5 Related work

Mechanistic Interpretability Olah et al. (2020) defines a framing for mechanistic interpretability in terms of *features* and *circuits*. It claims that neural network latent spaces have directions in them called features that correspond to meaningful variables. These features interact through model components sparsely to form circuits: interpretable computation subgraphs relevant to particular tasks. These circuits can be found through manual inspection in vision models (Cammarata et al., 2020). In language models, they can be found through manual patching (Wang et al., 2022; Hanna et al., 2023; Lieberum et al., 2023; Chan et al., 2022) or automated circuit discovery (Conmy et al.

(2023); Syed et al. (2023); Bhaskar et al. (2024), though see Miller et al. (2024)). Marks et al. (2024) extends this research area to use **Sparse Autoencoders**, as discussed below.

In-Context Learning (ICL) ICL was first introduced in Brown et al. (2020) and refers to models learning to perform tasks from prompt information at test time. There is a large area of research studying its applications (Dong et al., 2024), high-level mechanisms (Min et al., 2022) and limitations (Peng et al., 2023). Elhage et al. (2021) and Olsson et al. (2022) find *induction heads* partly responsible for in-context learning. However, since these attention heads do more than just induction (Goldowsky-Dill et al., 2023), and are not sufficient for complex task-following, induction heads alone cannot explain ICL. Anil et al. (2024, Appendix G) proposes a mechanistic hypothesis for an aspect of simple in-context task behavior. Hendel et al. (2023) and Todd et al. (2024) find that simple in-context learning tasks create strong directions in the residual stream adding which makes it possible for a network to perform tasks zero-shot, but does not explain how task vectors form nor what interpretable components the task vectors are composed of. A more detailed discussion can be found in Appendix H. Of particular interest is Wang et al., which investigates a simple ICL classification task and finds similar results with different terminology (information flow instead of circuits, ”label words” instead of task-detection features).

Sparse Autoencoders A major roadblock to mechanistic interpretability research is superposition (Elhage et al., 2022b), where the interpretable units of neural network do not tend to align with the basis directions (e.g. neurons). Sparse autoencoders (Ng, 2011; Bricken et al., 2023) are one method of addressing this roadblock, and multiple works since proposed improvements to SAE training (Rajamanoharan et al., 2024b; Bussmann et al., 2024; Braun et al., 2024; Gao et al., 2024; Templeton et al., 2024b), and we use several more in our work (Rajamanoharan et al., 2024a; Adam Jermyn, 2024; Conerly et al., 2024). Cunningham et al. (2023), building on Bills et al. (2023), apply Conmy et al. (2023) to find circuits in small language models. Marks et al. (2024) adapt Syed et al. (2023) in the SAE basis to find circuits and address a practical bias reduction problem. Kissane et al. (2024) apply a slightly different automated SAE algorithm (similar to ours in that it operates on single prompts) to IOI (Wang et al., 2022), using SAEs on the attention layer outputs and residual stream. Dunefsky et al. (2024) introduce *transcoders* (which are also briefly discussed in Templeton et al. (2024a) and Li et al. (2023)) to simplify analysis of circuits involving MLPs. We build on their work and train transcoders as part of our suite of Gemma-1 SAEs.

6 Conclusion

Limitations Our work focused on the simple task vector setting to study ICL (Section 2.3), which does not capture all ways that ICL is used in practice (generally involving far more tokens and open-ended tasks). We also only interpreted Gemma-1 2B. Therefore, other LLM architectures or model sizes could lead to different results (though this is not likely, since task vectors exist across models (Todd et al., 2024)). Finally, the complexity of the task studied meant our interpretations have some approximation error: attention heads matter for the detection-execution connection, but the succeeding MLP is necessary to capture the full effect (Section 4.2). This means that our explanation needs to include moving parts aside from task-detection attention output features. It is possible to model the effects of the MLP through transcoder features, but we leave that for future work.

Future Work Future work could extend SFC methods to work on more than a band of layers in the middle of the model (Section 2.2). Since many features correspond to individual input tokens and output predictions (due to the three stages of inference in LLMs; Elhage et al. (2022a); Lad et al. (2024)), this will require further adaptation of the SFC methodology. Moreover, our multiple contributions will hopefully spur further work that finds new tasks to interpret or explain in greater depth than prior work, as discussed in our concluding paragraph below.

To summarize our work: we use SAEs to explain in-context learning in greater detail than any prior mechanistic interpretability work. This provides strong evidence that Sparse Autoencoders are valuable circuit analysis tools, and the innovations developed: TVC (Section 3.1), SFC improvements (Section 2.2) and an SAE training codebase in JAX with open SAE weights (Section 7) are likely to help enable lots of other SAE research to tackle more ambitious tasks and larger models.

7 Reproducibility Statement

We are committed to fostering reproducibility and advancing research in the field of mechanistic interpretability. To support this goal, we plan to release the following resources upon successful acceptance of this paper:

1. Two JAX libraries optimized for TPU:
    - A library for Sparse Autoencoder (SAE) training
    - A library for SAE inference and model analysis, built upon Penzai with our custom Llama and Gemma ports
2. A full suite of SAEs for Gemma 2B, along with a dataset of their max activating examples
3. Two custom dashboards used in our analysis:
    - A dashboard for browsing max activating examples
    - An interactive dashboard for exploring extracted Sparse Feature Circuits (SFC)

These resources will enable researchers to replicate our experiments, extend our work, and conduct their own investigations using our tools and methodologies. The release of our custom dashboards will provide additional transparency and facilitate a deeper exploration of our results. Due to the complexity of our infrastructure, we only share anonymized versions of our analysis, cleaning, and SFC scripts, which still require our JAX libraries to run. We hope that reviewers will find this, along with the detailed methodologies described in the paper, sufficient evidence of reproducibility.

References

Marah Abdin, Jyoti Aneja, Hany Awadalla, Ahmed Awadallah, Ammar Ahmad Awan, Nguyen Bach, Amit Bahree, Arash Bakhtiari, Jianmin Bao, Harkirat Behl, Alon Benhaim, Misha Bilenko, Johan Bjorck, Sébastien Bubeck, Martin Cai, Qin Cai, Vishrav Chaudhary, Dong Chen, Dongdong Chen, Weizhu Chen, Yen-Chun Chen, Yi-Ling Chen, Hao Cheng, Parul Chopra, Xiyang Dai, Matthew Dixon, Ronen Eldan, Victor Fragoso, Jianfeng Gao, Mei Gao, Min Gao, Amit Garg, Allie Del Giorno, Abhishek Goswami, Suriya Gunasekar, Emman Haider, Junheng Hao, Russell J. Hewett, Wenxiang Hu, Jamie Huynh, Dan Iter, Sam Ade Jacobs, Mojan Javaheripi, Xin Jin, Nikos Karampatziakis, Piero Kauffmann, Mahoud Khademi, Dongwoo Kim, Young Jin Kim, Lev Kurilenko, James R. Lee, Yin Tat Lee, Yuanzhi Li, Yunsheng Li, Chen Liang, Lars Liden, Xihui Lin, Zeqi Lin, Ce Liu, Liyuan Liu, Mengchen Liu, Weishung Liu, Xiaodong Liu, Chong Luo, Piyush Madan, Ali Mahmoudzadeh, David Majercak, Matt Mazzola, Caio César Teodoro Mendes, Arindam Mitra, Hardik Modi, Anh Nguyen, Brandon Norick, Barun Patra, Daniel Perez-Becker, Thomas Portet, Reid Pryzant, Heyang Qin, Marko Radmilac, Liliang Ren, Gustavo de Rosa, Corby Rosset, Sambudha Roy, Olatunji Ruwase, Olli Saarikivi, Amin Saied, Adil Salim, Michael Santacroce, Shital Shah, Ning Shang, Hiteshi Sharma, Yelong Shen, Swadheen Shukla, Xia Song, Masahiro Tanaka, Andrea Tupini, Praneetha Vaddamanu, Chunyu Wang, Guanhua Wang, Lijuan Wang, Shuohang Wang, Xin Wang, Yu Wang, Rachel Ward, Wen Wen, Philipp Witte, Haiping Wu, Xiaoxia Wu, Michael Wyatt, Bin Xiao, Can Xu, Jiahang Xu, Weijian Xu, Jilong Xue, Sonali Yadav, Fan Yang, Jianwei Yang, Yifan Yang, Ziyi Yang, Donghan Yu, Lu Yuan, Chenruidong Zhang, Cyril Zhang, Jianwen Zhang, Li Lyna Zhang, Yi Zhang, Yue Zhang, Yunan Zhang, and Xiren Zhou. Phi-3 technical report: A highly capable language model locally on your phone, 2024. URL https://arxiv.org/abs/2404.14219.

Adly Templeton Adam Jermyn. Ghost grads: An improvement on resampling, 2024. URL https://transformer-circuits.pub/2024/jan-update/index.html#dict-learning-resampling.

Cem Anil, Esin Durmus, Mrinank Sharma, Joe Benton, Sandipan Kundu, Joshua Batson, Nina Rimsky, Meg Tong, Jesse Mu, Daniel Ford, et al. Many-shot jailbreaking. 2024.

Yu Bai, Fan Chen, Huan Wang, Caiming Xiong, and Song Mei. Transformers as statisticians: Provable in-context learning with in-context algorithm selection.

Hritik Bansal, Karthik Gopalakrishnan, Saket Dingliwal, Sravan Bodapati, Katrin Kirchhoff, and Dan Roth. Rethinking the role of scale for in-context learning: An interpretability-based case study at 66 billion scale. URL http://arxiv.org/abs/2212.09095.

Adithya Bhaskar, Alexander Wettig, Dan Friedman, and Danqi Chen. Finding transformer circuits with edge pruning, 2024. URL https://arxiv.org/abs/2406.16778.

Steven Bills, Nick Cammarata, Dan Mossing, et al. Language models can explain neurons in language models. https://openaipublic.blob.core.windows.net/neuron-explainer/paper/index.html, 2023.

Joseph Bloom. Open source sparse autoencoders for all residual stream layers of gpt2-small, 2024. URL https://www.alignmentforum.org/posts/f9EgfLSurAiqRJySD/open-source-sparse-autoencoders-for-all-residual-stream.

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/jax-ml/jax.

Dan Braun, Jordan Taylor, Nicholas Goldowsky-Dill, and Lee Sharkey. Identifying functionally important features with end-to-end sparse dictionary learning, 2024. URL https://arxiv.org/abs/2405.12241.

Trenton Bricken, Adly Templeton, Joshua Batson, Brian Chen, Adam Jermyn, Tom Conerly, Nick Turner, Cem Anil, Carson Denison, Amanda Askell, Robert Lasenby, Yifan Wu, Shauna Kravec, Nicholas Schiefer, Tim Maxwell, Nicholas Joseph, Zac Hatfield-Dodds, Alex Tamkin, Karina Nguyen, Brayden McLean, Josiah E Burke, Tristan Hume, Shan Carter, Tom Henighan, and Christopher Olah. Towards monosemanticity: Decomposing language models with dictionary learning. *Transformer Circuits Thread*, 2023. https://transformer-circuits.pub/2023/monosemantic-features/index.html.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D. Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pp. 1877–1901, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html.

