arXiv:2506.20790v2 [cs.LG] 4 Sep 2025

Stochastic Parameter Decomposition

Lucius Bushnaq[*] lucius@goodfire.ai
Dan Braun[*] dan.braun@goodfire.ai
Lee Sharkey[†] lee@goodfire.ai

Abstract

A key step in reverse engineering neural networks is to decompose them into simpler parts that can be studied in relative isolation. Linear parameter decomposition—a framework that has been proposed to resolve several issues with current decomposition methods—decomposes neural network parameters into a sum of sparsely used vectors in parameter space. However, the current main method in this framework, Attribution-based Parameter Decomposition (APD), is impractical on account of its computational cost and sensitivity to hyperparameters. In this work, we introduce *Stochastic Parameter Decomposition* (SPD), a method that is more scalable and robust to hyperparameters than APD, which we demonstrate by decomposing models that are slightly larger and more complex than was possible to decompose with APD. We also show that SPD avoids other issues, such as shrinkage of the learned parameters, and better identifies ground truth mechanisms in toy models. By bridging causal mediation analysis and network decomposition methods, this demonstration opens up new research possibilities in mechanistic interpretability by removing barriers to scaling linear parameter decomposition methods to larger models. We release a library for running SPD and reproducing our experiments at https://github.com/goodfire-ai/spd/tree/spd-paper.

1 Introduction

We have little understanding of the internal mechanisms that neural networks learn that enable their impressive capabilities. Understanding—or reverse engineering—these mechanisms may enable us to better predict and design neural network behavior and propensities for the purposes of safety and control. It may also be useful for scientific knowledge discovery: Neural networks can often perform better than humans on some tasks. They must therefore 'know' things about the world that we do not know—things that we could uncover by understanding their mechanisms.

An important first step to reverse engineering neural networks is to decompose them into individual mechanisms whose structure and interactions can be studied in relative isolation. Previous work has taken a variety of approaches to network decomposition. A popular approach is sparse dictionary learning (SDL) [Cunningham et al., 2024, Bricken et al., 2023], which aims to decompose neural network activations by optimizing sparsely activating dictionary elements to reconstruct or predict neural activation vectors. However, this approach suffers from a range of conceptual and practical problems, such as failing to account for feature geometry [Leask et al., 2025, Mendel, 2024] and not decomposing networks into functional components [Chanin et al., 2024, Bricken et al., 2023, Till, 2024] (see Sharkey et al. [2025] for a review).

Recently, *linear parameter decomposition* [Braun et al., 2025], has been proposed to address some of the issues faced by SDL and other current approaches. Instead of decomposing networks into directions in activation space, linear parameter decomposition methods decompose networks into *vectors in parameter space*, called *parameter components*. Parameter components are selected such that, simultaneously, (a) they sum to the parameters of the original model, (b) as few as possible

[*]Goodfire – Work primarily carried out while at Apollo Research
[†]Goodfire

are required to replicate the network's behavior on any given input, and (c) they are as 'simple' as possible. This approach promises a framework that suggests solutions to issues like 'feature splitting' [Chanin et al., 2024, Bricken et al., 2023]; the foundational conceptual issue of defining a 'feature' (by re-basing it in the language of 'mechanisms'); and the issues of multidimensional features and feature geometry [Braun et al., 2025]. It also suggests a new way to bridge mechanistic interpretability and causal mediation analysis [Mueller et al., 2024].

However, *Attribution-based Parameter Decomposition* (APD) [Braun et al., 2025], the only method that has been so far proposed for linear parameter decomposition (with which this paper assumes some familiarity), suffers from several significant issues that hinder its use in practice, including:

1. **Scalability**: APD has a high memory cost, since it decomposes a network into many parameter components, each of which is a whole vector in parameter space. They therefore each have the same memory cost as the original network.
2. **Sensitivity to hyperparameters**: In the toy models it was tested on, APD only recovers ground-truth mechanisms for a very narrow range of hyperparameters. In particular, APD requires choosing the top-k hyperparameter, the expected number of active parameter components per datapoint, which would usually not be known in advance for non-toy models. As discussed in Braun et al. [2025], choosing a value for top-k that is too high or low makes it difficult for APD to identify optimal parameter components.
3. **Use of attribution methods**: APD relies on attribution methods (e.g. gradient attributions, used in Braun et al. [2025]), to estimate the causal importance of each parameter component for computing the model's outputs on each datapoint. Gradient-based attributions, and attribution methods more generally, are often poor approximations of ground-truth causal importance [Syed et al., 2024] and sometimes fail to pass basic sanity checks [Adebayo et al., 2018].

In this work, we introduce a new method for linear parameter decomposition that overcomes all of these issues: *Stochastic Parameter Decomposition* (SPD) (Section 2).

Our approach decomposes each matrix in a network into a set of rank-one matrices called *subcomponents*. The number of rank-one matrices can be higher than the rank of the decomposed matrix. Subcomponents are not full parameter components as in the APD method, but they can later be aggregated into full components. In this work, we use toy models with known ground-truth mechanisms, where the clusters are therefore straightforward to identify. However, in future it will be necessary to algorithmically cluster these components in cases where ground truth is not known.

Instead of relying on attribution techniques and a top-k hyperparameter that needs to be chosen in advance, we define the *causal importance* of a subcomponent as how ablatable it is on a given datapoint. Causally important subcomponents should not be ablatable, and ablatable subcomponents should be causally unimportant for computing the output. We train a *causal importance function* to predict the causal importance $g_i \in [0, 1]$ of subcomponent i for computing the model's output on a given datapoint and use the predicted causal importances to ablate unimportant subcomponents by random amounts by masking them multiplicatively with a random scalar sampled from a uniform distribution $\mathcal{U}(g_i, 1)$. We train a model that is parametrized by the sum of these randomly masked subcomponents to compute the same output as the target model. Crucially, we regularize the predicted causal importance values g_i to be close to zero, so that as many subcomponents as possible will be predicted to be ablatable on any given datapoint. The core intuition behind this training setup is that all combinations of ablating the subcomponents are checked with some probability. And since the causal importance function is penalized for outputting large values of g, it should only output high values of g_i when subcomponent i is really 'used' by the network.

We apply SPD to all of the toy models that Braun et al. [2025] used to study APD, including: A Toy Model of Superposition [Elhage et al., 2022] (Section 3.1); a Toy Model of Compressed Computation [Braun et al., 2025] (Section 3.3); and a Toy Model of Cross-Layer Distributed Representations (Section 3.4). We demonstrate that the method recovers ground-truth mechanisms in all of these models. We also extend the suite of models to include two more challenging models where APD struggles but SPD succeeds: A Toy Model of Superposition with an additional identity matrix in the hidden space (Section 3.2) and a deeper Toy Model of Cross-Layer Distributed Representations (Section 3.4). Using APD, these new models were unmanageably difficult to correctly decompose, but SPD succeeds with relative ease.

The successful application of SPD to more challenging models demonstrates that SPD is more scalable and stable than APD. Nevertheless, some challenges remain: Firstly, the method needs to be scaled to larger models, which will likely require further improvements in training stability. Second, SPD only finds rank-one components in individual layers, meaning that further clustering step is required to find components that span more than one rank and/or more than one layer. In the toy models presented in this paper, these clusters are known and are therefore straightforward to identify. However, a general clustering solution will be needed in order to find such components where ground-truth is unknown. Despite these challenges, SPD opens up new research avenues for mechanistic interpretability by introducing a linear parameter decomposition method that removes the main barriers to scaling to larger, non-toy models such as language models (Section 5).

2 Method: Stochastic Parameter Decomposition

Suppose we have a trained neural network $f(x, W)$ that maps inputs x to outputs $y = f(x, W)$, parametrized by a set of weight matrices[1] $W = \{W^1, \ldots, W^L\}$. This set W can also be represented as a single vector in a high-dimensional vector space called *parameter space*. Linear parameter decomposition methods such as APD aim to decompose neural network parameters into a set of *parameter components*, which are vectors in parameter space that are trained to exhibit three desirable properties [Braun et al., 2025]:

- **Faithfulness:** The parameter components should sum to the parameters of the original network.
- **Minimality:** As few parameter components as possible should be used by the network for a forward pass of any given datapoint in the training dataset.
- **Simplicity:** Parameter components should use as little computational machinery as possible, in that they should span as few matrices and as few ranks as possible.

If a set of parameter components exhibit these three properties, we say that they comprise the network's *mechanisms*[2]. In APD, gradient-based attributions are used to estimate the importance of each parameter component for a given datapoint. Then, the top-k most important parameter components are summed together and used for a second forward pass. These active parameter components are trained to produce the same output on that datapoint as the target model. Simultaneously, the parameter components are trained to sum to the parameters of the target model, and are trained to be simple by penalizing the sum of the spectral p-norms of their individual weight matrices, encouraging them to be low-rank.

In our work, we aim to identify parameter components with the same three properties, but we achieve it in a different way.

2.1 Our approach optimizes rank-one subcomponents instead of full-rank parameter components

A major issue with the APD method is that it is computationally very expensive: It involves optimizing L full-rank matrices for every parameter component (where L is the number of matrices in the model). But this is wasteful if we expect most parameter components to be low-rank and localized only to a subset of layers. With SPD, instead of using full-rank parameter components that span every layer, we decompose each of a neural network's weight matrices $W^1, \ldots, W^L$ into a set of C rank-one matrices called *subcomponents*, $\vec{U}_c^l \vec{V}_c^{l\top}$:

$$W_{i,j}^l \approx \sum_{c=1}^{C} U_{i,c}^l V_{c,j}^l \,. \tag{1}$$

[1] As in Braun et al. [2025], we do not decompose biases. Biases can be folded into the weights by treating them as an additional column in each weight matrix, meaning they can in theory be decomposed like any other type of parameter. However, in this work, for simplicity we treat them as their own parameter component that is active for every input, and leave their decomposition for future work.

[2] Note that these properties can trade off against each other. Therefore, in practice, we quantify how much we care about each property, and find the set of parameter components that minimise the resulting overall loss.

Here, l indexes the neural network's matrices and i, j are its hidden indices. We will later cluster these subcomponents into full parameter components if they tend to co-activate together. In this paper, the groups are easy to identify and this clustering process is implicit. However, future work will require an explicit clustering algorithm for cases where groups of subcomponents are harder to identify.

Note that the number of subcomponents in each layer C may be larger than the minimum of the number of rows or columns of the matrix at that layer, thus enabling SPD to identify computations in superposition.

2.2 Optimizing for faithfulness

The way we optimize for faithfulness is the same as in Braun et al. [2025], by optimizing the sum of our subcomponents to approximate the parameters of the target model:

$$\mathcal{L}_{\text{faithfulness}} = \frac{1}{N} \sum_{l=1}^{L} \sum_{i,j} \left(W^l_{i,j} - \sum_{c=1}^{C} U^l_{i,c} V^l_{c,j} \right)^2, \tag{2}$$

where N is the total number of parameters in the target model.

2.3 Optimizing for minimality and simplicity by learning a causal importance function to stochastically sample masks

The way we optimize for minimality and simplicity is different from Braun et al. [2025]. Since we already start with rank-one subcomponents that are localized in single layers, we don't need to optimize for subcomponent simplicity. Instead, we only need to train our set of subcomponents such that as few as possible are "*active*" or "*used*" or "*required*" by the network to compute its output on any given datapoint in the training set. We consider this equivalent to requiring that as few subcomponents as possible be *causally important* for computing the network's output.

To optimize our set of subcomponents such that as few as possible are causally important for computing the model's output on any given datapoint, we have three requirements:

1. A formal definition of what it means for a subcomponent to be 'causally important' for computing the model's outputs (Section 2.3.1);
2. A loss function that trains causally important subcomponents to compute the same function as the original network (Section 2.3.2);
3. A loss function that encourages as many subcomponents as possible to be causally *un*-important on each datapoint (Section 2.3.3).