Bart Bussmann, Patrick Leask, and Neel Nanda. Batchtopk: A simple improvement for topk-saes, 2024. URL https://www.alignmentforum.org/posts/Nkx6yWZNbAsfvic98/batchtopk-a-simple-improvement-for-topk-saes.

Nick Cammarata, Shan Carter, Gabriel Goh, Chris Olah, et al. Thread: Circuits. *Distill*, 2020. doi: 10.23915/distill.00024. https://distill.pub/2020/circuits.

Lawrence Chan, Adria Garriga-Alonso, Nix Goldowsky-Dill, Ryan Greenblatt, Jenny Nitishinskaya, Ansh Radhakrishnan, Buck Shlegeris, and Nate Thomas. Causal scrubbing: A method for rigorously testing interpretability hypotheses, 2022. URL https://www.alignmentforum.org/posts/JvZhhzycHu2Yd57RN/causal-scrubbing-a-method-for-rigorously-testing.

Siyu Chen, Heejune Sheen, Tianhao Wang, and Zhuoran Yang. Unveiling induction heads: Provable training dynamics and feature learning in transformers. URL http://arxiv.org/abs/2409.10559.

Tom Conerly, Adly Templeton, Trenton Bricken, Jonathan Marcus, and Tom Henighan. Update on how we train saes, 2024. URL https://transformer-circuits.pub/2024/april-update/index.html#training-saes.

Arthur Conmy, Augustine N. Mavor-Parker, Aengus Lynch, et al. Towards automated circuit discovery for mechanistic interpretability. In *Proceedings of NeurIPS*, 2023.

Hoagy Cunningham, Aidan Ewart, Logan Riggs, et al. Sparse autoencoders find highly interpretable features in language models, 2023. URL https://arxiv.org/abs/2309.08600.

Damai Dai, Yutao Sun, Li Dong, Yaru Hao, Shuming Ma, Zhifang Sui, and Furu Wei. Why can GPT learn in-context? language models implicitly perform gradient descent as meta-optimizers. URL http://arxiv.org/abs/2212.10559.

Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Jingyuan Ma, Rui Li, Heming Xia, Jingjing Xu, Zhiyong Wu, Tianyu Liu, Baobao Chang, Xu Sun, Lei Li, and Zhifang Sui. A survey on in-context learning, 2024. URL https://arxiv.org/abs/2301.00234.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, et al. The llama 3 herd of models, 2024. URL https://arxiv.org/abs/2407.21783.

Jacob Dunefsky, Philippe Chlenski, and Neel Nanda. Transcoders find interpretable llm feature circuits, 2024. URL https://arxiv.org/abs/2406.11944.

Michael Elad. *Sparse and Redundant Representations: From Theory to Applications in Signal and Image Processing*. Springer, New York, 2010. ISBN 978-1-4419-7010-7. doi: 10.1007/978-1-4419-7011-4.

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. *Transformer Circuits Thread*, 2021. URL https://transformer-circuits.pub/2021/framework/index.html.

Nelson Elhage, Tristan Hume, Catherine Olsson, Neel Nanda, Tom Henighan, Scott Johnston, Sheer ElShowk, Nicholas Joseph, Nova DasSarma, Ben Mann, Danny Hernandez, Amanda Askell, Kamal Ndousse, 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. Softmax linear units. *Transformer Circuits Thread*, 2022a. 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, et al. Toy Models of Superposition. *arXiv preprint arXiv:2209.10652*, 2022b.

Eoin Farrell. Experiments with an alternative method to promote sparsity in sparse autoencoders, 2024. URL https://www.lesswrong.com/posts/cYA3ePxy8JQ8ajo8B/experiments-with-an-alternative-method-to-promote-sparsity.

Leo Gao, Tom Dupré la Tour, Henk Tillman, Gabriel Goh, Rajan Troll, Alec Radford, Ilya Sutskever, Jan Leike, and Jeffrey Wu. Scaling and evaluating sparse autoencoders, 2024. URL https://arxiv.org/abs/2406.04093.

Shivam Garg, Dimitris Tsipras, Percy Liang, and Gregory Valiant. What can transformers learn in-context? a case study of simple function classes.

Gemma Team. Gemma: Open models based on gemini research and technology, 2024. URL https://arxiv.org/abs/2403.08295.

Nicholas Goldowsky-Dill, Chris MacLeod, Lucas Sato, and Aryaman Arora. Localizing model behavior with path patching, 2023. URL https://arxiv.org/abs/2304.05969.

Roger Grosse, Juhan Bae, Cem Anil, Nelson Elhage, Alex Tamkin, Amirhossein Tajdini, Benoit Steiner, Dustin Li, Esin Durmus, Ethan Perez, Evan Hubinger, Kamilė Lukošiūtė, Karina Nguyen, Nicholas Joseph, Sam McCandlish, Jared Kaplan, and Samuel R. Bowman. Studying large language model generalization with influence functions, 2023. URL https://arxiv.org/abs/2308.03296.

Xiaochuang Han, Daniel Simig, Todor Mihaylov, Yulia Tsvetkov, Asli Celikyilmaz, and Tianlu Wang. Understanding in-context learning via supportive pretraining data. URL http://arxiv.org/abs/2306.15091.

Michael Hanna, Ollie Liu, and Alexandre Variengien. How does gpt-2 compute greater-than?: Interpreting mathematical abilities in a pre-trained language model, 2023. URL https://arxiv.org/abs/2305.00586.

Roee Hendel, Mor Geva, and Amir Globerson. In-context learning creates task vectors, 2023. URL https://arxiv.org/abs/2310.15916.

Daniel D. Johnson. Penzai + treescope: A toolkit for interpreting, visualizing, and editing models as data, 2024. URL https://arxiv.org/abs/2408.00211.

Patrick Kidger and Cristian Garcia. Equinox: neural networks in jax via callable pytrees and filtered transformations, 2021. URL https://arxiv.org/abs/2111.00254.

Connor Kissane, Robert Krzyzanowski, Arthur Conmy, and Neel Nanda. Attention output saes improve circuit analysis, 2024. URL https://www.alignmentforum.org/posts/EGvtgB7ctifzxZg6v/attention-output-saes-improve-circuit-analysis.

Vedang Lad, Wes Gurnee, and Max Tegmark. The remarkable robustness of llms: Stages of inference?, 2024. URL https://arxiv.org/abs/2406.19384.

Max Li, Sam Marks, and Aaron Mueller. dictionary_learning repository, 2023. URL https://github.com/saprmarks/dictionary_learning?tab=readme-ov-file#extra-functionalitysupported-by-this-repo. Accessed on September 30, 2024.

Tom Lieberum, Matthew Rahtz, János Kramár, et al. Does circuit analysis interpretability scale? evidence from multiple choice capabilities in chinchilla, 2023. URL https://arxiv.org/abs/2307.09458.

Tom Lieberum, Senthooran Rajamanoharan, Arthur Conmy, Lewis Smith, Nicolas Sonnerat, Vikrant Varma, János Kramár, Anca Dragan, Rohin Shah, and Neel Nanda. Gemma scope: Open sparse autoencoders everywhere all at once on gemma 2, 2024. URL https://arxiv.org/abs/2408.05147.

Johnny Lin. Neuronpedia: Interactive reference and tooling for analyzing neural networks, 2023. URL https://www.neuronpedia.org. Software available from neuronpedia.org.

Arvind Mahankali, Tatsunori B. Hashimoto, and Tengyu Ma. One step of gradient descent is provably the optimal in-context learner with one layer of linear self-attention. URL http://arxiv.org/abs/2307.03576.

Samuel Marks, Can Rager, Eric J. Michaud, et al. Sparse feature circuits: Discovering and editing interpretable causal graphs in language models. *Computing Research Repository*, arXiv:2403.19647, 2024. URL https://arxiv.org/abs/2403.19647.

Joseph Miller, Bilal Chughtai, and William Saunders. Transformer circuit faithfulness metrics are not robust, 2024. URL https://arxiv.org/abs/2407.08734.

Sewon Min, Xinxi Lyu, Ari Holtzman, Mikel Artetxe, Mike Lewis, Hannaneh Hajishirzi, and Luke Zettlemoyer. Rethinking the role of demonstrations: What makes in-context learning work? *arXiv preprint arXiv:2202.12837*, 2022. URL https://arxiv.org/abs/2202.12837.

Andrew Ng. Sparse autoencoder. *CS294A Lecture Notes*, 2011. Unpublished lecture notes.

Chris Olah, Nick Cammarata, Ludwig Schubert, Gabriel Goh, Michael Petrov, and Shan Carter. Zoom in: An introduction to circuits. *Distill*, 2020. doi: 10.23915/distill.00024.001. https://distill.pub/2020/circuits/zoom-in.

Catherine Olsson, Nelson Elhage, Neel Nanda, Nicholas Joseph, Nova DasSarma, Tom Henighan, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, Dawn Drain, Deep Ganguli, Zac Hatfield-Dodds, Danny Hernandez, Scott Johnston, Andy Jones, Jackson Kernion, Liane Lovitt, Kamal Ndousse, Dario Amodei, Tom Brown, Jack Clark, Jared Kaplan, Sam McCandlish, and Chris Olah. In-context learning and induction heads, 2022. URL https://arxiv.org/abs/2209.11895.

Johannes von Oswald, Eyvind Niklasson, Ettore Randazzo, João Sacramento, Alexander Mordvintsev, Andrey Zhmoginov, and Max Vladymyrov. Transformers learn in-context by gradient descent. URL http://arxiv.org/abs/2212.07677.

Jane Pan, Tianyu Gao, Howard Chen, and Danqi Chen. What in-context learning "learns" in-context: Disentangling task recognition and task learning. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (eds.), *Findings of the Association for Computational Linguistics: ACL 2023*, pp. 8298–8319. Association for Computational Linguistics. doi: 10.18653/v1/2023.findings-acl.527. URL https://aclanthology.org/2023.findings-acl.527.

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, 2001. Morgan Kaufmann Publishers Inc. ISBN 1558608001.

Judea Pearl. *Causality: Models, Reasoning and Inference*. Cambridge University Press, USA, 2nd edition, 2009. ISBN 052189560X.

Guilherme Penedo, Hynek Kydlíček, Loubna Ben allal, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro Von Werra, and Thomas Wolf. The fineweb datasets: Decanting the web for the finest text data at scale, 2024. URL https://arxiv.org/abs/2406.17557.

Hao Peng, Xiaozhi Wang, Jianhui Chen, Weikai Li, Yunjia Qi, Zimu Wang, Zhili Wu, Kaisheng Zeng, Bin Xu, Lei Hou, and Juanzi Li. When does in-context learning fall short and why? a study on specification-heavy tasks, 2023. URL https://arxiv.org/abs/2311.08993.

Senthooran Rajamanoharan. Improving ghost grads, 2024. URL https://www.alignmentforum.org/posts/C5KAZQib3bzzpeyrg/progress-update-1-from-the-gdm-mech-interp-team-full-update#Improving_ghost_grads.

Senthooran Rajamanoharan, Arthur Conmy, Lewis Smith, Tom Lieberum, Vikrant Varma, János Kramár, Rohin Shah, and Neel Nanda. Improving dictionary learning with gated sparse autoencoders, 2024a. URL https://arxiv.org/abs/2404.16014.