2.3.1 Requirement 1: Formally defining a subcomponent's *causal importance* as the extent to which it can be ablated

Intuitively, we say a subcomponent is *causally important* on a particular datapoint x if and only if it is required to compute the model's output for that datapoint. Conversely, we say a subcomponent is *causally unimportant* if it can be ablated by some arbitrary amount while leaving the model's output unchanged. In particular, if a component is fully unimportant, then it shouldn't matter how much we ablate it by; we should be able to fully ablate all causally unimportant components, or only partially ablate them, or ablate only a subset of them, and still get the same model output. Note, the distinction between causally important and unimportant is not binary; we can say that a subcomponent is causally unimportant *to the extent* that it can be ablated without affecting the model's output.

Formally, suppose $g^l_c(x) \in [0, 1]$ indicates the causal importance of subcomponent (c) of weight matrix l on a given datapoint x. For now, we take this quantity as given, but later we discuss how we obtain it. In general, we want to get the same model output for any weight matrices along all monotonic 'ablation curves' $m^l_c(x, r)$ that interpolate between the original model and the model with

all inactive subcomponents ablated. Here, r is a vector sampled such that:

$$\begin{aligned}
&r_c^l \in [0,1] \\
&m_c^l(x,r) := g_c^l(x) + (1 - g_c^l(x))r_c^l \\
&W'^l_{i,j}(x,r) := \sum_{c=1}^{C} U_{i,c}^l m_c^l(x,r) V_{c,j}^l \\
&\forall r : f(x|W'^1(x,r),\ldots,W'^L(x,r)) \approx f(x|W^1,\ldots,W^L)\,.
\end{aligned} \tag{3}$$

Now we need a differentiable loss function(s) in order to be able to train the masked model $f(x|W'^1(x,r),\ldots,W'^L(x,r))$ to approximate the target model $f(x|W^1,\ldots,W^L)$ for all values of $r \in [0,1]^{C\times L}$.

2.3.2 Requirement 2: A loss function that lets us optimize causally important subcomponents to approximate the same function as the original network

Unfortunately, calculating $f(x|W'^1(x,r),\ldots,W'^L(x,r))$ for all values of $r \in [0,1]^{C\times L}$ is computationally intractable because there are infinitely many possible values. But we can calculate it for a randomly chosen subset of values. To do this, we uniformly sample S points, $r_c^{l,(s)} \sim \mathcal{U}(0,1)$ and mask the subcomponents with the resulting stochastic masks $m_c^l(x,r^{(s)})$. Then, as $S \to \infty$, the following loss approximately optimizes $f(x|W'^1(x,r),\ldots,W'^L(x,r))$ to satisfy Equation 3:

$$\mathcal{L}_{\text{stochastic-recon}} = \frac{1}{S}\sum_{s=1}^{S} D\left(f(x|W'(x,r^{(s)})), f(x|W)\right) \tag{4}$$

Here, D is some appropriate divergence measure in the space of model outputs, such as KL-divergence for language models, or MSE loss.

However, this loss can be somewhat noisy because it involves a forward pass in which every subcomponent has been multiplied by a random mask. Therefore, in addition to this loss, we also use an auxiliary loss $\mathcal{L}_{\text{stochastic-recon-layerwise}}$, which is simply a layerwise version of $\mathcal{L}_{\text{stochastic-recon}}$ where only the parameters in a single layer at a time are replaced by stochastically masked subcomponents. The gradients are still calculated at the output of the model:

$$\mathcal{L}_{\text{stochastic-recon-layerwise}} = \frac{1}{LS}\sum_{l=1}^{L}\sum_{s=1}^{S} D\left(f(x|W^1,\ldots,W'^l(x,r^{l,(s)}),\ldots,W^L), f(x|W)\right) \tag{5}$$

This should not substantially alter the global optimum of training, because Equation 5 is equivalent to Equation 4 if the subcomponents sum to the original weights and if we sample $r_c^l = 1$ for all layers except one at a time.

2.3.3 Requirement 3: A loss function that encourages as many subcomponents as possible to be causally unimportant

We have not yet defined how we obtain subcomponents' causal importance values $g_c^l(x)$. Measuring this directly would be intractable, so we learn a function to predict it.

In theory, we could use any arbitrary *causal importance function* $\Gamma : X \to [0,1]^{C\times L}$ to predict causal importance values. In practice, we use a set of independent functions Γ_c^l, one for each subcomponent. Each function consists of a very small trained MLP γ_c^l (see Appendix A.1 for its architecture). Each MLP takes as input a scalar, the subcomponent's 'inner activation', $h_c^l(x) := \sum_j V_{c,j}^l a_j^l(x)$, where $\vec{a^l(x)}$ is the activation vector in the target model that gets multiplied by weight matrix W^l. The MLP outputs are passed to a hard sigmoid σ_H to get a prediction of the subcomponent's causal

importance[3]:

$$g_c^l(x) = \Gamma_c^l(x) = \sigma_H(\gamma_c^l(h_c^l(x))) \tag{6}$$

The output of the causal importance function for each subcomponent is therefore a single scalar number that should be in the range $[0, 1]$.

While this form of causal importance function works well for the toy models in this paper, it is likely that the best causal importance functions for arbitrary models require more expressivity. For example, a subcomponent's causal importance function may take as input the inner activations of all subcomponents, rather than just its own. Such variants may be explored in future work.

The causal importance values $g_c^l(x)$ are used to sample masks that randomly ablate each subcomponent $m_c^l(x, g_c^l(x)) \sim \mathcal{U}(g_c^l(x), 1)$. We use the reparametrization trick [Kingma and Welling, 2013](Equation 3) to allow gradients to be backpropagated through the masks $m_c^l(x, g_c^l(x))$ to train the causal importance functions Γ_c^l. The causal importance functions can therefore learn to produce masks that better predict the ablatability of each subcomponent on a given datapoint. However, if the causal importance functions were trained using only the $\mathcal{L}_{\text{stochastic-recon}}$ loss, they could perform optimally (but pathologically) by outputting a causal importance value of $g_c^l(x) = 1$ for every subcomponent on every input, since there is no incentive to learn to predict the full extent of the ablatability of the subcomponents. An additional loss is required to encourage the causal importance function to always try to ablate as much as possible. We therefore penalize causal importance values for being above 0 using a $\mathcal{L}_{\text{importance-minimality}}$ loss:

$$\mathcal{L}_{\text{importance-minimality}} = \sum_{l=1}^{L} \sum_{c=1}^{C} |g_c^l(x)|^p \,, \tag{7}$$

where $p > 0$ [4].

2.4 Summary of training setup

Our full loss function consists of four losses:

$$\mathcal{L}_{\text{SPD}} = \mathcal{L}_{\text{faithfulness}} + (\beta_1 \mathcal{L}_{\text{stochastic-recon}} + \beta_2 \mathcal{L}_{\text{stochastic-recon-layerwise}}) + \beta_3 \mathcal{L}_{\text{importance-minimality}} \tag{8}$$

Our training setup involves five hyperparameters (excluding optimizer hyperparameters such as learning rate): The coefficients $\beta_1, \beta_2, \beta_3$ for each of the losses; the p-norm used in $\mathcal{L}_{\text{importance-minimality}}$ and the number of mask samples S that are used for each training step, for which we find $S = 1$ sufficient. We discuss some heuristics for choosing the loss coefficients in Appendix A.3. Pseudocode for the SPD algorithm can be found in Appendix A.6.

3 Results

We apply SPD to decompose a set of toy models with known ground-truth mechanisms. Some of these models were previously studied by Braun et al. [2025] to evaluate APD, while others are new. We study:

[3]We use hard sigmoids rather than standard sigmoids because we would like to make it possible for causal importance values to take values of exactly 0 or 1 or anywhere in between; a standard sigmoid function only tends toward 0 or 1 as its input tends toward $-\infty$ or ∞ respectively. However, note that hard sigmoids have large regions where gradients are 0. We therefore actually use leaky-hard sigmoids to increase training stability. Outputs may therefore lie slightly outside of the $[0, 1]$ range (Appendix A.2).

[4]It is important to note that $\mathcal{L}_{\text{importance-minimality}}$ is somewhat different from the L_p penalties often used to sparsify latents in SDL, where p is restricted to $(0, 1]$. We find that training with $\mathcal{L}_{\text{importance-minimality}}$ successfully minimizes importance values even when $p > 1$. We believe that this happens for the following reason: If $p > 1$ in SDL, a single active feature can always be split into many active features to improve the sparsity loss. However, if $p > 1$ in SPD, then if one causally important subcomponent were pathologically split into two, both resulting subcomponents would still have causal importance of ≈ 1, since both are still needed to maintain low $\mathcal{L}_{\text{stochastic-recon}}$, meaning $\mathcal{L}_{\text{importance-minimality}}$ would increase.

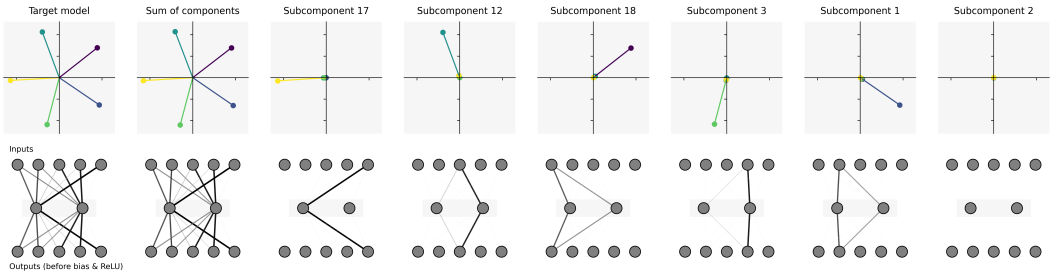

Figure 1: Results of running SPD on TMS_{5-2}. **Top row:** Plots of (left to right) the columns of the weight matrix of the target model; the sum of the SPD parameter components; and individual parameter components. Although this run of SPD used 20 subcomponents, only 6 subcomponents are shown, ordered by the sum of the norms of each of the columns of their (rank-one) weight matrices. The first five have learned one direction each, each corresponding to one of the columns of the target model. The final column and the other 14 components (not shown) have a negligible norm because they are superfluous for replicating the behavior of the target model. **Bottom row:** Depiction of the corresponding parametrized networks.

	MMCS	ML2R
TMS_{5-2}	1.000 ± 0.000	0.993 ± 0.002
TMS_{40-10}	1.000 ± 0.000	1.010 ± 0.007
$\text{TMS}_{5-2+\text{ID}}$	1.000 ± 0.000	0.992 ± 0.010
$\text{TMS}_{40-10+\text{ID}}$	1.000 ± 0.000	1.031 ± 0.001

Table 1: Mean Max Cosine Similarity (MMCS) and Mean L2 Ratio (ML2R) with their standard deviations (to 3 decimal places) between learned parameter subcomponents and the target model weights in the subcomponents found by SPD for the embedding matrix W matrix in the TMS_{5-2} and TMS_{40-10} models (Section 3.1) and $\text{TMS}_{5-2+\text{ID}}$ and $\text{TMS}_{40-10+\text{ID}}$ models (Section 3.2). These results indicate that the ground truth mechanisms are recovered perfectly and with negligible shrinkage for all models.

1. A **Toy Model of Superposition (TMS)** [Elhage et al., 2022] previously studied by Braun et al. [2025] (Section 3.1);
2. A **TMS model with an identity matrix** inserted in the middle of the model (not previously studied) (Section 3.2);
3. A **Toy Model of Compressed Computation** previously studied by Braun et al. [2025] (Section 3.3);
4. Two **Toy Models of Cross-Layer Distributed Representations**: One with two residual MLP blocks (previously studied by Braun et al. [2025]) and one with three MLP blocks (not previously studied) (Section 3.4);

In all cases, we find that SPD seems to identify known ground-truth mechanisms up to a small error. For the models that were also decomposed with APD in Braun et al. [2025], we find that the SPD decompositions have fewer errors despite requiring less hyperparameter tuning to find the ground-truth mechanisms.