Senthooran Rajamanoharan, Tom Lieberum, Nicolas Sonnerat, Arthur Conmy, Vikrant Varma, János Kramár, and Neel Nanda. Jumping ahead: Improving reconstruction fidelity with jumprelu sparse autoencoders, 2024b. URL https://arxiv.org/abs/2407.14435.

Allan Raventós, Mansheej Paul, Feng Chen, and Surya Ganguli. Pretraining task diversity and the emergence of non-bayesian in-context learning for regression. URL http://arxiv.org/abs/2306.15063.

Logan Riggs and Jannik Brinkman. Improving sae's by sqrt()-ing l1 removing lowest activating features, 2024. URL https://www.lesswrong.com/posts/YiGs8qJ8aNBgwt2YN/improving-sae-s-by-sqrt-ing-l1-and-removing-lowest.

Lingfeng Shen, Aayush Mishra, and Daniel Khashabi. Do pretrained transformers learn in-context by gradient descent? URL http://arxiv.org/abs/2310.08540.

Chenglei Si, Dan Friedman, Nitish Joshi, Shi Feng, Danqi Chen, and He He. Measuring inductive biases of in-context learning with underspecified demonstrations. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 11289–11310. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-long.632. URL https://aclanthology.org/2023.acl-long.632.

Lewis Smith. Replacing sae encoders with inference-time optimisation, 2024. URL https://www.alignmentforum.org/posts/C5KAZQib3bzzpeyrg/full-post-progress-update-1-from-the-gdm-mech-interp-team#Replacing_SAE_Encoders_with_Inference_Time_Optimisation.

Aaquib Syed, Can Rager, and Arthur Conmy. Attribution patching outperforms automated circuit discovery, 2023. URL https://arxiv.org/abs/2310.10348.

Adly Templeton, Joshua Batson, Adam Jermyn, and Chris Olah. Predicting future activations, January 2024a. URL https://transformer-circuits.pub/2024/jan-update/index.html#predict-future. Accessed on September 30, 2024.

Adly Templeton, Tom Conerly, Jonathan Marcus, Jack Lindsey, Trenton Bricken, Brian Chen, Adam Pearce, Craig Citro, Emmanuel Ameisen, Andy Jones, Hoagy Cunningham, Nicholas L Turner, Callum McDougall, Monte MacDiarmid, C. Daniel Freeman, Theodore R. Sumers, Edward Rees, Joshua Batson, Adam Jermyn, Shan Carter, Chris Olah, and Tom Henighan. Scaling monosemanticity: Extracting interpretable features from claude 3 sonnet. *Transformer Circuits Thread*, 2024b. URL https://transformer-circuits.pub/2024/scaling-monosemanticity/index.html.

Eric Todd, Millicent L. Li, Arnab Sen Sharma, et al. Function vectors in large language models. In *Proceedings of the 2024 International Conference on Learning Representations*, 2024.

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, 2022. URL https://arxiv.org/abs/2211.00593.

Lean Wang, Lei Li, Damai Dai, Deli Chen, Hao Zhou, Fandong Meng, Jie Zhou, and Xu Sun. Label words are anchors: An information flow perspective for understanding in-context learning. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 9840–9855. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.609. URL https://aclanthology.org/2023.emnlp-main.609.

Xinyi Wang, Wanrong Zhu, Michael Saxon, Mark Steyvers, and William Yang Wang. Large language models are latent variable models: Explaining and finding good demonstrations for in-context learning, 2024. URL https://arxiv.org/abs/2301.11916.

Sang Michael Xie, Aditi Raghunathan, Percy Liang, and Tengyu Ma. An explanation of in-context learning as implicit bayesian inference. URL http://arxiv.org/abs/2111.02080.

Zhangchen Xu, Fengqing Jiang, Luyao Niu, Yuntian Deng, Radha Poovendran, Yejin Choi, and Bill Yuchen Lin. Magpie: Alignment data synthesis from scratch by prompting aligned llms with nothing, 2024. URL https://arxiv.org/abs/2406.08464.

Steve Yadlowsky, Lyric Doshi, and Nilesh Tripuraneni. Pretraining data mixtures enable narrow model selection capabilities in transformer models. URL http://arxiv.org/abs/2311.00871.

A Model and dataset details

For our experiments, we utilized the Gemma 1 2B model, a member of the Gemma family of open models based on Google's Gemini models (Gemma Team, 2024). The model's architecture is largely the same as that of Llama (Dubey et al., 2024) except for tied input and output embeddings and a different activation function for MLP layers, so we could reuse our infrastructure for loading Llama models. We train residual and attention output SAEs as well as transcoders for layers 1-18 of the model on FineWeb (Penedo et al., 2024).

Our dataset for circuit finding is primarily derived from the function vectors paper (Todd et al., 2024), which provides a diverse set of tasks for evaluating the existence and properties of function vectors in language models. We supplemented this dataset with three additional algorithmic tasks to broaden the scope of our analysis:

- Extract the first element from an array of length 4
- Extract the second element from an array of length 4
- Extract the last element from an array of length 4

The complete list of tasks used in our experiments with task descriptions is as follows:

Task ID	Description
location_continent	Name the continent where the given landmark is located.
football_player_position	Identify the position of a given football player.
location_religion	Name the predominant religion in a given location.
location_language	State the primary language spoken in a given location.
person_profession	Identify the profession of a given person.
location_country	Name the country where a given location is situated.
country_capital	Provide the capital city of a given country.
person_language	Identify the primary language spoken by a given person.
singular_plural	Convert a singular noun to its plural form.
present_simple_past_simple	Change a verb from present simple to past simple tense.
antonyms	Provide the antonym of a given word.
plural_singular	Convert a plural noun to its singular form.
present_simple_past_perfect	Change a verb from present simple to past perfect tense.
present_simple_gerund	Convert a verb from present simple to gerund form.
en_it	Translate a word from English to Italian.
it_en	Translate a word from Italian to English.
en_fr	Translate a word from English to French.
en_es	Translate a word from English to Spanish.
fr_en	Translate a word from French to English.
es_en	Translate a word from Spanish to English.
algo_last	Extract the last element from an array of length 4.
algo_first	Extract the first element from an array of length 4.
algo_second	Extract the second element from an array of length 4.

This diverse set of tasks covers a wide range of linguistic and cognitive abilities, including geographic knowledge, language translation, grammatical transformations, and simple algorithmic operations. By using this comprehensive task set, we aimed to thoroughly investigate the in-context learning capabilities of the Gemma 1 2B model across various domains.

B SAE Training

Our Gemma 1 2B SAEs are trained with a learning rate of 1e-3 and Adam betas of 0.0 and 0.99 for 150M ($\pm$100) tokens of FineWeb (Penedo et al., 2024). The methodology is overall similar to (Bloom, 2024). We initialize encoder weights orthogonally and set decoder weights to their transpose. We initialize decoder biases to 0. We use Rajamanoharan (2024)'s ghost gradients variant (ghost gradients applied to dead features only, loss multiplied by the proportion of death features) with the additional modification of using softplus instead of exp for numerical stability. A feature is considered dead when its density (according to a 1000-batch buffer) is below 5e-6 or when it has not fired in 2000 steps. We use Anthropic's input normalization and sparsity loss for Gemma 1 2B (Conerly et al., 2024). We found it to improve Gated SAE training stability. We modified it to work with transcoders by keeping track of input and output norms separately and predicting normed outputs.

We convert our Gated SAEs into JumpReLU SAEs after training, implementing algorithms like TVC and SFC in a unified manner for all SAEs in this format (including simple SAEs). The conversion procedure involves setting thresholds to replicate the effect of the gating branch. For further details, see Rajamanoharan et al. (2024b).

We use 4 v4 TPU chips running Jax (Bradbury et al., 2018) (Equinox (Kidger & Garcia, 2021)) to train our SAEs. We found that training with Huggingface's Flax LM implementations was very slow. We reimplemented LLaMA (Dubey et al., 2024) and Gemma (Gemma Team, 2024) in Penzai (Johnson, 2024) with a custom layer-scan transformation and quantized inference kernels as well as support for loading from GGUF compressed model files. We process an average of around 4400 tokens per second, which makes training SAEs and not caching LM activations the main bottleneck. For this and other reasons, we don't do SAE sparsity coefficient sweeps to increase TPU utilization.

For caching, we use a distributed ring buffer which contains separate pointers on each device to allow for processing masked data. The (in-place) buffer update is in a separate JIT context. Batches are sampled randomly from the buffer for each training step.

We train our SAEs in bfloat16 precision. We found that keeping weights and scales in bfloat16 and biases in float32 performed best in terms of the number of dead features and led to a Pareto improvement over float32 SAEs.

For training Phi 3 (Abdin et al., 2024) SAEs, we use data generated by the model unconditionally, similarly to (Xu et al., 2024)[5]. The resulting dataset we train the model on contains many math problems and is formatted as a natural-seeming interaction between the user and the model.

Each SAE training run takes us about 3 hours. We trained 3 models (a residual SAE, an attention output SAE, and a transcoder) for each of the 18 layers of the model. This is about 1 week of v4-8 TPU time.

Our SAEs and training code will be made public after paper acceptance.

C EXAMPLE CIRCUITS

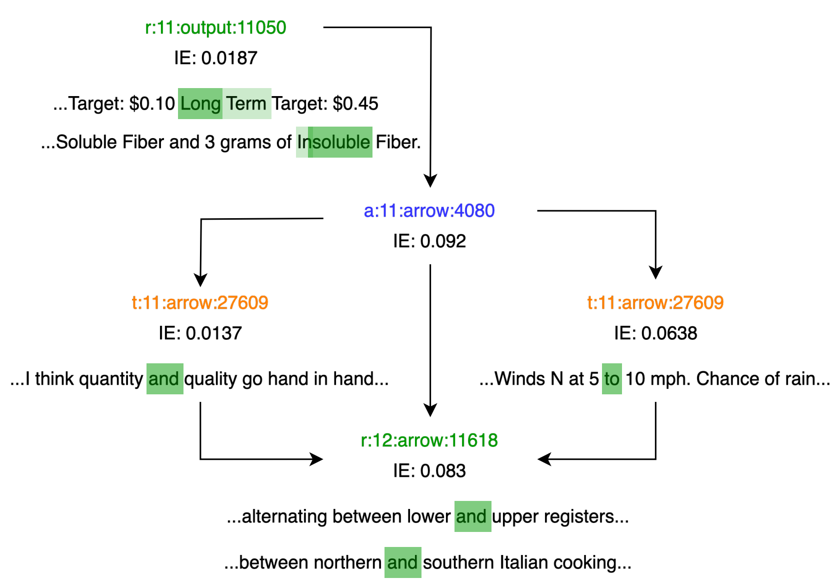


Figure 9: An example of a circuit found using our SFC variant. We focused on a subcircuit with high indirect effects. Maximum activating examples from the SAE training distribution are included.

An example output of our circuit cleaning algorithm can be found in Figure 9. We can see the flow of information through a single high-IE attention feature from a task-detection feature (activating on output tokens) to transcoder and residual execution features (activating on arrow tokens). The feature activates on antonyms on the detection feature #11050: one can assume the first sequence began as "Short Term Target", making the second half an antonym.

We will release a web interface for viewing maximum activating examples and task feature circuits.

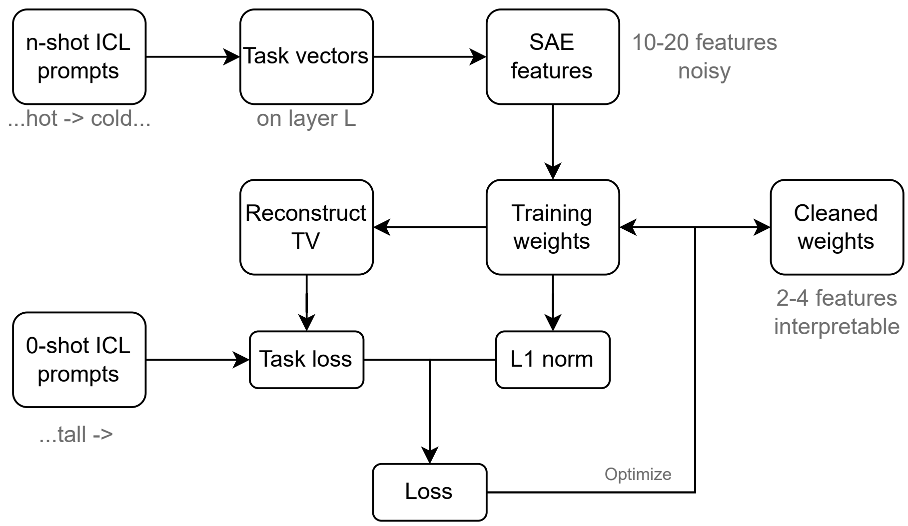


Figure 10: An overview of our Task Vector Cleaning algorithm. TV stands for Task Vector.

D Task Vector Cleaning Algorithm

The task vector cleaning algorithm is a novel approach we developed to isolate task-relevant features from task vectors. Figure 10 provides an overview of this algorithm.

Our process begins with collecting residuals for task vectors using a batch of 16 and 16-shot prompts. We then calculate the SAE features for these task vectors. We explored two methods: (1) calculating feature activation and then averaging across tokens, and (2) averaging across tokens first and then calculating the task vector. They had similar performances.

The cleaning process is performed on a training batch of 24 pairs, with evaluation conducted on an additional 24 pairs. All prompts are zero-shot. An example prompt is as follows:

BOS Follow the pattern : \n
tall → short \n
...
old → young \n
hot → cold

Example 2: The steered token is highlighted in red. Loss is calculated on the yellow token.

The algorithm is initialized with the SAE reconstruction as a starting point. It then iteratively steers the model on the reconstruction layer and calculates the loss on the training pairs. To promote sparsity, we add the L_1 norm of weights with coefficient l to the loss function. The algorithm implements early stopping when the L_0 norm remains unchanged for n iterations.

```
def tvc_algorithm(task_vector, model, sae):
  initial_weights = sae.encode(task_vector)
  def tvc_loss(weights, tokens):
    task_vector = sae.decode(weights)
    mask = tokens == self.separator
```

[5] Phi-3 is trained primarily with instruction following data, making it an aligned chat model.

```
    model.residual_stream[layer, mask] += task_vector
    # loss only on the "output" tokens,
    # ignoring input and prompt tokens
    loss = logprobs(model.logits, tokens, ...)
    return loss + l1_coeff * l1_norm(weights)
  weights = initial_weights.copy()
  optimizer = adam(weights, lr=0.15)
  last_l0, without_change = 0, 0  # early stopping
  for _ in range(1000):
    grad = jax.grad(tvc_loss)(weights, tokens)
    weights = optimizer.step(grad)
    if l0_norm(weights) != last_l0:
      last_l0, without_change = l0_norm(weights), 0
    elif without_change >= 50:
      break
  return weights
```

Algorithm 1: Pseudocode for Task Vector Cleaning.

The hyperparameters l, n, and learning rate α can be fixed for a single model. We experimented with larger batch sizes but found that they did not significantly improve the quality of extracted features while substantially slowing down the algorithm due to gradient accumulation.

The algorithm takes varying amounts of time to complete for different tasks and models. For Gemma 1, it stops at 100-200 iterations, which is close to 40 seconds at 5 iterations per second.

It's worth noting that we successfully applied this method to the recently released Gemma 2 2B and 9B models using the Gemma Scope SAE suite (Lieberum et al., 2024). It was also successful with the Phi-3 3B model (Abdin et al., 2024) and with our SAEs, which were trained similarly to the Gemma 1 2B SAEs.

D.1 L_1 SWEEPS

To provide more details about the method's effectiveness across various models and SAE widths, we conducted L_1 coefficient sweeps with our Phi-3 and Gemma 1 2B SAEs, as well as Gemma Scope Gemma 2 SAEs. We chose two SAE widths for Gemma 2 2B and 9B: 16k and 65k. For Gemma 2 2B we also sweeped across several different target SAE l0 norms. We studied only the optimal task vector layer for each model: 12 for Gemma 1, 16 for Gemma 2, 18 for Phi-3, and 20 for Gemma 2 9B. We used a learning rate of 0.15 with the Gemma 1 2B, Phi-3, and Gemma 2 2B 65k models, 0.3 with Gemma 2 2B 16k, and 0.05 with 200 early stopping steps for Gemma 2 9B.

Figures 11, 12, 13 compare TVC and ITO against original task vectors. The X-axis displays the fraction of active task vector SAE features used. The Y-axis displays the TV loss delta, calculated as $(L_{TV} - L_{Method})/L_{Zero}$, where L_{TV} is the loss from steering with the task vector, L_{Method} is the loss after it has been cleaned using the corresponding method, and L_{Zero} is the uninformed (no-steering) model loss. This metric shows improvement over the task vector relative to the loss of the uninformed model. Points were collected from all tasks using 5 different L_1 coefficient values.

We observe that our method often improves task vector loss and can reduce the number of active features to one-third of those in the original task vector while maintaining relatively intact performance. In contrast, ITO rarely improves the task vector loss and is almost always outperformed by TVC.

Figures 14, 15 and 16 show task-mean loss decrease (relative to no steering loss) and remaining TV features fraction plotted against L_1 sweep coefficients. We see that L_1 coefficients between 0.001 and 0.025 result in relatively intact performance, while significantly reducing the amount of active SAE features. From Figure 15 we can notice that the method performs better with higher target l0 SAEs, being able to affect the loss with just a fraction of active SAE features.

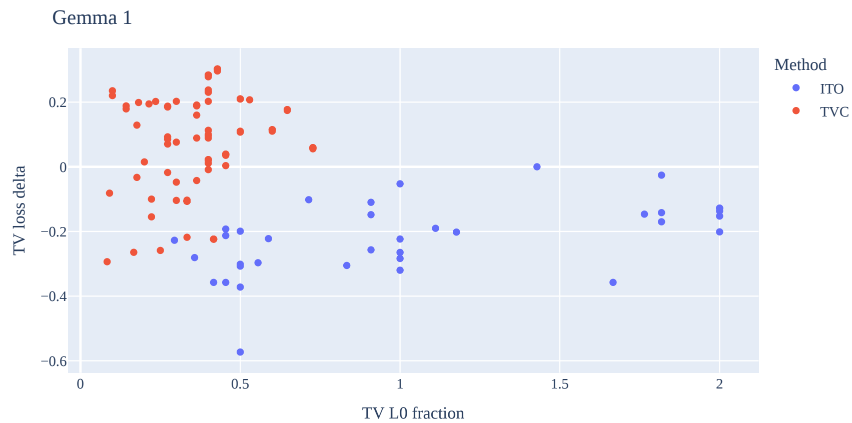


Figure 11: Performance of ITO and TVC across different tasks and optimization parameters compared to task vectors for Gemma 1 2B. The Y-axis shows relative improvement over task vector loss, while the X-axis shows the fraction of active TV features used. Metric calculation details are available in D.1

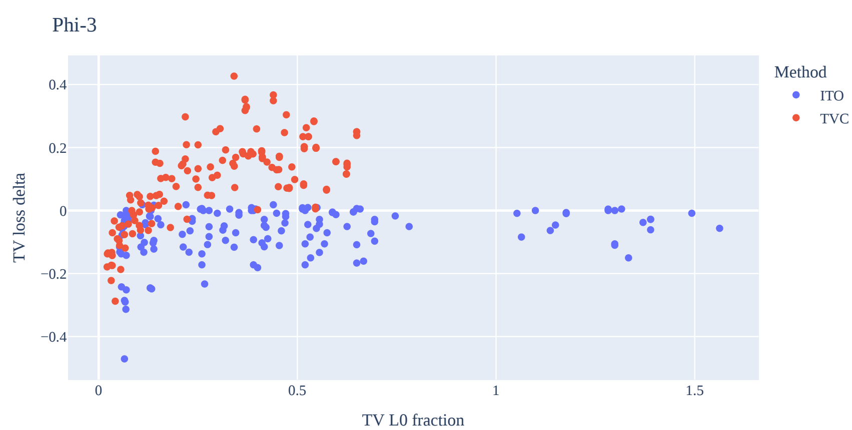


Figure 12: Performance of ITO and TVC across different tasks and optimization parameters compared to task vectors for Phi-3. The Y-axis shows relative improvement over task vector loss, while the X-axis shows the fraction of active TV features used. Metric calculation details are available in D.1