Code to reproduce our experiments can be found at `https://github.com/goodfire-ai/spd/tree/spd-paper`. Training details and hyperparameters can be found in Appendix A.4. Additional figures and training logs can be found in the WandB report here.

3.1 Toy Model of Superposition

We decompose Elhage et al. [2022]'s Toy Model of Superposition (TMS), which can be written as $\hat{x} = \text{ReLU}(W^\top W x + b)$, with weight matrix $W \in \mathbb{R}^{m_1 \times m_2}$. The model is trained to reconstruct its inputs, which are sparse sums of one-hot m_2-dimensional input features whose activations are scaled

to a random uniform distribution $[0, 1]$. Typically, $m_1 < m_2$, so the model is forced to 'squeeze' representations through a m_1-dimensional bottleneck. When the model is trained on sufficiently sparse data distributions, it can learn to represent features in superposition in this bottleneck. For certain values of m_1 and m_2, the columns of the W matrix can form regular polygons in the m_1-dimensional hidden activation space (Figure 1 - Leftmost panel).

The ground truth mechanisms in this model should be a set of rank-1 matrices that are zero everywhere except in the cth column, where they take the values $\vec{W}_{:,c}$. Intuitively, a column of W is only 'used' if the corresponding input feature is active; other columns can be ablated without significantly affecting the output of the model.

SPD Results: Toy Model of Superposition

We apply SPD to TMS models with 5 features in 2 dimensions, denoted TMS_{5-2} and 40 features in 10 dimensions, denoted TMS_{40-10}. In both cases, SPD successfully decomposes the model into subcomponents that closely correspond to the columns of W (Figure 1). This result is robust to different training seeds and required less hyperparameter tuning than APD [Braun et al., 2025].

We quantify how aligned the learned parameter components vectors are to the columns of W in the target model using the mean max cosine similarity (MMCS) [Sharkey et al., 2022]. The MMCS measures alignment between each column of W and the corresponding column in the parameter component that it best aligns with:

$$\text{MMCS}(W, \{\vec{U}_c \vec{V}_c^\top\}) = \frac{1}{m_2} \sum_{j=1}^{m_2} \max_c \left(\frac{\vec{U}_{:,c} V_{c,j} \cdot W_{:,j}}{||\vec{U}_{:,c} V_{c,j}||_2 ||W_{:,j}||_2} \right), \tag{9}$$

where $c \in C$ are parameter component indices and $j \in \{1, \cdots, m_2\}$ are input feature indices. A value of 1 for MMCS indicates that, for all input feature directions in the target model, there exists a parameter subcomponent whose corresponding column points in the same direction.

We also quantify how close their magnitudes are with the mean L2 Ratio (ML2R) between the Euclidean norm of the columns of W and the Euclidean norm of the columns of the parameter components with which they have the highest cosine similarity:

$$\text{ML2R}(W, \{\vec{U}_c \vec{V}_c^\top\}) = \frac{1}{m_2} \sum_{j=1}^{m_2} \frac{||\vec{U}_{:,\text{mcs}(j)} V_{\text{mcs}(j),j}||_2}{||W_{:,j}||_2}, \tag{10}$$

where $\text{mcs}(j)$ is the index of the subcomponent that has maximum cosine similarity with weight column j of the target model. A value close to 1 for the ML2R indicates that the magnitude of each parameter component is close to that of its corresponding target model column.

For both TMS_{5-2} and TMS_{40-10}, the MMCS and ML2R values are ≈ 1, with a closer match than that obtained for the decomposition in [Braun et al., 2025] (Table 1). This indicates that the parameter components are close representations of the target model geometrically. In contrast, APD exhibited significant shrinkage of the learned parameter components, with an ML2R of approximately 0.9 for both models [Braun et al., 2025], reminiscent of feature shrinkage in SAEs [Jermyn et al., 2024, Wright and Sharkey, 2024].

3.2 Toy Model of Superposition with hidden identity

SDL methods are known to suffer from the phenomenon of 'feature splitting', where the features that are learned depend on the dictionary size, with larger dictionaries finding more sparsely activating, finer-grained features than smaller dictionaries [Bricken et al., 2023, Chanin et al., 2024]: Suppose a network has a hidden layer that simply implements a linear map. When decomposing this layer with a transcoder, we can continually increase the number of latents and learn ever more sparsely activating, ever more fine-grained latents to better minimize its reconstruction and sparsity losses. This problem is particularly salient in the case of a linear map, but similar arguments apply to nonlinear maps.

By contrast, Braun et al. [2025] claimed that linear parameter decomposition methods do not suffer from feature splitting. In the linear case, SPD losses would be minimized by learning a single d-dimensional component that performs the linear map. The losses cannot be further reduced by

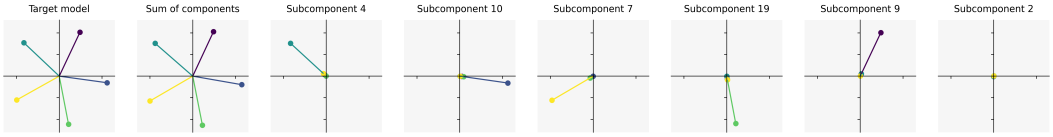

Figure 2: Plots of (left to right) the columns of the input weight matrix W of the $\text{TMS}_{5-2+\text{ID}}$ model (which can be written as $\hat{x} = \text{ReLU}(W^\top IWx + b)$, with a weight matrix $W \in \mathbb{R}^{m_1 \times m_2}$ and an identity matrix $I \in \mathbb{R}^{m_1 \times m_1}$); the sum of the parameter subcomponents for that matrix found by SPD; and the individual parameter subcomponents. Although this run of SPD used 20 subcomponents, only 6 subcomponents are shown, ordered by the sum of their matrix norms. The first five have learned one direction each, each corresponding to one of the columns of the target model. The final column and the other 14 components (not shown) have a negligible norm because they are superfluous for replicating the behavior of the target model.

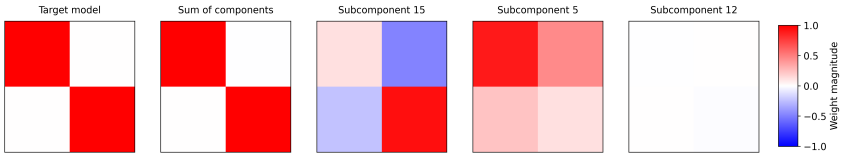

Figure 3: Plots of (left to right) the weights in the hidden identity matrix I of the $\text{TMS}_{5-2+\text{ID}}$; the sum of all subcomponents found by SPD for that matrix (including small-norm subcomponents that are not shown); and the largest three individual subcomponents. We see that SPD finds two subcomponents that together sum to the original rank-2 identity matrix of the target model, while the other subcomponents have a negligible weight norm.

adding more subcomponents, because that would prevent the components from summing to the original network weights.

Here, we empirically demonstrate this claim in a simple setting. We train a toy model of superposition identical to the TMS_{5-2} and TMS_{40-10}, but with identity matrices inserted between the down-projection and up-projection steps of the models. These models, denoted $\text{TMS}_{5-2+\text{ID}}$ and $\text{TMS}_{40-10+\text{ID}}$, can be written as $\hat{x} = \text{ReLU}(W^\top IWx + b)$, with a weight matrix $W \in \mathbb{R}^{m_1 \times m_2}$ and an identity matrix $I \in \mathbb{R}^{m_1 \times m_1}$.

We should expect SPD to find $m_2 + m_1$ subcomponents in total: m_2 subcomponents that correspond to the columns of the matrix W (each having causal importance for the model output if and only if one particular feature is present in the input, as in standard TMS (Section 3.1)) and m_1 subcomponents for the matrix I (almost all of which should have causal importance for the model output on every input). There are m_1 subcomponents because we need m_1 rank-one matrices to sum to a rank-m_1 matrix.

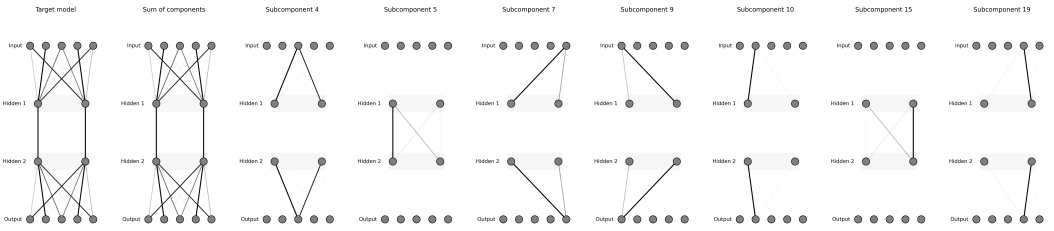

Figure 4: Plots of (left to right) the $\text{TMS}_{5-2+\text{ID}}$ networks parametrized by: The target model parameters; the sum of all parameter subcomponents found by SPD the decomposition of the model; and the seven individual subcomponents of non-negligible size. We see that SPD finds five subcomponents for the embedding matrix W, corresponding to the five input features, and two subcomponents that span the identity matrix I in the middle of the model.

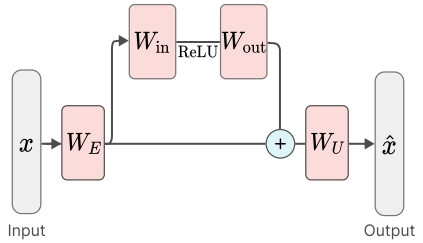

Figure 5: The architecture of the Toy Model of Compressed Computation. It uses a 1-layer residual MLP. Figure adapted from Braun et al. [2025].

SPD Results: Toy Model of Superposition with hidden identity

As expected, we find that SPD decomposes the embedding matrices W of both the $\text{TMS}_{5-2+\text{ID}}$ and $\text{TMS}_{40-10+\text{ID}}$ models into parameter subcomponents that closely correspond to their columns (Figure 3; Table 1).

Also as expected, SPD decomposes the identity matrix $I \in \mathbb{R}^{m_1 \times m_1}$ into only m_1 subcomponents that sum to the original matrix I (Figure 3). In total, SPD identifies only $m_2 + m_1$ subcomponents with non-negligible norm, thus identifying the ground truth mechanisms in each layer (Figure 4). APD fails to learn ground truth mechanisms in this model.

3.3 Toy Model of Compressed Computation

In this setting, the target network is a residual MLP that was previously studied by Braun et al. [2025] (Figure 5). It consists of a single residual MLP layer of width $d_{\text{mlp}} = 50$; a fixed, random embedding matrix with unit norm rows W_E; an unembedding matrix $W_U = W_E^\top$; and 100 input features, with a residual stream width of $d_{\text{resid}} = 1000$. This model is trained to approximate a function of sparsely activating input features $x_i \in [-1, 1]$, using a Mean Squared Error (MSE) loss between the model output and the labels. The labels we train the model to predict are produced by the function $y_i = x_i + \text{ReLU}(x_i)$. Crucially, the task involves learning to compute more ReLU functions than the network has neurons.

A naive solution to this task would be to dedicate each of the 50 neurons 'monosemantically' to computing one of the 100 input-output mappings. But this solution would perform poorly on the other 50 mappings. Instead, the model seems to achieve a better loss by using multiple neurons 'polysemantically' to compute a single input-output mapping. We thus expected that, for each mapping, a single subcomponent in the model's MLP input weight matrix W_{in} would be responsible for computing it, with every component connecting to many neurons[5].

Originally, we were unsure what the ground-truth mechanisms in the model's MLP output weight matrix W_{out} were. It seemed both possible that (a) for each input-output mapping, the model would have a single mechanism within W_{out} that projects the result of MLP activations back into the residual stream, or (b) that W_{out} was a single mechanism embedding the entire MLP output back into the residual stream, similar to the case of the identity matrix in the TMS model in Section 3.2. The analysis based on the APD decomposition in Braun et al. [2025] claimed the former, but our SPD decomposition, as shown below, clearly finds the latter.