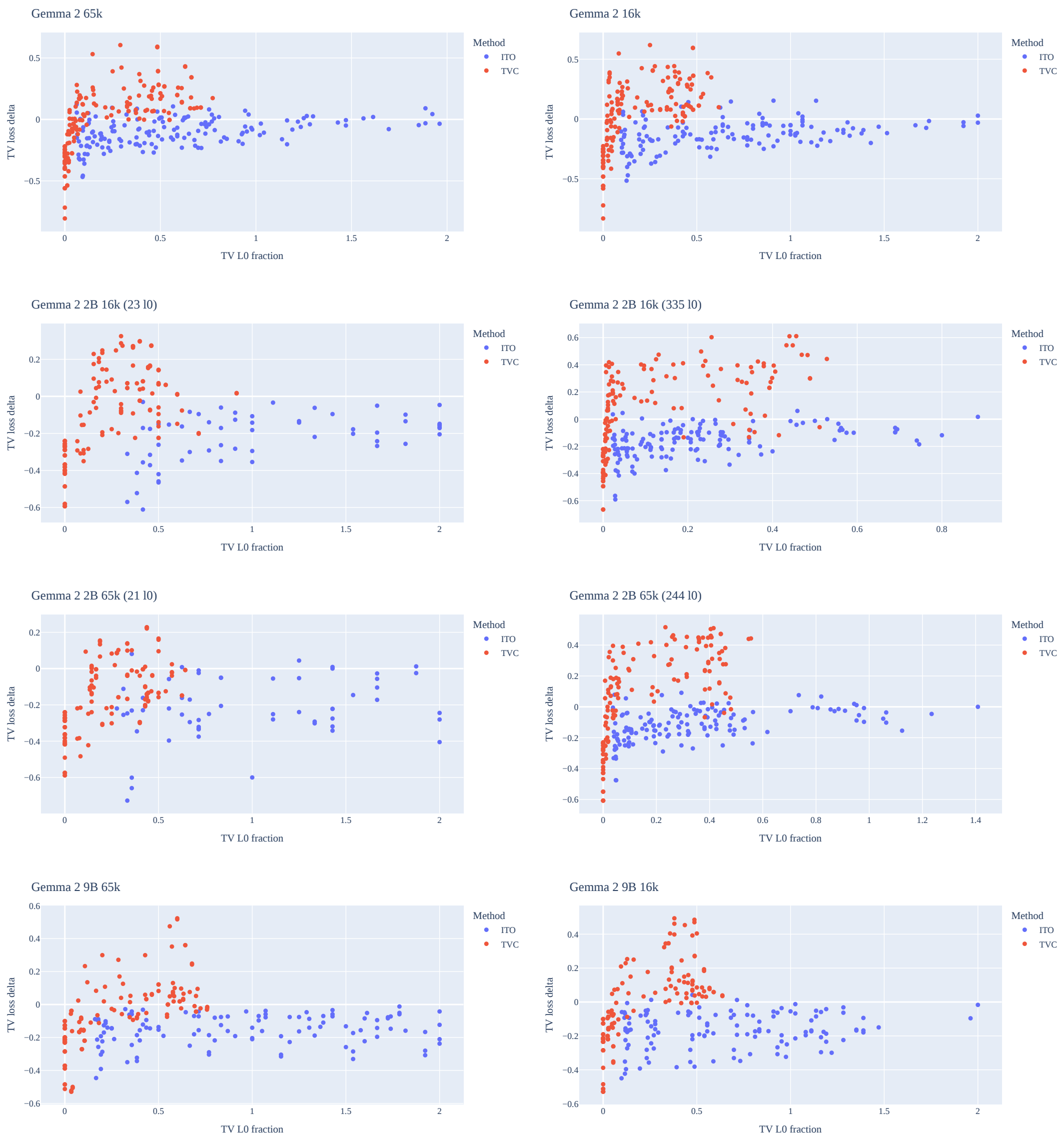


Figure 13: Performance of ITO and TVC across different tasks and optimization parameters compared to task vectors for Gemma 2 Gemma Scope SAEs. The Y-axis shows the relative improvement over the loss from steering with a task vector, while the X-axis shows the fraction of active TV features used. Metric calculation details are available in Appendix D.1.

E Details of our SFC implementation

E.1 Implementation details

Our implementation of circuit finding attribution patching is specialized for Jax and Penzai.

We first perform a forward-backward pass on the set of prompts, collecting residuals and gradients from the metric to residuals. We collect gradients with `jax.grad` by introducing "dummy" zero-valued inputs to the metric computation function that are added to the residuals of each layer. Note that we do not use SAEs during this stage.

We then perform an SAE encoding step and find the nodes (residual, attention output, and transcoder SAE features and error nodes) with the highest indirect effects using manually computed gradients

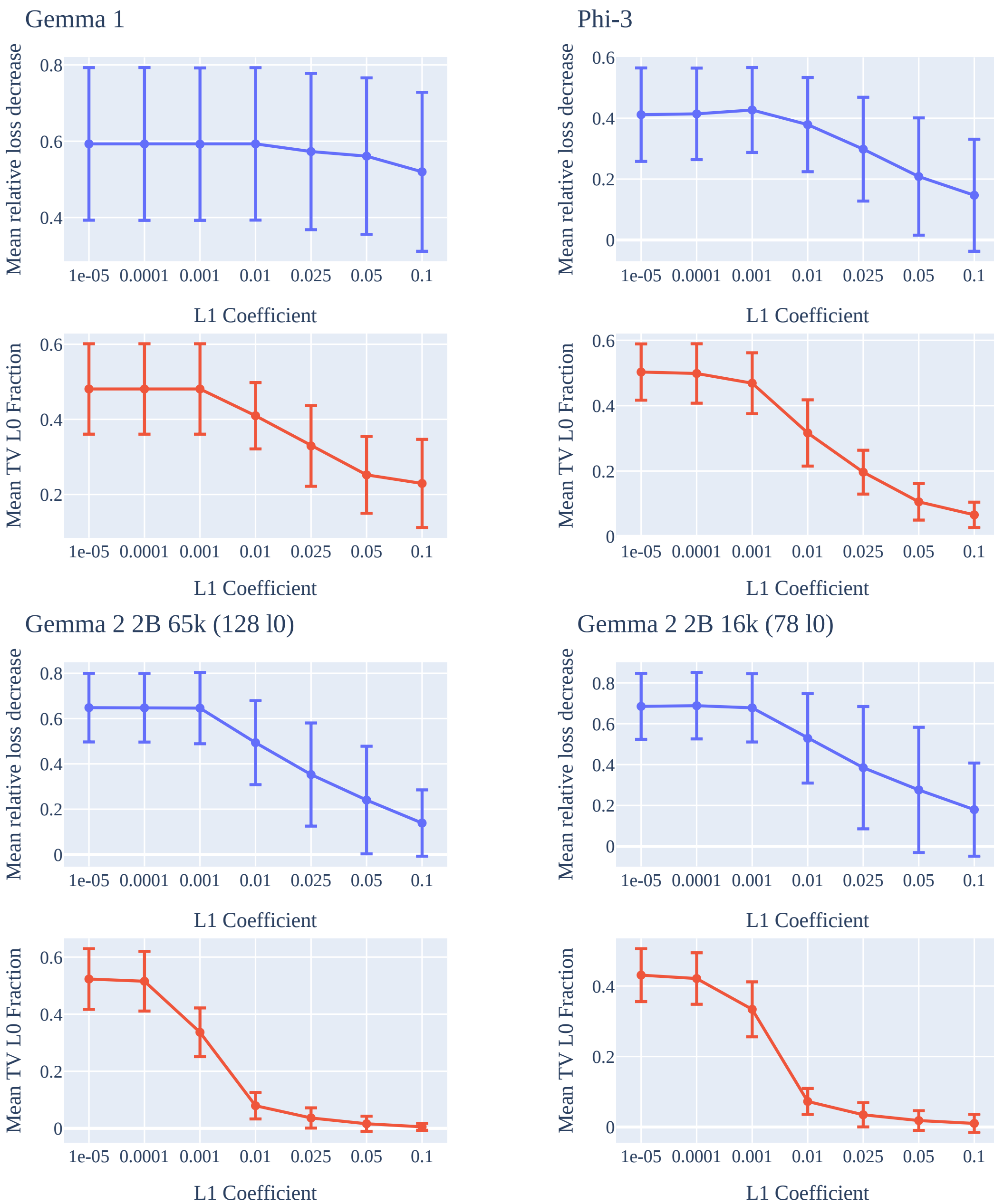


Figure 14: L_1 coefficient sweeps across different models and SAEs. All metrics are averaged across all tasks. Error bars show the standard deviation of the average for each case. Metric calculation details are available in D.1.

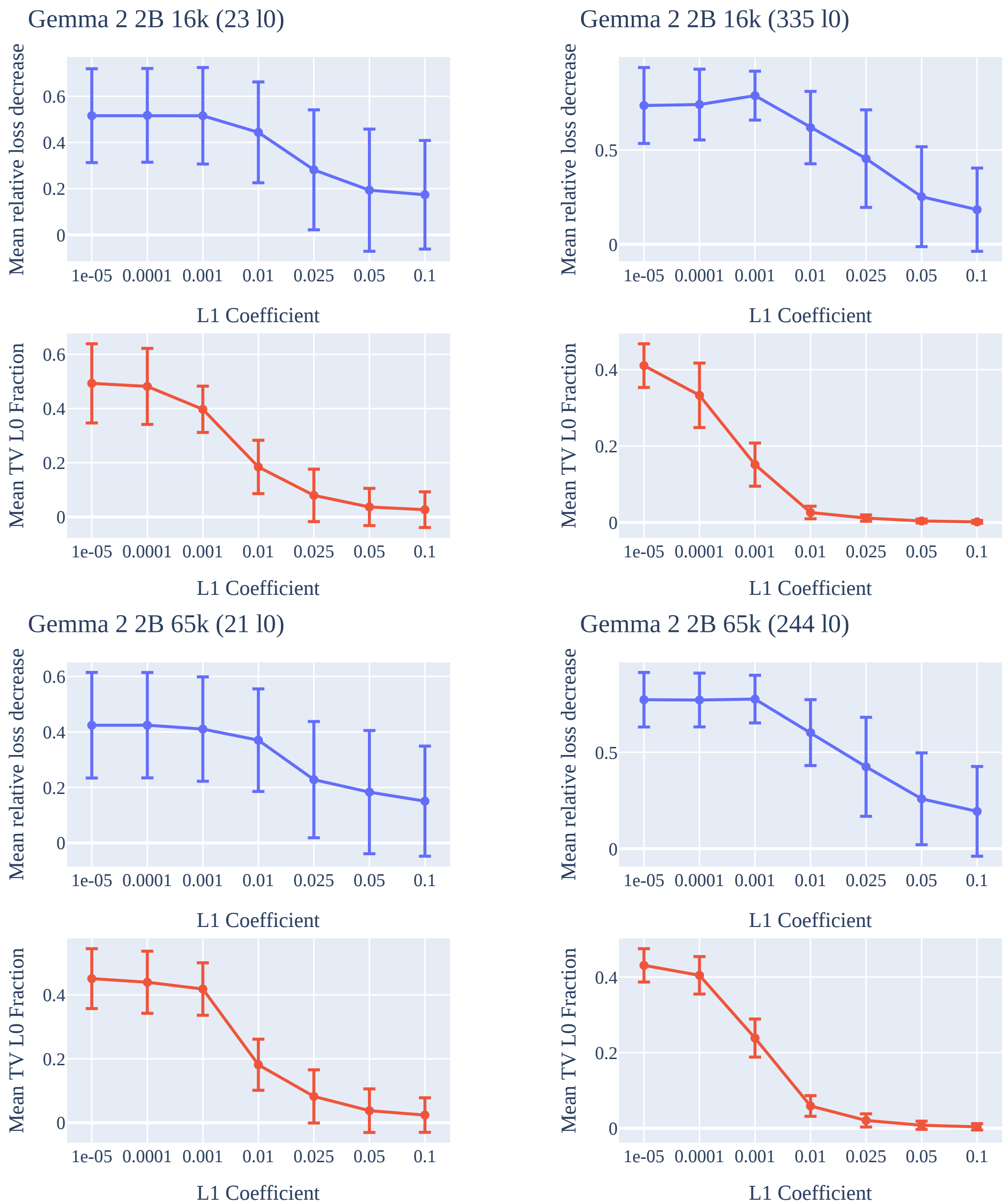


Figure 15: L_1 coefficient sweeps across different target SAE sparsities and widths for Gemma 2 2B. All metrics are averaged across all tasks. Error bars show the standard deviation of the average for each case. Metric calculation details are available in Appendix D.1.