SPD Results: Toy Model of Compressed Computation

To understand how each neuron participates in computing the output for a given input feature, Braun et al. [2025] measured the neuron's *contribution* to each input feature computation. The neuron contributions for input features $i \in \{0, \cdots, 99\}$ are calculated as:

$$\text{NeuronContribution}_{\text{model}}(i) = (W_{U[i,:]} W_{\text{out}}) \odot (W_{\text{in}} W_{E[:,i]}) \tag{11}$$

where $W_{E[:,i]}, W_{U[i,:]}$ are the i-th column of the embedding matrix and the i-th row of the unembedding matrix, and $\odot$ denotes element-wise multiplication. A large positive contribution indicates that the neuron plays an important role in computing the output for input feature i.

[5] See Bhagat et al. [2025] for further discussion on how the model may be performing this task.

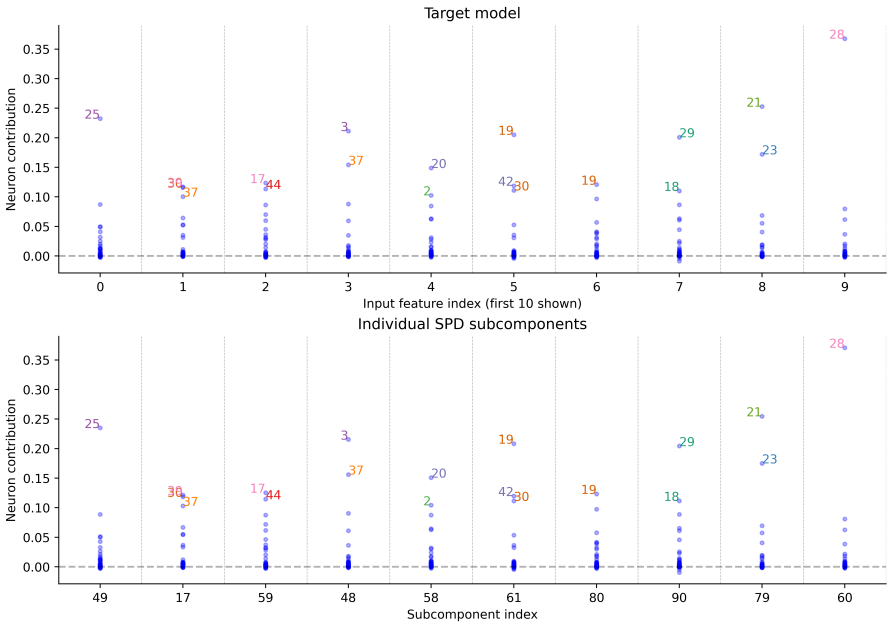

Figure 6: Toy Model of Compressed Computation: Similarity between target model weights and SPD subcomponents for the first 10 (out of 100) input feature dimensions. **Top**: Neuron contributions measured by Equation 11 for each input feature index $i \in \{0, \ldots, 9\}$. **Bottom**: Neuron contributions for the corresponding parameter subcomponents, measured by Equation 12 for each input feature index $i \in \{0, \ldots, 9\}$. The neurons are numbered from 0 to 49 based on their raw position in the MLP layer. An extended version of this figure showing all input features and parameter components can be found here.

Neuron contributions for individual subcomponents of W_{in} are calculated in a similar way:

$$\text{NeuronContribution}_{\text{subcomponent}}(i) = \max_c[(W_{U[i,:]}U_{\text{out}}V_{\text{out}}^\top) \odot (U_{\text{in},[:,c]}V_{\text{in},[c,:]}^\top W_{E[:,i]})] \tag{12}$$

We apply SPD to the MLP weight matrices $W_{\text{in}}, W_{\text{out}}$ in the target model. We find that SPD decomposes the W_{in} matrix into 100 subcomponents, each of which is involved in implementing $y_i = x_i + \text{ReLU}(x_i)$ for a unique input dimension $i \in \{0, \cdots, 99\}$. The computations required to produce the output for each input dimension in the target network (Figure 6 - top) are well replicated by individual parameter subcomponents in the SPD model (Figure 6 - bottom). For each input dimension, there is a corresponding parameter subcomponent of W_{in} that uses the same neurons to compute the function as the target model does. In other words, for each input dimension, the neuron contributions for each input dimension match the neuron contributions of some corresponding subcomponent. Figure 7 also summarizes all neuron contributions in the target model compared to individual subcomponents in the SPD model and shows they all match closely. Given a one-hot input x_i of magnitude 0.75 to the model, the masking functions in W_{in} select only their corresponding component to activate (Figure 8 - middle column).

Meanwhile, SPD splits its corresponding W_{out} into 50 subcomponents, which appear to all be effectively part of a single rank-50 component comprising the entire W_{out} matrix, since the masking functions seem to activate many components despite being a one-hot input $x_i = 0.75$ to the model. Both APD and SPD split the MLP input weight matrices W_{in} into one component per input feature [Braun et al., 2025]. However, APD also decomposed the MLP output weight matrix W_{out} into 100 components roughly corresponding to different input features, instead of a single rank-50 component. We believe that the SPD decomposition is more accurate, and that the APD decomposition of W_{out} was incorrect due to a poorly chosen value for the top-k hyperparameter. The SPD decomposition seems to reconstruct the computations of the original model more cleanly. In particular, Braun et al. [2025] note that some components partially represent secondary features (See Appendix C2 of Braun et al. [2025]), whereas SPD's masks do not have this issue. Additionally, the neuron contributions of the SPD decomposition are much closer to the target model and have less shrinkage (Compare neuron

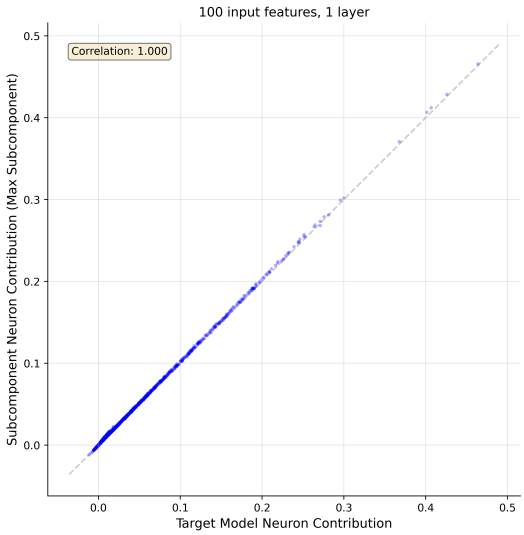

Figure 7: Toy Model of Compressed Computation: Similarity between target model weights and SPD subcomponents for all 100 input feature dimensions. **X-axis**: Neuron contributions measured by Equation 11 for each input feature index $i \in \{0, \dots, 99\}$. **Y-axis**: Neuron contributions for the corresponding parameter subcomponents of W_{in}, measured by Equation 12 for each feature index $i \in \{0, \dots, 99\}$. There is a very close match between the X and Y axis for each neuron contribution, indicating that each subcomponent connects its corresponding feature to the MLP neurons with almost the same weights as the target model.

contributions in Figure 6 of Braun et al. [2025] to our Figure 6). The SPD results also seem to better match theoretical predictions made by a mathematical framework for computation in superposition [unpublished work, forthcoming].

When the importance loss coefficient β_3 is too small, a single input feature activates a large number of components in W_{in} (Figure 8 - left two columns). This is suboptimal with respect to minimality, and is therefore a poor decomposition. When the importance loss coefficient β_3 is too high, the importance values decrease below 1 (Figure 8 - right two columns), which, aside from being a suboptimal decomposition, is also associated with high reconstruction losses.

3.4 Toy Models of Cross-Layer Distributed Representations

Realistic neural networks seem capable of implementing mechanisms distributed across more than one layer [Yun et al., 2021, Lindsay et al., 2024]. To study the ability of SPD to identify components that are spread over multiple layers, Braun et al. [2025] also studied a toy model trained on the same task with the same residual MLP architecture as the one in Section 3.3, but with the 50 neurons spread over two MLPs instead of one (Figure 9).

As in the Toy Model of Compressed Computation, the model learns to compute individual functions using multiple neurons. But here it learns to do so using neurons that are spread over two layers. SPD should find subcomponents in both W^1_{in} and W^2_{in} that are causally important for computing each function. And, for the same reasons as in the Toy Model of Compressed Computation, W^1_{out} and W^2_{out} should be decomposed into subcomponents that are in fact all part of one large parameter component, but in this model this one large component should span both layers.

We apply SPD on this model, as well as another model that spreads 51 neurons over three MLPs to compute functions of 102 input features. APD struggled to decompose a model with more than two layers due to hyperparameter sensitivity, but SPD succeeds.

SPD Results for two- and three-layer models of Cross-Layer Distributed Representations

SPD finds qualitatively similar results to the 1-layer Toy Model of Compressed Computation presented in Section 3.3. In the two-layer model, the MLP input matrices $W^1_{\text{in}}, W^2_{\text{in}}$, which each have shape

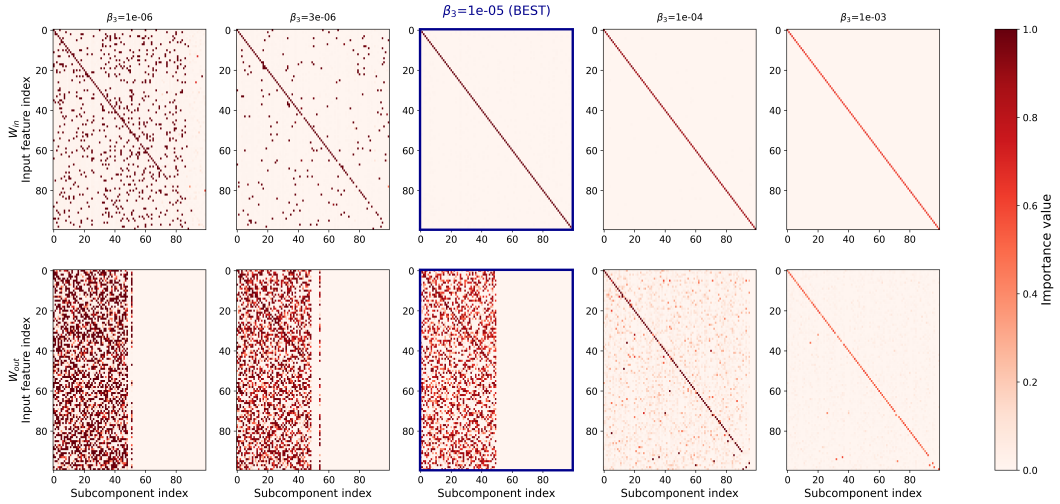

Figure 8: Causal importance values of each subcomponent (clipped between 0 and 1) in response to one-hot inputs ($x_i = 0.75$) for multiple importance loss coefficients β_3. The columns of each matrix are permuted differently, ordered by iteratively choosing the subcomponent (without replacement) with the highest causal importance for each input feature. When the importance loss coefficient β_3 is too low (Left), subcomponents in W_{in} are not 'monosemantic' (i.e. multiple subcomponents have causal importance for the same feature). When the importance loss coefficient β_3 is just right (middle column), each subcomponent in W_{in} has causal importance for computing a unique input feature. It also identifies the correct number of subcomponents in W_{out} (50). Although it identifies the correct number of components, these subcomponents need not align with any particular basis, and hence look 'noisy' because they align with multiple features. But this does not matter, since they always co-activate together and sum to the target model's identity-matrix parameters. When the importance loss coefficient β_3 is too high (Right), the rank-50 W_{out} component is split into too many subcomponents—approximately one subcomponent for each feature, but where many of the subcomponents have small causal importance values for other features. Also, the causal importance values on the diagonal shrink far below 1.0, resulting in high $\mathcal{L}_{\text{stochastic-recon}}$ and $\mathcal{L}_{\text{stochastic-recon-layerwise}}$ losses.

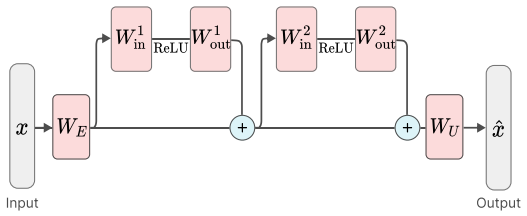

Figure 9: The architecture of one of our two Toy models of Cross-Layer Distributed representations. The other toy model has three MLP blocks instead of two. Figure adapted from Braun et al. [2025].

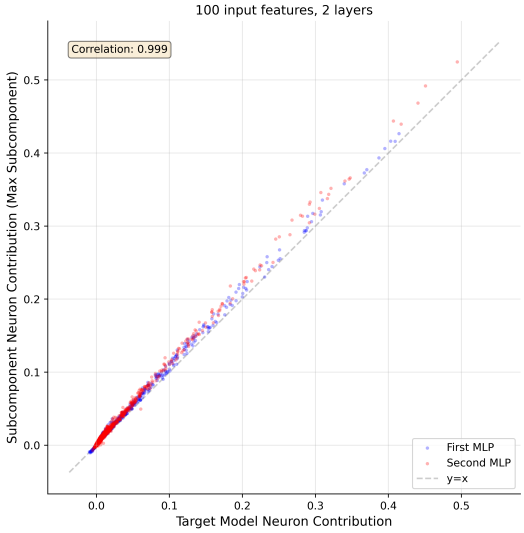

Figure 10: Toy Model of Distributed Representations (Two Layers): Similarity between target model weights and SPD subcomponents for all 100 input feature dimensions in a 2-layer residual MLP. Each point represents one neuron's contribution to a particular input feature. **X-axis**: Neuron contributions measured by Equation 11. **Y-axis**: Neuron contributions for the same neuron on the same input feature in the corresponding parameter subcomponents of $W^1_{\text{in}}, W^2_{\text{in}}$, measured by Equation 12. There is a close match between the X and Y axes for each neuron contribution, indicating that each subcomponent connects its corresponding feature to the MLP neurons with similar weights as the target model. However, there is a systematic skew toward higher values on the Y-axis, indicating that the neuron contributions of the subcomponents tend to be slightly larger. This is in contrast to the one-layer case (Figure 7) and three-layer case (Figure 11). We currently do not understand the source of this discrepancy, but it is possibly an outcome of suboptimal hyperparameters.

(d_mlp$=25$, d_embed$=1000$), are decomposed into 100 components that each compute the functions for one of the input features, using neurons spread over both MLP layers. The MLP output matrices $W^1_{\text{out}}, W^2_{\text{out}}$ are decomposed into a single rank 50 component. Notably, however, this component is now spread over both layers (Figure 12). This demonstrates that SPD can identify mechanisms that span multiple layers.

In the three-layer model, the MLP input matrices $W^1_{\text{in}}, W^2_{\text{in}}, W^3_{\text{in}}$, which each have shape (d_mlp $= 17$, d_embed $= 1000$), are likewise decomposed into 102 components all spread over 3 layers, and the MLP output matrices $W^1_{\text{out}}, W^2_{\text{out}}, W^3_{\text{out}}$ are decomposed into a single rank 51 component spread over all three layers (Figure 13). In both the two- and three-layer models, we find that the computations occurring in each parameter subcomponent of the W_{in} matrices closely correspond to individual input feature computations in the target model (Figures 10, 11, 14, 15). We note, however, that there is sensitivity to random seeds in these results for these models, possibly due to unfavorable random initialization, which we intend to explore in future work.

4 Related Work

SPD uses randomly sampled masks in order to identify the minimal set of parameter subcomponents that are causally important for computing a model's output on a given input. This is, in essence, a method for finding a good set of causal mediators [Mueller et al., 2024, Vig et al., 2020, Geiger et al., 2024a,b]: Our causal importance function can be thought of as learning how to causally intervene on our target model in order to identify a minimal set of simple causal mediators of a model's computation. However, unlike many approaches in the causal intervention literature (e.g. Chan et al. [2022], Wang et al. [2022], Conmy et al. [2023]), our approach does not assume a particular basis for these interventions. Instead, it learns the basis in which causal interventions are made. It also makes these interventions in parameter space.

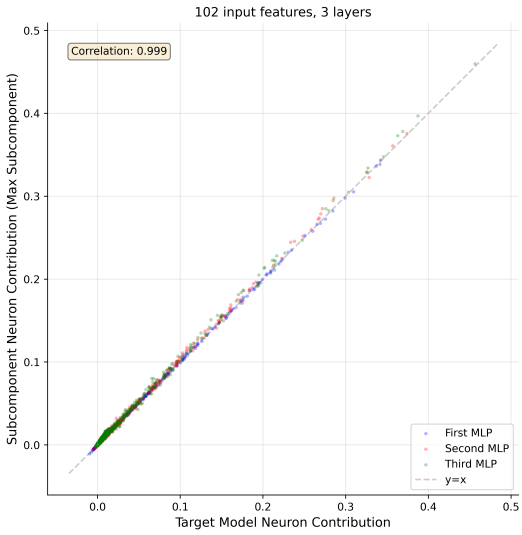

Figure 11: Toy Model of Distributed Representations (Three Layers): Similarity between target model weights and SPD subcomponents for all 102 input feature dimensions in a 3-layer residual MLP. Each point represents one neuron's contribution for a particular input feature. **X-axis**: Neuron contributions measured by Equation 11. **Y-axis**: Neuron contributions for the same neuron on the same input feature in the corresponding parameter subcomponents of $W^1_{\text{in}}, W^2_{\text{in}}, W^3_{\text{in}}$, measured by Equation 12. There is a close match between the X and Y axes for each neuron contribution, indicating that each subcomponent connects its corresponding feature to the MLP neurons with similar weights as the target model.

Some previous work learns fixed masks to ablate parts of the model's input or parts of its parameters [Csordás et al., 2021, Cao et al., 2021, Zhang et al., 2021]. But these masks tend to be for fixed inputs, and also often assume a particular basis. Our work also has an important difference to other masking-based attribution approaches: While our causal importances are predicted, the masks themselves are stochastically sampled with an amount of randomness based on the predicted causal importances.

Our definition of causal importance of subcomponents is related but not identical to the definition of causal dependence used in other literature [Lewis, 1973, Mueller, 2024]. For one, it may be possible that networks compute particular outputs even if one causally important subcomponent is ablated, using emergent self-repair [McGrath et al., 2023]. In that case, a subcomponent may be causally important for a particular output, but the output may not be causally dependent on it.

SPD also has parallels to work that uses attribution methods to approximate the causal importance of model components (e.g. Mozer and Smolensky [1988], Syed et al. [2024], Marks et al. [2024], Braun et al. [2025]) or the inputs (as in saliency maps) (e.g. Fong and Vedaldi [2017], Simonyan et al. [2014]). Our approach learns to predict causal importances given an input, which can be thought of as learning to predict attributions. To the best of our knowledge, we are not aware of similar approaches that learn to predict the input-dependent ablatability of model components using the method described in this paper.

Chrisman et al. [2025] decomposed networks in parameter space by finding low-rank parameter components that can reconstruct, using sparse coefficients, the gradient of a loss between the network output and a baseline output. Somewhat similarly, Matena and Raffel [2025] decomposed models in parameter space via non-negative factorisation of the models' per-sample Fisher Information matrices into components. SPD also decomposes networks into low-rank components in parameter space based on how the network output responds to perturbations. But instead of relying on local approximations like gradients to estimate the effects of a perturbation, it is trained by directly checking the effect ablating components in various combinations has on the output.

SPD can be viewed as approximately quantifying the degeneracy in neural network weights over different subdistributions of the data. SPD was in part inspired by singular learning theory [Watanabe,

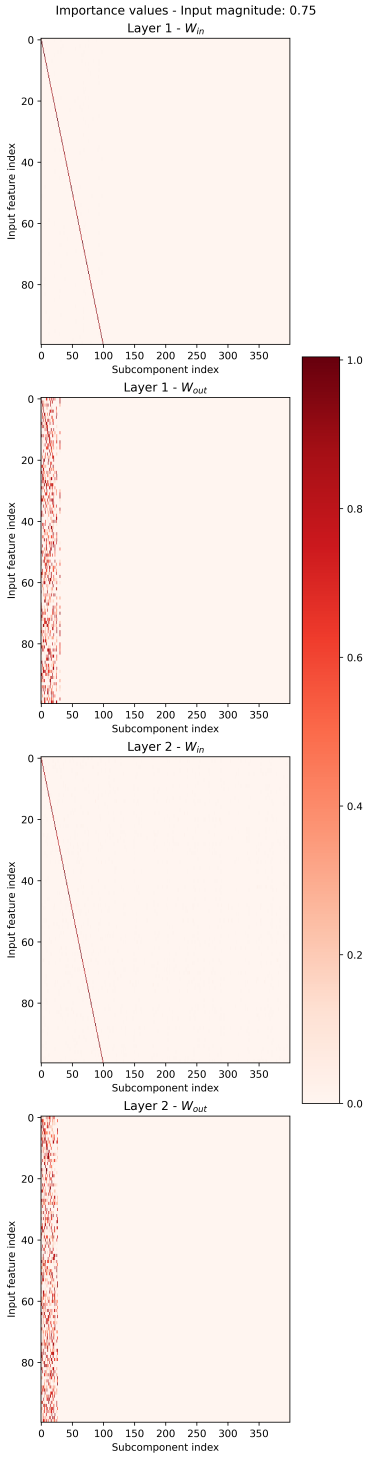

Figure 12: Toy Model of Distributed Representations (Two Layers): Causal importance values of each subcomponent (clipped between 0 and 1) for the matrices in both MLP layers, $W^1_{\text{in}}, W^1_{\text{out}}, W^2_{\text{in}}, W^2_{\text{out}}$ in response to one-hot inputs ($x_i = 0.75$). Each subcomponent in $W^1_{\text{in}}, W^2_{\text{in}}$ has causal importance for computing a unique input feature. On the other hand, the combined 50 subcomponents of $W^1_{\text{out}}, W^2_{\text{out}}$ all coactivate for all input features, indicating they are part of a single rank-50 identity component. The columns of each matrix are permuted differently, ordered by iteratively choosing the subcomponent (without replacement) with the highest causal importance for each input feature.

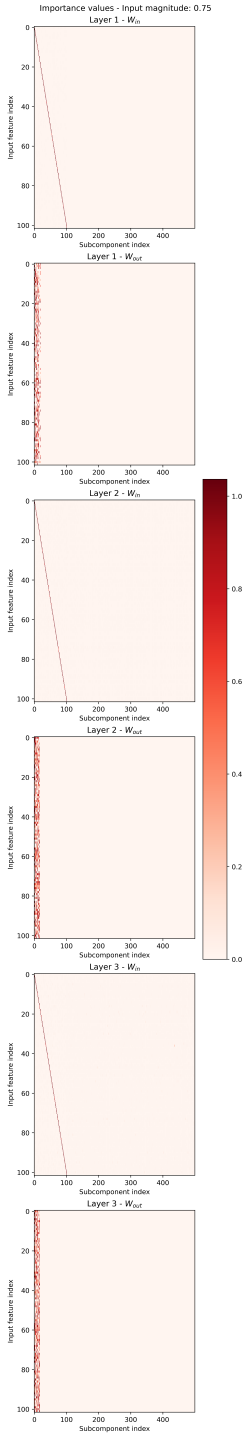

Figure 13: Toy Model of Distributed Representations (Three Layers): Causal importance values of each subcomponent (clipped between 0 and 1) for the matrices in all three MLP layers, $W^1_{\text{in}}, W^1_{\text{out}}, W^2_{\text{in}}, W^2_{\text{out}}, W^3_{\text{in}}, W^3_{\text{out}}$ in response to one-hot inputs ($x_i = 0.75$). Each subcomponent in $W^1_{\text{in}}, W^2_{\text{in}}, W^3_{\text{in}}$ has causal importance for computing a unique input feature. On the other hand, the combined 51 subcomponents of $W^1_{\text{out}}, W^2_{\text{out}}, W^3_{\text{out}}$ all coactivate for all input features, indicating they are part of a single rank-51 identity component. The columns of each matrix are permuted differently, ordered by iteratively choosing the subcomponent (without replacement) with the highest causal importance for each input feature.