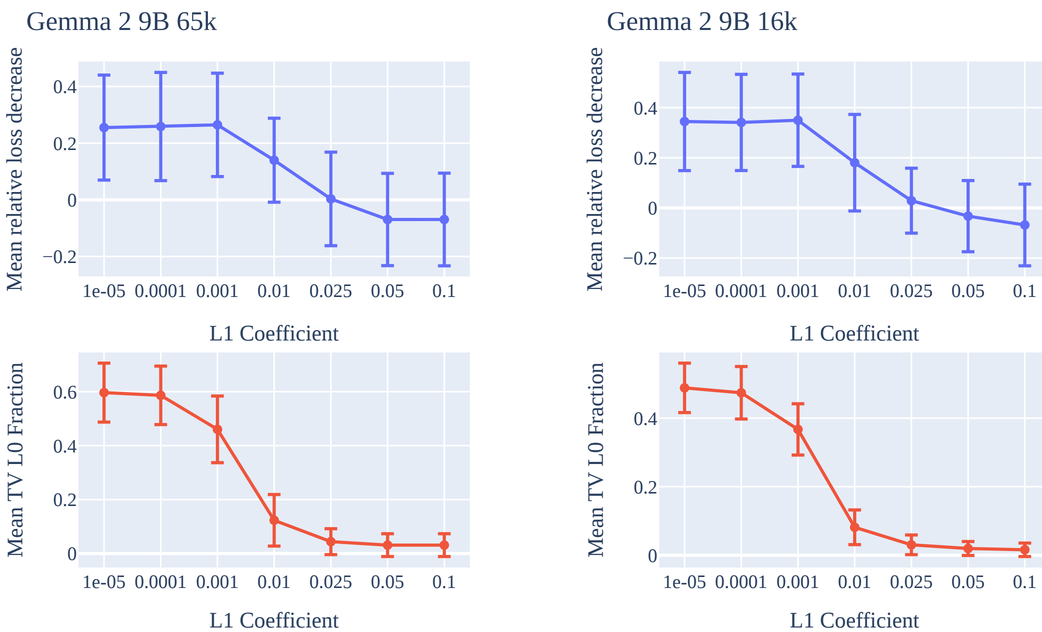


Figure 16: L_1 coefficient sweeps across two SAE widths for Gemma 2 9B. All metrics are averaged across all tasks. Error bars show the standard deviation of the average for each case. Metric calculation details are available in D.1.

from the metric. After that, we find the features with the top K indirect effects for each layer and position mask and treat them as candidates for circuit edge targets. We compute gradients with respect to the metric to the values of those nodes, propagate them to "source features" up to one layer above, and multiply by the values of the source features. This way, we can compute indirect effects for circuit edges and prune the initially fully connected circuit. However, like Marks et al. (2024), we do not perform full ablation of circuit edges.

We include a simplified implementation of node-only SFC in Algorithm 2.

```
# resids_pre: L x N x D - the pre-residual stream at layer L
# resids_mid: L x N x D - the middle of the residual stream
# (between attention and MLP) at layer L
# grads_pre: L x N x D - gradients from the metric to resids_pre
# grads_mid: L x N x D - gradients from the metric to resids_mid
# all of the above are computed with a forward and backward
# pass without SAEs

# saes_resid: L - residual stream SAEs
# saes_attn: L - attention output SAEs
# transcoders_attn: L - transcoders predicting resids_pre[l+1]
# from resids_mid[l]

def indirect_effect_for_residual_node(layer):
    sae_encoding = saes_resid[layer].encode(
        resids_pre[layer])
    grad_to_sae_latents = jax.vjp(
        saes_resid[layer].decode,
        sae_encoding
    )(grads_pre[l])
    return (grad_to_sae_latents * sae_encoding).sum(-1)

def indirect_effect_for_attention_node(layer):
    sae_encoding = saes_attn[layer].encode(
```

```
            resids_mid[layer] - resids_pre[layer])
        grad_to_sae_latents = jax.vjp(
            saes_attn[layer].decode,
            sae_encoding
        )(grads_mid[l])
        return (grad_to_sae_latents * sae_encoding).sum(-1)

    def indirect_effect_for_transcoder_node(layer):
        sae_encoding = transcoders[layer].encode(
            resids_mid[layer])
        grad_to_sae_latents = jax.vjp(
            transcoders[layer].decode,
            sae_encoding
        )(grads_pre[l+1])
        return (grad_to_sae_latents * sae_encoding).sum(-1)
```

Algorithm 2: Pseudocode for Sparse Feature Circuits indirect effect calculation.

E.2 Faithfulness charts

Figure 17 shows the average effect of node trimming on faithfulness in all tasks. We follow the methodology of Marks et al. (2024) thresholding removing nodes with low IE first. We can see that the circuits keep at least 0.8 faithfulness on average with just 1000 nodes (on layers 11-17).

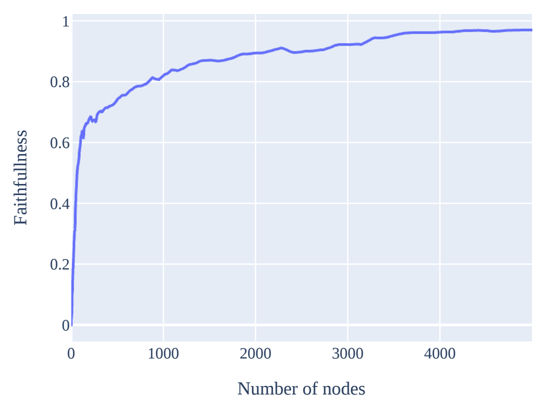


Figure 17: Average faithfulness across tasks depending on the number of nodes left in the circuit.

Figure 18 shows the averaged inverse node trimming effect on faithfulness across all tasks. Marks et al. (2024) calls this metric completeness and calculates it as the faithfulness of the model just with the circuit ablated. We calculate it by thresholding the nodes starting with those that have the highest IE. We can see that the ablation of even just several hundred nodes has a drastic impact on faithfulness. These results were also computed with the window of layers being 11-17).

F Steering with task-execution features

To evaluate the causal relevance of our identified ICL features, we conducted a series of steering experiments. Our methodology employed zero-shot prompts for task-execution features, measuring effects across a batch of 32 random pairs.

We set the target layer as 12 using Figure 3a and extracted all task-relevant features on it using our cleaning algorithm. To determine the optimal steering scale, we conducted preliminary experiments using manually identified task-execution features across all tasks. Through this process, we established an optimal steering scale of 15, which we then applied consistently across all subsequent experiments.

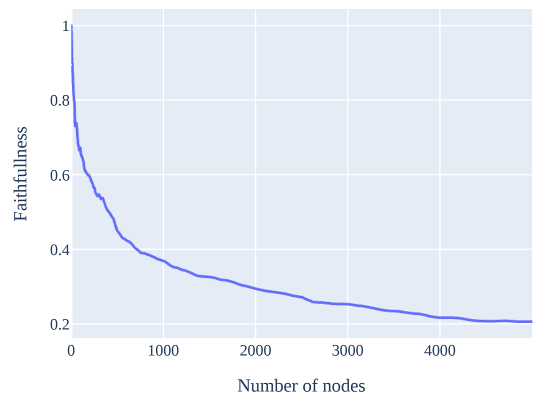


Figure 18: Average faithfulness across tasks depending on the amount of important nodes ablated from the circuit .

For each pair of tasks and features, we steered with the feature and measured the relative loss improvement compared to the model's task performance on a prompt without steering. This relative improvement metric allowed us to quantify the impact of each feature on task performance.

To normalize our results and highlight the most significant effects, we applied several post-processing steps:

- We clipped the effect to be no more than 1, thus ignoring any instances of loss increase.
- We then normalized the effects for all features within the same task to be in the 0 to 1 range.
- To remove clutter and highlight important features, we set effects lower than 0.2 to 0.
- Finally, we removed features with low maximum effect across all tasks to reduce the size of the resulting diagram. The full version of this diagram is present in Figure 19.

Prompt example with the steered token highlighted in red. Loss is calculated on the yellow token:

`BOS`	Follow	the	pattern	:	`\n`
hot	$\rightarrow$	cold			

Example 3: Task-execution steering setup. The steered token is highlighted in red and the loss is calculated on the yellow token.

We also share the version of Figure 19 without normalization and value clipping. It is present in Figure 21. We see that task vectors generally contain just a few task-execution features that can boost the task themselves. The remaining features have much weaker and less specific effects.

F.1 NEGATIVE STEERING

To further explore the effects of the executor feature, we also conducted negative steering experiments. The setup involved a batch of 16 ICL prompts, each containing 32 examples for each task. We collected all features from the cleaned task vectors for every task. Similar to positive steering, we steered with features on arrow tokens, but this time multiplying the direction by -1. Prompts this time contained several arrow tokens, and we steered on all of them simultaneously.

An important distinction from positive steering is that performance degradation in negative steering may occur due to two factors: (1) our causal intervention on the ICL circuit and (2) the steering scale being too high. To address this, we measured accuracy across all pairs in the batch instead of loss, as accuracy does not decrease indefinitely. We also observed that features no longer share a common optimal scale. Consequently, for each task pair, we iterated over several scales between 1 and 30.

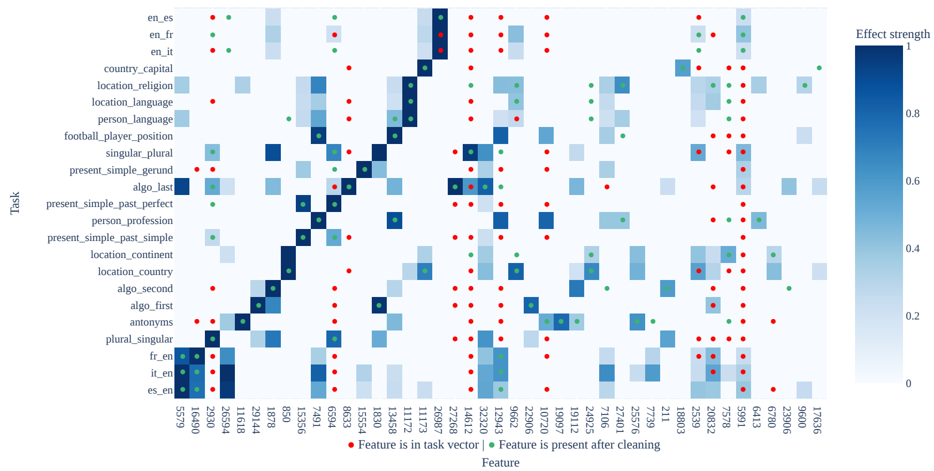


Figure 19: Full version of the heatmap in Figure 5 showing the effect of steering with individual task-execution features for each task. The features present in the task vector of the corresponding task are marked with dots. Green dots show the features that were extracted by cleaning. Red dots are features present in the original task vector. Not all original features from the task vectors are present.

For each feature, we then selected a scale that reduced accuracy by at least 0.1 for at least one task. Steering results at this scale were used for this feature across all tasks.

Figure 20 displays the resulting heatmap. While we observe some degree of task specificity — and even note that some executing features from Figure 19 have their expected effects — we also find that negative steering exhibits significantly lower task specificity. Additionally, we observe that non-task-specific features have a substantial impact in this experiment. This suggests that steering experiments alone may not suffice for a comprehensive analysis of the ICL mechanism, thus reinforcing the importance of methods such as our modification of SFC.