2009], which quantifies degeneracy in network weights present over the entire data distribution using the learning coefficient. Wang et al. [2024] defined the data-refined learning coefficient, which measures degeneracy in neural network weights over a chosen subset of the distribution. In contrast, SPD works in an unsupervised manner, finding a single set of vectors to represent the neural network parameters, such that as many vectors as possible are degenerate on any given datapoint in the distribution. SPD also requires vectors to be ablatable to zero rather than just being degenerate locally.

See Braun et al. [2025] for further discussion on the relationship between linear parameter decomposition methods and sparse autoencoders; transcoders; weight masking and pruning; circuit discovery and causal mediation analysis; interpretability of neural network parameters; mixture of experts; and loss landscape intrinsic dimensionality and degeneracy.

5 Discussion

In this paper we introduced SPD, a method that resolves many of the issues of APD [Braun et al., 2025]. The method is considerably more scalable and robust to hyperparameters, which we demonstrate by using the method to decompose deeper and more complex models than APD has successfully decomposed.

We hypothesize that this relative robustness comes from various sources:

1. **APD required estimating the expected number of active components in advance**, because it needed to set the hyperparameter k for selecting the top-k most attributed components per batch. This number would usually not be known in advance for realistic models. APD results were very sensitive to it. SPD uses trained causal importance functions instead, and therefore no longer needs to use a fixed estimate for the number of active subcomponents per datapoint. We still need to pick the loss coefficient for the causal importance penalty β_3, but this is a much more forgiving hyperparameter than the hyper-sensitive top-k hyperparameter[6]
2. **Gradients flow through every subcomponent on every datapoint**, unlike in APD, where gradients only flowed through the top-k most attributed components. Top-k activation functions create discontinuities that, in general, tend to lead to unstable gradient-based training. In SPD, even subcomponents with causal importance values of zero will almost always permit gradients to flow, thus helping them error-correct if they are wrong.
3. **SPD does not need to optimize for 'simplicity'** (in the sense of Braun et al. [2025], where simple parameter components span as few ranks and layers as possible). Not only does this remove one hyperparameter (making tuning easier), but it also avoids inducing shrinkage in the singular values of the parameter component weight matrices. SPD does not exhibit shrinkage in the parameter subcomponents because the importance norm penalizes the probability that a subcomponent will be unmasked, but does not directly penalize norms of the singular values of the parameter matrices themselves. This can be helpful for learning correct solutions: For example, if correctly-oriented parameter components exhibit shrinkage, then their sum will not sum to the parameters of the target model, and therefore other parameter components will need to compensate. For this reason, the faithfulness and simplicity losses in APD were in tension. Removing this tension, and removing a whole hyperparameter to tune, makes it easier to hit small targets in parameter space. Although there is shrinkage in the causal importance values, this does not appear to be very influential since causal importance values only determine the allowed minimum value of the masks. For example, a causal importance value of 0.95 indicates that we can ablate a subcomponent by up to five percent without significantly affecting the network output, but we can also just not ablate it at all.
4. **APD used gradient-based attribution methods to estimate causal importance**, even though those methods are only first-order approximations of ideal causal attributions, which is often a poor approximation [Watson, 2022, Kramár et al., 2024]. If causal attributions

[6] In an idealized setting where (a) there is no batch noise in the loss, (b) no finite floating point precision, and (c) inactive mechanisms do not causally influence the network output at all, then setting β_3 to any value infinitesimally larger than zero should suffice to make our desired decomposition the global optimum.

are wrong, then the wrong parameter components would activate, so the wrong parameter components would be trained to compute the model's function on particular inputs. Systematic errors in causal attributions will lead to systematic biases in the gradients, which can be catastrophic when trying to hit a very particular target in parameter space. SPD instead directly optimizes for causal importance and is likely a much better estimate than the approximations found by even somewhat sophisticated attribution methods (e.g. Sundararajan et al. [2017]).

We hope that the stability and scalability of SPD will facilitate further scaling to much larger models than the ones studied here. In future work, we plan to test the scaling limits of the current method and explore any necessary adjustments for increased scalability and robustness.

We plan to investigate several outlying issues in future work. One issue is that we are unsure if learning independent subcomponents will enable the method to learn subcomponents that can describe the network's function using as short a description as possible. It may be possible that information from future layers is necessary to identify whether a given subcomponent is causally important. If it is, then calculating causal importance values layerwise will mean that some subcomponents are active when they need not be. It may therefore be interesting to explore causal importance functions that take as input more global information from throughout the network, rather than only the subcomponent inner activations at a given layer. Another issue is that both APD and SPD privilege mechanisms that span individual layers due to the importance loss or simplicity loss in SPD and APD respectively; it may be desirable to identify loss functions that privilege layers less.

The toy models in our work had known ground truth mechanisms, and therefore it was straightforward to identify which subcomponents should be grouped together into full parameter components. However, in the general case we will not know this by default. We therefore need to develop approaches that cluster subcomponents together in a way that combines the sparse and dense coding schemes laid out in the appendix of Braun et al. [2025] to achieve components that permit a minimum length description of the network's function in terms of parameter components.

It is worth noting that the SPD approach can be generalized in multiple straightforward ways. It is not necessary, for instance, to decompose parameter components strictly into subcomponents consisting of rank-one matrices. Subcomponents could, for instance, span only one rank but across all matrices in all layers. Alternatively, different implementations of the causal importance function could be used.

We expect that SPD's scalability and stability will enable new research directions previously inaccessible with APD, such as investigating mechanisms of memorization and their relationship to neural network parameter storage capacity. Additionally, the principles behind SPD may be useful for training intrinsically decomposed models.

Acknowledgments

We thank Tom McGrath, Stefan Heimersheim, Daniel Filan, Bart Bussmann, Logan Smith, Nathan Hu, Dashiell Stander, Brianna Chrisman, Kola Ayonrinde, and Atticus Geiger for helpful feedback on previous drafts of this paper. We also thank Stefan Heimersheim for initially suggesting the idea to train on an early version of what became the stochastic loss, which we had up to then only treated as a validation metric. Additionally, we thank Kaarel Hänni, whose inputs helped us develop the definition of component causal importance, and Linda Linsefors, whose corrections of earlier work on interference terms arising in computation in superposition helped us interpret the toy model of compressed computation.

References

Julius Adebayo, Justin Gilmer, Michael Muelly, Ian Goodfellow, Moritz Hardt, and Been Kim. Sanity checks for saliency maps. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper_files/paper/2018/file/294a8ed24b1ad22ec2e7efea049b8737-Paper.pdf.

Jai Bhagat, Sara Molas Medina, Giorgi Giglemiani, and Stefan Heimersheim. Compressed computation is (probably) not computation in superposition. https://www.lesswrong.com/posts/ZxFchCFJFcgysYsT9/compressed-computation-is-probably-not-computation-in, 2025.

Dan Braun, Lucius Bushnaq, Stefan Heimersheim, Jake Mendel, and Lee Sharkey. Interpretability in parameter space: Minimizing mechanistic description length with attribution-based parameter decomposition, 2025. URL https://arxiv.org/abs/2501.14926.

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. URL https://transformer-circuits.pub/2023/monosemantic-features/index.html.

Steven Cao, Victor Sanh, and Alexander M. Rush. Low-complexity probing via finding subnetworks, 2021. URL https://arxiv.org/abs/2104.03514.

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

David Chanin, James Wilken-Smith, Tomáš Dulka, Hardik Bhatnagar, and Joseph Bloom. A is for absorption: Studying feature splitting and absorption in sparse autoencoders, 2024. URL https://arxiv.org/abs/2409.14507.

Brianna Chrisman, Lucius Bushnaq, and Lee Sharkey. Identifying sparsely active circuits through local loss landscape decomposition, 2025. URL https://arxiv.org/abs/2504.00194.

Arthur Conmy, Augustine Mavor-Parker, Aengus Lynch, Stefan Heimersheim, and Adrià Garriga-Alonso. Towards automated circuit discovery for mechanistic interpretability. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 16318–16352. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/34e1dbe95d34d7ebaf99b9bcaeb5b2be-Paper-Conference.pdf.

Róbert Csordás, Sjoerd van Steenkiste, and Jürgen Schmidhuber. Are neural nets modular? inspecting functional modularity through differentiable weight masks. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=7uVcpu-gMD.

Hoagy Cunningham, Logan Riggs Smith, Aidan Ewart, Robert Huben, and Lee Sharkey. Sparse autoencoders find highly interpretable features in language models. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=F76bwRSLeK.

Nelson Elhage, Tristan Hume, Catherine Olsson, Nicholas Schiefer, Tom Henighan, Shauna Kravec, Zac Hatfield-Dodds, Robert Lasenby, Dawn Drain, Carol Chen, Roger Grosse, Sam McCandlish, Jared Kaplan, Dario Amodei, Martin Wattenberg, and Christopher Olah. Toy models of superposition, 2022.

Ruth C. Fong and Andrea Vedaldi. Interpretable explanations of black boxes by meaningful perturbation. In *2017 IEEE International Conference on Computer Vision (ICCV)*. IEEE, October 2017. doi: 10.1109/iccv.2017.371. URL http://dx.doi.org/10.1109/ICCV.2017.371.

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.

Atticus Geiger, Hanson Lu, Thomas Icard, and Christopher Potts. Causal abstractions of neural networks. In *Proceedings of the 35th International Conference on Neural Information Processing Systems*, NIPS '21, Red Hook, NY, USA, 2024a. Curran Associates Inc. ISBN 9781713845393.

Atticus Geiger, Zhengxuan Wu, Christopher Potts, Thomas Icard, and Noah Goodman. Finding alignments between interpretable causal variables and distributed neural representations. In Francesco Locatello and Vanessa Didelez, editors, *Proceedings of the Third Conference on Causal Learning and Reasoning*, volume 236 of *Proceedings of Machine Learning Research*, pages 160–187. PMLR, 01–03 Apr 2024b. URL https://proceedings.mlr.press/v236/geiger24a.html.

Kaarel Hänni, Jake Mendel, Dmitry Vaintrob, and Lawrence Chan. Mathematical models of computation in superposition, 2024. URL https://arxiv.org/abs/2408.05451.

Adam Jermyn, Adly Templeton, Joshua Batson, and Trenton Bricken. Tanh penalty in dictionary learning. https://transformer-circuits.pub/2024/feb-update/index.html#:~:text=handle%20dying%20neurons.-,Tanh%20Penalty%20in%20Dictionary%20Learning,-Adam%20Jermyn%2C%20Adly, 2024.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes, 2013. URL https://arxiv.org/abs/1312.6114.

János Kramár, Tom Lieberum, Rohin Shah, and Neel Nanda. Atp*: An efficient and scalable method for localizing llm behaviour to components, 2024. URL https://arxiv.org/abs/2403.00745.

Patrick Leask, Bart Bussmann, Michael Pearce, Joseph Bloom, Curt Tigges, Noura Al Moubayed, Lee Sharkey, and Neel Nanda. Sparse autoencoders do not find canonical units of analysis, 2025. URL https://arxiv.org/abs/2502.04878.

David K. Lewis. *Counterfactuals*. Blackwell, Malden, Mass., 1973.

Jack Lindsay, Adly Templeton, Jonathan Marcus, Thomas Conerly, Joshua Batson, and Christopher Olah. Sparse crosscoders for cross-layer features and model diffing, October 2024. URL https://transformer-circuits.pub/2024/crosscoders/index.html.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization, 2019. URL https://arxiv.org/abs/1711.05101.