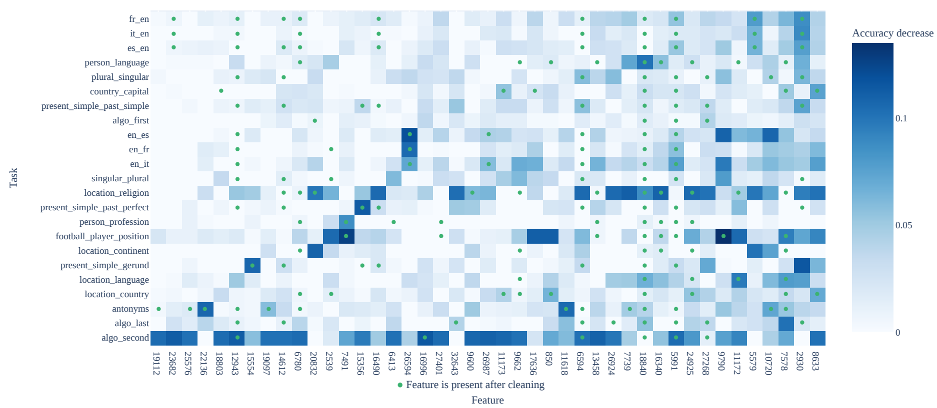


Figure 20: Negative steering heatmap. Displays accuracy decrease after optimal scale negative steering on full ICL prompts. Green circles show which features were present in the cleaned task vector of the corresponding task. More details in Appendix F.1

F.2 Gemma 2 2B positive steering

Additionally, we conducted zero-shot steering experiments with Gemma 2 2B 16k and 65k SAEs. Contrary to Gemma 1 2B, task executors from Gemma 2 2B did not have a single common optimal steering scale. Thus, we added an extra step to the experiment: for each feature and task pair, we performed steering with several scales from 30 to 300, and then selected the scale that had maximal loss decrease on any of the tasks. We then used this scale for this feature in application to all other tasks. Figure 22a and Figure 22b contain steering heatmaps for Gemma 2 2B 16k SAEs and Gemma 2 2B 65k SAEs respectively.

We observe a relatively similar level of executor task-specificity compared to Gemma 1. One notable difference between 16k and 65k SAEs is that 65k cleaned task vectors appear to contain more features with a strong effect on the task. However, this may be due to the l_1 regularization coefficient being too low.

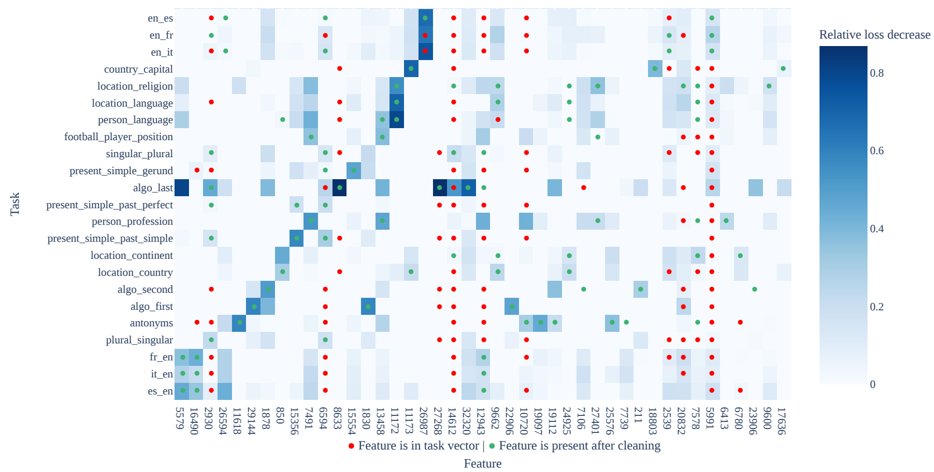


Figure 21: Unfiltered version of the heatmap in Figure 7 showing the effect of steering with individual task-execution features for each task. The features present in the task vector of the corresponding task are marked with dots. Green dots show the features that were extracted by cleaning. Red dots are the features present in the original task vector. Since the chart only contains features from cleaned task vectors, not all features from the original task vectors are present.

G Task-Detection Features

For our investigation of task-detection features, we employed a methodology similar to that used for task execution features, with a key modification. We introduced a fake pair to the prompt and focused our steering on its output. This approach allowed us to simulate the effect of the detection features the way it happens on real prompts.

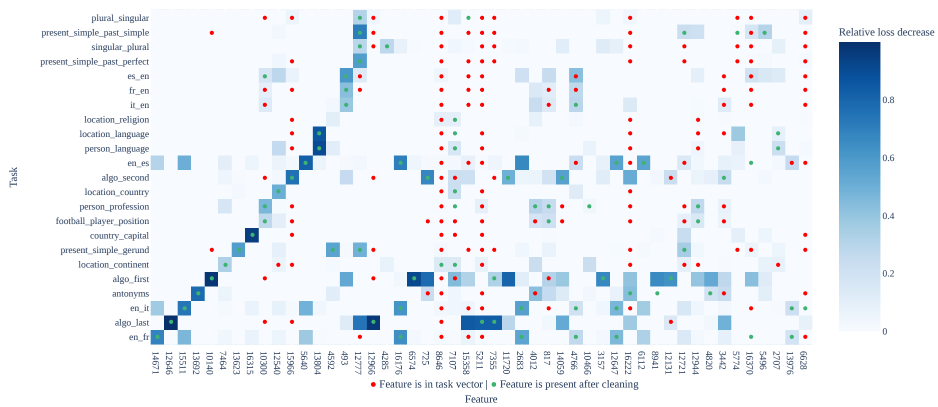
Our analysis revealed that layers 10 and 11 were optimal for task detection, with performance notably declining in subsequent layers. We selected layer 11 for our primary analysis due to its proximity to layer 12, where we had previously identified the task execution features. This choice potentially facilitates a more direct examination of the interaction between detection and execution mechanisms.

The steering process for detection features followed the general methodology outlined in Appendix F, including the use of a batch of 32 random pairs, extraction of task-relevant features, and application of post-processing steps to normalize and highlight significant effects. The primary distinction lies in the application of the steering to the prompt.

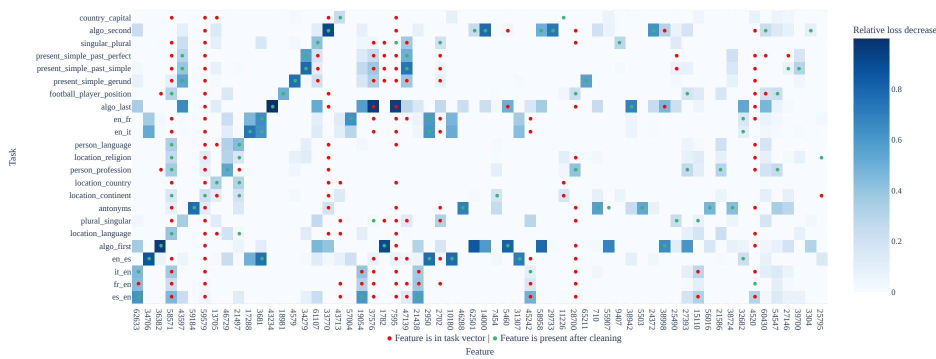
This approach allowed us to create a comprehensive representation of the causal relationships between task-detection features and the model's ability to recognize specific tasks, as visualized in Figure 7.

```
BOS  Follow  the  pattern  :  \n
X  →  Y  \n
hot  →  cold
```

Example 4: Task-detection steering setup. The steered token is highlighted in red and the loss is calculated on the yellow token.



(a) Gemma 2 2B 16k



(b) Gemma 2 2B 65k

Figure 22: Unfiltered positive steering heatmap for Gemma 2 2B SAEs showing the effect of steering with individual task-execution features for each task. Steering scales were optimized for each feature. The features present in the task vector of the corresponding task are marked with dots. Green dots show the features that were extracted by cleaning. Red dots are the features present in the original task vector. Since the chart only contains features from cleaned task vectors, not all features from the original task vectors are present.

H ICL INTERPRETABILITY LITERATURE REVIEW

This section will cover work on understanding ICL not mentioned in Section 5.

Raventós et al. provides evidence for two different Bayesian algorithms being learned for linear regression ICL: one for limited task distributions and one that is similar to ridge regression. It also intriguingly shows that the two solutions lie in different basins of the loss landscape, a phase transition necessary to go from one to the other. While interesting, it is not clear if the results apply to real-world tasks.

The existence of discrete task detection and execution features hinges on the assumption that in-context learning works by classifying the task to perform and not by learning a task. Pan et al. aims to disentangle the two with a black-box approach that mixes up outputs to force the model to learn the task from scratch. Si et al. look at biases in task recognition in ambiguous examples through a black-box lens. We find more clear task features for some tasks than others but do not consider whether this is linked to how common a task is in pretraining data.

Xie et al. proposes that in-context learning happens because language models aim to model a latent topic variable to predict text with long-range coherence. Wang et al. (2024) show following the two proposed steps rigorously improves results in real-world models. However, they do not endeavor to explain the behavior of non-finetuned models by looking at internal representations; instead, they aim to improve ICL performance.

Han et al. use a weight-space method to find examples in training data that promote in-context learning using a method akin to Grosse et al. (2023), producing results similar to per-token loss analyses in Olsson et al. (2022), and, similarly to the studies mentioned above, finds that those examples involve long-range coherence. Our method is also capable of finding examples in data that are similar to ICL, and we find crisp examples for many tasks being performed Appendix I.

Bansal et al. offers a deeper look into induction heads, scaling up Olsson et al. (2022) the way we scale up Marks et al. (2024). Crucially, it finds that MLPs in later layers cannot be removed while preserving ICL performance, indirectly corroborating our findings from Section 4.2. Chen et al. come up with a proof that states that gradient flow converges to a generalized version of the algorithm suggested by Olsson et al. (2022) when trained on n-gram Markov chain data.

Garg et al. studies the performance of toy models trained on in-context regression various *function classes*. Yadlowsky et al. find that Transformers trained on regression with multiple function classes have trouble combining solutions for learning those functions. Oswald et al. construct a set of weights for linear attention Transformers that reproduce updates from gradient descent and find evidence for the algorithm being represented on real models trained on toy tasks. Mahankali et al. proves that this algorithm is optimal for single-layer transformers on noisy linear regression data. Shen et al. questions the applicability of this model to real-world transformers. Bai et al. finds that transformers can switch between multiple different learning algorithms for ICL. Dai et al. find multiple similarities between changes made to model predictions from in-context learning and weight finetuning.

While important, we do not consider this direction of interpreting transformers trained on regression for concrete function classes through primarily white-box techniques. Instead, we aim to focus on clear discrete tasks which are likely to have individual features.

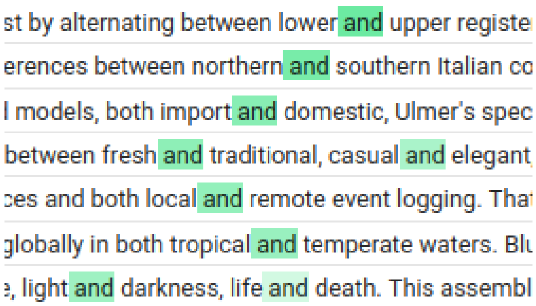
The results of Wang et al. are perhaps the most similar to our findings. The study finds "anchor tokens" responsible for aggregating semantic information, analogous to our "output tokens" (Section 2.3) and task-detection features. They tackle the full circuit responsible for ICL bottom-up and intervene on models using their understanding, improving accuracy. Like this paper, they do not deeply investigate later attention and MLP layers. Our study uses SAE features to find strong linear directions on output and arrow tokens corresponding to task detection and execution respectively, offering a different perspective. Additionally, we consider over 20 diverse token-to-token tasks, as opposed to the 4 text classification datasets considered in citewang$_{l}abel_{2}$023.