Samuel Marks, Can Rager, Eric J. Michaud, Yonatan Belinkov, David Bau, and Aaron Mueller. Sparse feature circuits: Discovering and editing interpretable causal graphs in language models, 2024. URL https://arxiv.org/abs/2403.19647.

Michael Matena and Colin Raffel. Uncovering model processing strategies with non-negative per-example fisher factorization, 2025. URL https://arxiv.org/abs/2310.04649.

Thomas McGrath, Matthew Rahtz, Janos Kramar, Vladimir Mikulik, and Shane Legg. The hydra effect: Emergent self-repair in language model computations, 2023. URL https://arxiv.org/abs/2307.15771.

Jake Mendel. SAE feature geometry is outside the superposition hypothesis. *Alignment Forum*, 2024. URL https://www.alignmentforum.org/posts/MFBTjb2qf3ziWmzz6/sae-feature-geometry-is-outside-the-superposition-hypothesis.

Michael C Mozer and Paul Smolensky. Skeletonization: A technique for trimming the fat from a network via relevance assessment. In D. Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 1. Morgan-Kaufmann, 1988. URL https://proceedings.neurips.cc/paper_files/paper/1988/file/07e1cd7dca89a1678042477183b7ac3f-Paper.pdf.

Aaron Mueller. Missed causes and ambiguous effects: Counterfactuals pose challenges for interpreting neural networks, 2024. URL https://arxiv.org/abs/2407.04690.

Aaron Mueller, Jannik Brinkmann, Millicent Li, Samuel Marks, Koyena Pal, Nikhil Prakash, Can Rager, Aruna Sankaranarayanan, Arnab Sen Sharma, Jiuding Sun, Eric Todd, David Bau, and Yonatan Belinkov. The quest for the right mediator: A history, survey, and theoretical grounding of causal interpretability, 2024. URL https://arxiv.org/abs/2408.01416.

Lee Sharkey, Dan Braun, and Beren Millidge. Taking features out of superposition with sparse autoencoders, Dec 2022. URL https://www.alignmentforum.org/posts/z6QQJbtpkEAX3Aojj/interim-research-report-taking-features-out-of-superposition.

Lee Sharkey, Bilal Chughtai, Joshua Batson, Jack Lindsey, Jeff Wu, Lucius Bushnaq, Nicholas Goldowsky-Dill, Stefan Heimersheim, Alejandro Ortega, Joseph Bloom, Stella Biderman, Adria Garriga-Alonso, Arthur Conmy, Neel Nanda, Jessica Rumbelow, Martin Wattenberg, Nandi Schoots, Joseph Miller, Eric J. Michaud, Stephen Casper, Max Tegmark, William Saunders, David Bau, Eric Todd, Atticus Geiger, Mor Geva, Jesse Hoogland, Daniel Murfet, and Tom McGrath. Open problems in mechanistic interpretability, 2025. URL https://arxiv.org/abs/2501.16496.

Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps, 2014. URL https://arxiv.org/abs/1312.6034.

Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks, 2017.

Aaquib Syed, Can Rager, and Arthur Conmy. Attribution patching outperforms automated circuit discovery. In Yonatan Belinkov, Najoung Kim, Jaap Jumelet, Hosein Mohebbi, Aaron Mueller, and Hanjie Chen, editors, *Proceedings of the 7th BlackboxNLP Workshop: Analyzing and Interpreting Neural Networks for NLP*, pages 407–416, Miami, Florida, US, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.blackboxnlp-1.25. URL https://aclanthology.org/2024.blackboxnlp-1.25/.

Demian Till. Do sparse autoencoders find "true features"?, February 2024. URL https://www.lesswrong.com/posts/QoR8noAB3Mp2KBA4B/do-sparse-autoencoders-find-true-features.

Jesse Vig, Sebastian Gehrmann, Yonatan Belinkov, Sharon Qian, Daniel Nevo, Yaron Singer, and Stuart Shieber. Investigating gender bias in language models using causal mediation analysis. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 12388–12401. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/92650b2e92217715fe312e6fa7b90d82-Paper.pdf.

George Wang, Jesse Hoogland, Stan van Wingerden, Zach Furman, and Daniel Murfet. Differentiation and specialization of attention heads via the refined local learning coefficient, 2024. URL https://arxiv.org/abs/2410.02984.

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. *arXiv preprint arXiv:2211.00593*, 2022.

Sumio Watanabe. *Algebraic geometry and statistical learning theory*, volume 25. Cambridge university press, 2009.

David S. Watson. Conceptual challenges for interpretable machine learning. *Synthese*, 200:65, 2022. doi: 10.1007/s11229-022-03485-5. URL https://doi.org/10.1007/s11229-022-03485-5.

Benjamin Wright and Lee Sharkey. Addressing feature suppression in saes, Feb 2024. URL https://www.alignmentforum.org/posts/3JuSjTZyMzaSeTxKk/addressing-feature-suppression-in-saes.

Zeyu Yun, Yubei Chen, Bruno Olshausen, and Yann LeCun. Transformer visualization via dictionary learning: contextualized embedding as a linear superposition of transformer factors. In Eneko Agirre, Marianna Apidianaki, and Ivan Vulić, editors, *Proceedings of Deep Learning Inside Out (DeeLIO): The 2nd Workshop on Knowledge Extraction and Integration for Deep Learning Architectures*, pages 1–10, Online, June 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.deelio-1.1. URL https://aclanthology.org/2021.deelio-1.1/.

Dinghuai Zhang, Kartik Ahuja, Yilun Xu, Yisen Wang, and Aaron Courville. Can subnetwork structure be the key to out-of-distribution generalization?, 2021. URL https://arxiv.org/abs/2106.02890.

A Appendix

A.1 Architecture of Causal Importance Function MLPs

The MLPs γ_c^l that are used in the causal importance function consist of a single hidden layer of GELU neurons with width d_{gate}, followed by a hard sigmoid function:

$$\begin{aligned} g_c^l(x) &= \sigma_H\left(W_c^{l,\text{gate out}}\text{GELU}\left(W_c^{l,\text{gate in}}h_c^l(x) + b_c^{l,\text{gate in}}\right) + b_c^{l,\text{gate out}}\right) \\ \sigma_H(x) &:= \begin{cases} 0 & x \leq 0 \\ x & 0 \leq x \leq 1 \\ 1 & 1 \leq x \end{cases} \end{aligned} \tag{13}$$

Here, $h_c^l(x) := \sum_j V_{c,j}^l a_j^l(x)$ is the inner activation of the component and $W_c^{l,\text{gate in}} \in \mathbb{R}^{d_{\text{gate}} \times 1}, W^{l,\text{gate out}} \in \mathbb{R}^{1 \times d_{\text{gate}}}, b^{l,\text{gate in}} \in \mathbb{R}^{d_{\text{gate}}}, b^{l,\text{gate out}} \in \mathbb{R}$ are trainable parameters. Note that there is no sum over c in the above expression: Every subcomponent has its own separate causal importance MLP. This keeps the computational costs of training the gates low compared to the cost of training the subcomponents themselves.

It is important to note that this choice of causal importance function is only one of many possibilities. We chose it for its relative simplicity and low cost. In theory, SPD should be compatible with any method of predicting causal importance values $g_c^l(x)$ for the subcomponents. This is somewhat in contrast to sparse dictionary learning methods, where using arbitrarily expressive nonlinearities and optimization methods to determine dictionary activations raises concerns about whether a 'feature' is really represented by the network if it can only be identified in neural activations using a very complex nonlinear function versus a simple thresholded-linear function. See e.g. Bricken et al. [2023] for discussion.

A.2 Avoiding dead gradients with leaky hard sigmoids

The flat regions in a hard sigmoid function can lead to dead gradients for inputs below 0 or above 1. To avoid this, we use leaky hard sigmoids instead, which introduce a small non-zero slope below 0 or above 1. Specifically, we use *lower-leaky* hard sigmoids $\sigma_{H,\text{lower}}(x)$ with slope 0.01 below 0 for the gates used in the forward pass for the $\mathcal{L}_{\text{stochastic-recon}}$ and $\mathcal{L}_{\text{stochastic-recon-layerwise}}$ losses. And we use *upper-leaky* hard sigmoids $\sigma_{H,\text{upper}}(x)$ with slope 0.01 above 1 in the causal importance loss $\mathcal{L}_{\text{importance-minimality}}$:

$$\begin{aligned} \sigma_{H,\text{lower}}(x) &:= \begin{cases} 0.01x & x \leq 0 \\ x & 0 \leq x \leq 1 \\ 1 & 1 \leq x \end{cases} \\ \sigma_{H,\text{upper}}(x) &:= \begin{cases} 0 & x \leq 0 \\ x & 0 \leq x \leq 1 \\ 1 + 0.01(x-1) & 1 \leq x \end{cases} \end{aligned} \tag{14}$$

We use the lower leaky hard sigmoid for the forward pass because we should usually be able to scale a subcomponent that does not influence the output of the model below zero, but we cannot scale a subcomponent that does influence the output of the model above 1. So we can use masks smaller than zero in the forward pass, but not masks greater than one.

We use the upper leaky hard sigmoid in the importance loss because the causal importance values cannot be allowed to take negative values, else the training would not be incentivized to sparsify them. But there is no issue with allowing causal importance function outputs to be greater than 1.0 when computing the importance loss.

A.3 Heuristics for Hyperparameter Selection

Here we list some heuristics for how to select hyperparameters to balance the different loss terms of SPD in Equation 8. In particular, we focus on how to determine the appropriate trade-off between the stochastic reconstruction losses $\mathcal{L}_{\text{stochastic-recon}}$ and $\mathcal{L}_{\text{stochastic-recon-layerwise}}$ (controlled by β_1, β_2) and importance minimality loss $\mathcal{L}_{\text{importance-minimality}}$ (controlled by β_3).

- Negligible performance loss: The performance difference between the SPD model and the original model on the training dataset should be small enough to be mostly negligible. Quantitatively, one might want to judge how large the performance drop is based on LM scaling curves, as Gao et al. [2024] suggested for judging the quality of SAE reconstructions.
- Noise from superposition: Inactive mechanisms in superposition can still contribute to the model output through small interference terms [Hänni et al., 2024]. We can estimate the expected size of such terms, and ensure that $\mathcal{L}_{\text{stochastic-recon}}, \mathcal{L}_{\text{stochastic-recon-layerwise}}$ are no larger than what could plausibly be caused by such noise terms.
- Recovering known mechanisms: If some of the ground truth mechanisms in the target model are already known, we can restrict hyperparameters such that they recover those mechanisms. For example, in a language model, the embedding matrix mechanisms are usually known: Each vocabulary element should be assigned one mechanism. If none of a model's mechanisms are known to start with, we could insert known mechanisms into it. For example, one might insert an identity matrix at some layer in an LLM and check whether the SPD decomposition recovers it as a single high-rank component, as in the TMS-with-identity model in Section 3.2.
- Other sanity checks: The decomposition should pass other sanity checks. For example, a model parametrized by the sum of unmasked subcomponents should recover the performance of the target model. And at least some of the causal importance values should take values of 1 for most inputs; if they do not, then it is likely that the importance minimality loss coefficient β_3 is too high.

A.4 Training details and hyperparameters

A.4.1 Toy models of superposition (TMS)

Target model training All target models were trained for 10k steps using the AdamW optimizer [Loshchilov and Hutter, 2019] with weight decay 0.01 and constant learning rate 5×10^{-3}. The dataset uses input features sampled from $[0, 1]$ uniformly with probability 0.05, and 0 otherwise.