I Max Activating Examples

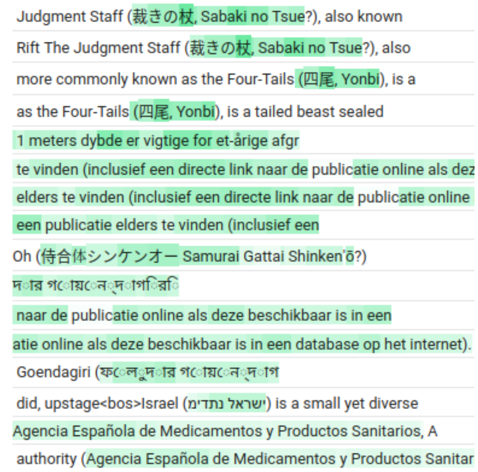
This section contains max activating examples for some executor and detector features for Gemma 1 2B, as described in (Bricken et al., 2023). They are computed by iterating over the training data distribution (FineWeb) and sampling activations of SAE features that fall within disjoint buckets for the activation value of span 0.5. We can observe that the degree of intuitive interpretability depends on the amount of task-similar contexts in the training data and SAE width.

We also provide max activating examples for Gemma 2 2B executor features from Figures 22b and 22a. These max activating examples are taken from the Neuronpedia (Lin, 2023) and are available in Figures 26 and 25.

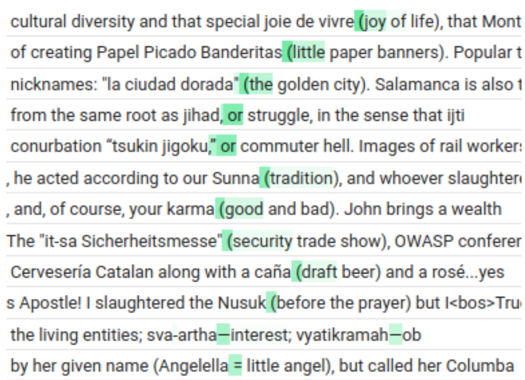
Here we can notice the main difference between executors and detectors: executors mainly activate **before the task completion**, while detectors activate on the **token that completes the task**. We also found that in Gemma 1 2B detector features for some tasks were split between several token-level features (like the journalism feature in Figure 24f), and they did not create a single feature before the task executing features activated. We attribute this to the limited expressivity of the SAEs that we used.



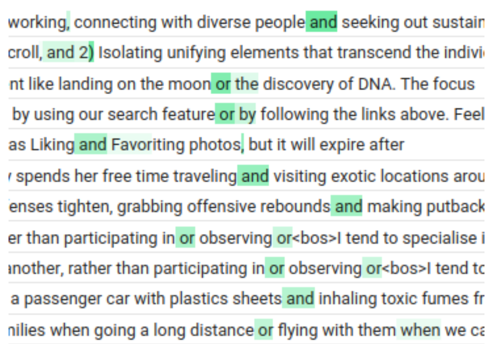
(a) Max activating examples for the antonyms executor feature 11618.



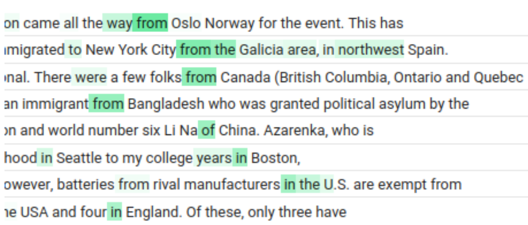
(b) Max activating examples for the English to foreign language translation executor feature 26987.



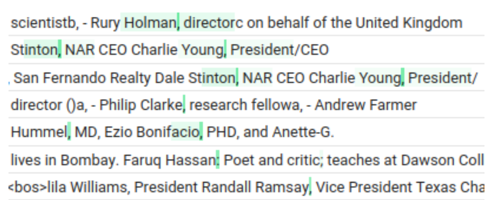
(c) Max activating examples for the translation to English executor feature 5579.



(d) Max activating examples for the "next comes gerund form" executor feature 15554.

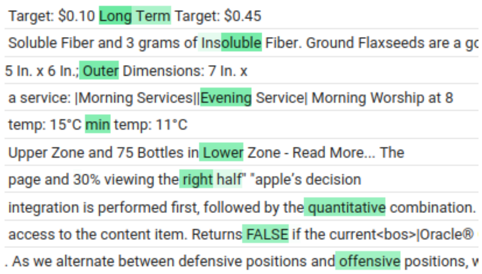


(e) Max activating examples for the prediction of city/-country feature 850.

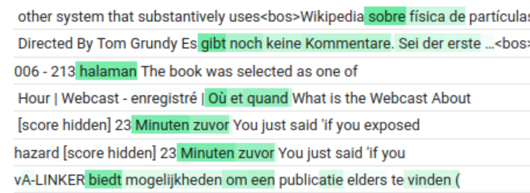


(f) Max activating examples for the person's occupation executor feature 13458.

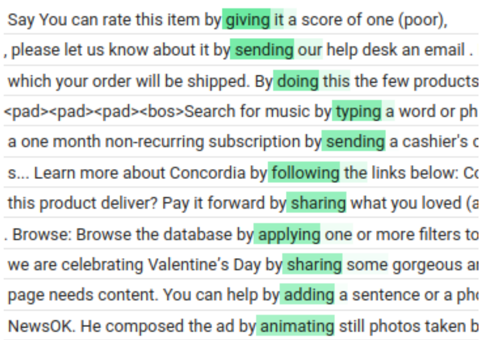
Figure 23: Max activating examples for executor features from Figure 5.



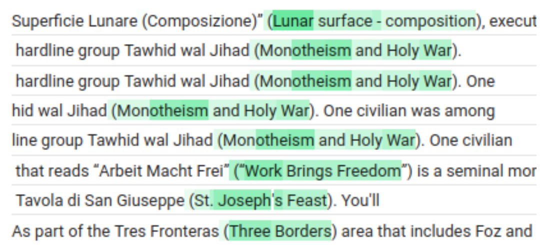
(a) Max activating examples for the antonyms detector feature 11050.



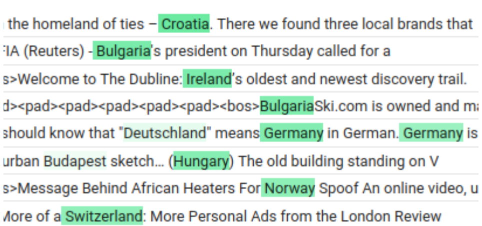
(b) Max activating examples for the English to foreign language switch detector feature 7928.



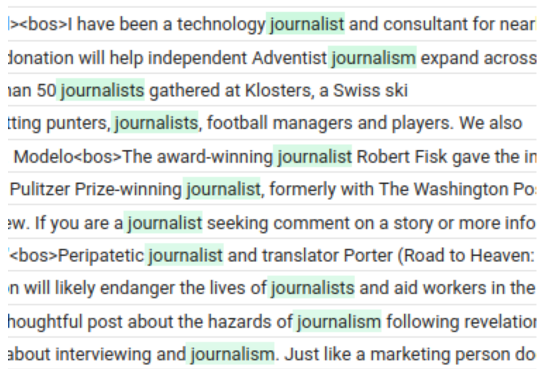
(c) Max activating examples for the gerund form detector feature 8446.



(d) Max activating examples for the translation to English detector feature 31123.



(e) Max activating examples for the country detector feature 11459.



(f) Max activating examples for the journalist feature 26436. (The strongest detector for the person_profession task).

Figure 24: Max activating examples for detector features from the Figure 7

means lost and found in the Mandingo language
no probleme i can speak english↵<q
because he could not speak Mandarin or Cantonese.
A. Heim. In German & French.
"market field" in Malay. It was
of her team speak in English, Elena immediately
international projects. I speak english, spanish and
" xml:lang="en">↵↵<

(a) Max activating examples for the language prediction executor feature 13804.

anyway, dogs will be dogs. My name
hook. Boys will be boys, I suppose
really hate explicit shock for the sake of shock
ExtractionOptionsJsonObject::ExtractionOptionsJsonObject()
Vertice3f::Vertice3f
Vertice3f::Vertice3f
four ways, and only four ways, in
ProfilerJniMethod::SamplingProfilerJni
particular device—and only for that device.

(b) Max activating examples for the repetition executor feature 12646. Extracted from the algo_last TV.

in the Swazi capital, Mbabane
In the densely populated capital of Monrovia,
km north of the capital Kabul. A crowd
east of the regional capital Poznań.↵↵References
south of the regional capital Bialy
west of the regional capital Wrocław.↵↵References
ero neighbourhood in the capital, Bujumbura
the smells that filled downtown Greeneville in December

(c) Max activating examples for the capital prediction executor feature 16315.

st issue of Norsk Entomologisk Tidsskrift (now Norwegian Journal of Entomology) appeared in May 1921.
vision and administration of the Dirección Provincial de Vialidad (Provincial Dept. of Transportation).↵
↵Insekt-Nytt (Insect News) is written in a popular science style and is the society's
eja Universal do Reino de Deus (Universal Church of the Kingdom of God) denied involvement in the scandal.
Theatre in the play Den Sorte Dronning (The Black Queen) in 1843. Many artist frequented the
icts Lithuanian Žalioji rinktinė (The Green Squad), belonging to partisans' Algimantas military distric
rvals.↵Norske Insekttabeller (Norwegian Insect Tables) is a series of inexpensive Norwegian-

(d) Max activating examples for the translation feature 493.

Figure 25: Max activating examples for Gemma 2 2B 16k executor features from the Figure 22a

apparent use of compliant and non-compliant form
die soon, today, tomorrow, or in
various degrees of reluctant and unlikely. There is
no matter how different or diverse these may be
: What are reliable and trusted websites? How
are clearly upsides and downsides for those companies
bed, standing up, falling back down,

(a) Max activating examples for the antonyms executor feature 45288.

), Judd Ringer (right end), George Benson
plays as a winger or as a left back
Castilla as either a central defender or a left
, it's right tackle. Filling in

(b) Max activating examples for the football_player_position executor feature 18981.

964), English rugby player↵↵See
014), American racing driver↵W
936), British composer, conductor and
4) was an English sportsman who played rugby
1) was an Italian footballer from Bastia

(c) Max activating examples for the person_profession executor feature 46729.

Mai-Mai Kata Katanga ("Secede Katanga").↵↵Other Mai-Mai groups↵↵There was a large Mai
or Low Saxon, i.e. that they remain for ever together undivided). Christian's ascension
an Žalioji rinktinė (The Green Squad), belonging to partisans' Algimantas military distri
jed square called Pasar Medan – literally, "market field" in Malay. It was here that the cit
ey had a plastic kagami mochi, which translates to mirror rice cake. Basically a snowman m
and Roberto Jefferson. The Ministério Público Federal (the Federal Prosecutor's Office)

(d) Max activating examples for translation to English executor feature 62633.

Figure 26: Max activating examples for Gemma 2 2B 65k executor features from the Figure 22b