- TMS_{5-2} and $\text{TMS}_{5-2+\text{ID}}$: batch size 1024
- TMS_{40-10} and $\text{TMS}_{40-10+\text{ID}}$: batch size 8192

SPD training: Common hyperparameters

- Optimizer: Adam with max learning rate 1×10^{-3} and cosine learning rate schedule
- Training: 40k steps, batch size 4096
- Data distribution: same as target model (feature probability 0.05)
- Stochastic sampling: $S = 1$ for $\mathcal{L}_{\text{stochastic-recon}}$ and $\mathcal{L}_{\text{stochastic-recon-layerwise}}$
- Loss coefficients: $\mathcal{L}_{\text{faithfulness}} = 1$, $\mathcal{L}_{\text{stochastic-recon}} = 1$, $\mathcal{L}_{\text{stochastic-recon-layerwise}} = 1$
- Causal importance functions: One MLP per subcomponent, each with one hidden layer of $d_{\text{gate}} = 16$ GELU neurons.

SPD training: Model-specific hyperparameters

- TMS_{5-2} and $\text{TMS}_{5-2+\text{ID}}$: $\mathcal{L}_{\text{importance-minimality}}$ coefficient 3×10^{-3}, $p = 1$
- TMS_{40-10} and $\text{TMS}_{40-10+\text{ID}}$: $\mathcal{L}_{\text{importance-minimality}}$ coefficient 1×10^{-4}, $p = 2$ (we expect that one could obtain similar results in these toy models with many different settings for p)

A.4.2 Toy models of Compressed computation and Cross-Layer Distributed Representation

Model architectures

- 1-layer and 2-layer residual MLPs: 100 input features, embedding dimension 1000, 50 MLP neurons total (25 per layer for 2-layer)
- 3-layer residual MLP: 102 input features, embedding dimension 1000, 51 MLP neurons total (17 per layer)

Target model training All models trained using AdamW with weight decay 0.01, max learning rate 3×10^{-3} with cosine decay, batch size 2048. The dataset uses input features sampled from $[-1, 1]$ uniformly with probability 0.01, and 0 otherwise.

SPD training: Common hyperparameters

- Optimizer: Adam with constant learning rate
- Batch size: 2048
- Data distribution: same as target model (feature probability 0.01)
- Stochastic sampling: $S = 1$ for both stochastic losses
- Loss coefficients: $\mathcal{L}_{\text{stochastic-recon}} = 1$, $\mathcal{L}_{\text{stochastic-recon-layerwise}} = 1$
- $p = 2$ for $\mathcal{L}_{\text{importance-minimality}}$

SPD training: Model-specific hyperparameters

- 1-layer residual MLP: learning rate 2×10^{-3}, $\mathcal{L}_{\text{importance-minimality}}$ coefficient 1×10^{-5}, $C = 100$ initial subcomponents, 30k training steps, causal importance function with $d_{\text{gate}} = 16$ hidden neurons per subcomponent
- 2-layer residual MLP: learning rate 1×10^{-3}, $\mathcal{L}_{\text{importance-minimality}}$ coefficient 1×10^{-5}, $C = 400$ initial subcomponents, 50k training steps, causal importance function with $d_{\text{gate}} = 16$ hidden GELU neurons per subcomponent
- 3-layer residual MLP: learning rate 1×10^{-3}, $\mathcal{L}_{\text{importance-minimality}}$ coefficient 0.5×10^{-5}, $C = 500$ initial subcomponents, 200k training steps (converges around 70k), causal importance function with $d_{\text{gate}} = 128$ hidden GELU neurons per subcomponent

The hyperparameters can also be found in the WandB report here.

A.5 Supplementary figures

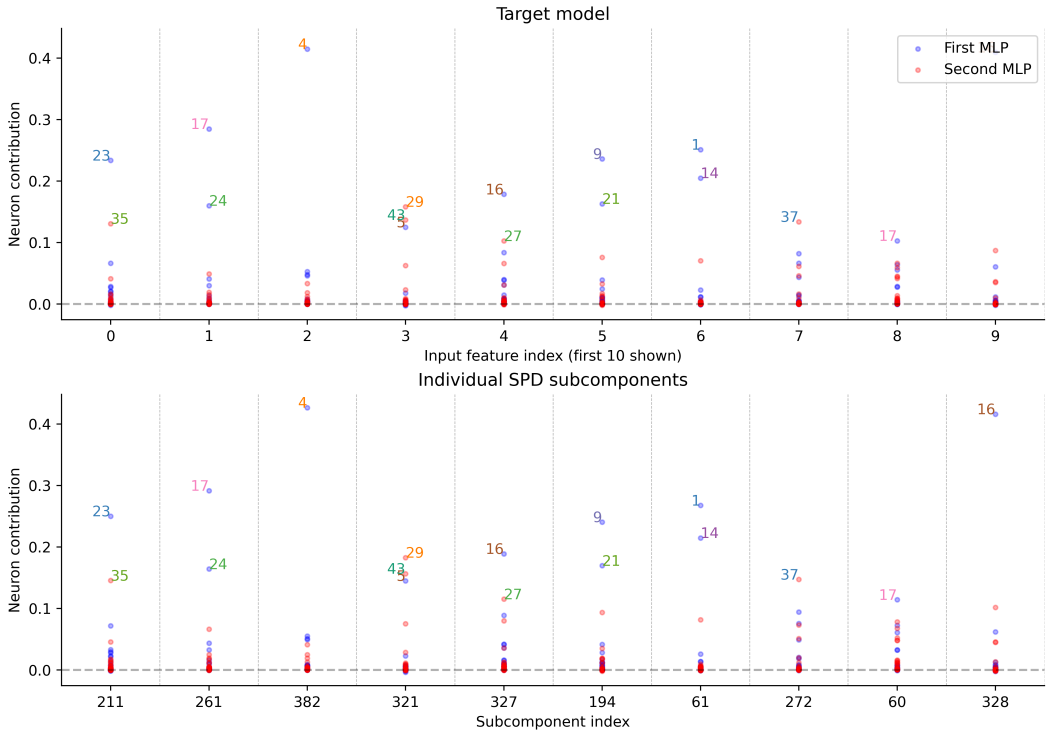

Figure 14: Toy Model of Distributed Representations (Two Layers): Similarity between target model weights and SPD model components for the first 10 input feature dimensions in a 2-layer residual MLP. **Top**: Neuron contributions measured by $(W_E W_{\text{IN}}) \odot (W_{\text{OUT}} W_U)$, where $\odot$ is an element-wise product and W_{IN} and W_{OUT} are the MLP input and output matrices in both layers concatenated together. **Bottom**: Neuron contributions for the learned parameter components, measured by $\max_m[(W_{U[i,:]} U^{\text{OUT}} V^{\text{OUT}}) \odot (U^{\text{IN}}_{[:,m]} V^{\text{IN}}_{[m,:]} W_{E[:,i]})]$ for each feature index $i \in [0, 9]$. The neurons are numbered based on their raw position in the network, with neurons 0 to 24 in the first layer and neurons 25 to 49 in the second layer.

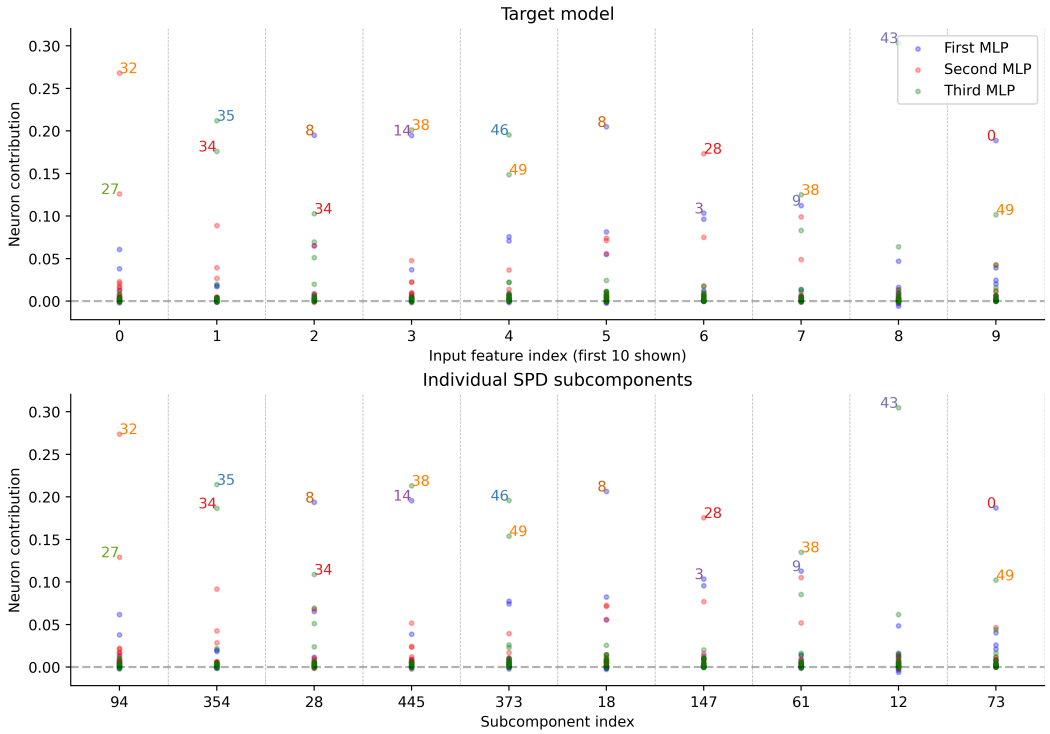

Figure 15: Toy Model of Distributed Representations (Three Layers): Similarity between target model weights and SPD model components for the first 10 input feature dimensions in a 3-layer residual MLP. **Top**: Neuron contributions measured by $(W_E W_{\text{IN}}) \odot (W_{\text{OUT}} W_U)$ where $\odot$ is an element-wise product and W_{IN} and W_{OUT} are the MLP input and output matrices in all three layers concatenated together. **Bottom**: Neuron contributions for the learned parameter components, measured by $\max_m[(W_{U[i,:]} U^{\text{OUT}} V^{\text{OUT}}) \odot (U^{\text{IN}}_{[:,m]} V^{\text{IN}}_{[m,:]} W_{E[:,i]})]$ for each feature index $i \in [0, 9]$. The neurons are numbered based on their raw position in the network, with neurons 0 to 16 in the first layer, 17 to 33 in the second layer and neurons 34 to 50 in the third layer.

A.6 SPD Pseudocode

Algorithm 1 Stochastic Parameter Decomposition (SPD)

Require: Target model $f(\cdot, W)$ with parameters $W = \{W^l\}_{l=1}^L$ to be decomposed.
Require: Dataset $\mathcal{D}$.
Require: C subcomponents per layer.
Require: Loss coefficients $\beta_1, \beta_2, \beta_3$.
Require: Causal Importance Minimality loss p-norm $p > 0$.
Require: Number of mask samples $S \geq 1$.
Ensure: Learned subcomponents $\{U^l, V^l\}_{l=1}^L$ and parameters for causal importance MLP $\{\Gamma^l\}_{l=1}^L$.

Initialize subcomponents $U^l \in \mathbb{R}^{d_{out} \times C}$, $V^l \in \mathbb{R}^{C \times d_{in}}$ for each layer l.
Initialize parameters for causal importance MLPs Γ^l.
Initialize an optimizer for all trainable parameters ($\{U^l, V^l, \Gamma^l\}_{l=1}^L$).
for each training step **do**
    Sample a data batch $X = \{x_1, \ldots, x_B\}$ from $\mathcal{D}$.
    Compute target model outputs Y_{target} and pre-weight activations $\{a^l\}_{l=1}^L$ for batch X.
    $\mathcal{L}_{\text{faithfulness}} \leftarrow \frac{1}{N} \sum_{l=1}^L \|W^l - U^l V^l\|_F^2$
    for each layer $l = 1, \ldots, L$ **do**
        $h^l \leftarrow V^l a^l$ *// Inner activations*
        $G^l_{\text{raw}} \leftarrow \Gamma^l(h^l)$ *// Raw causal importance MLP outputs*
    end for
    $\mathcal{L}_{\text{importance-minimality}} \leftarrow \frac{1}{B} \sum_{b=1}^B \sum_{l=1}^L \sum_{c=1}^C |\sigma_{H,\text{upper}}(G^l_{\text{raw},b,c})|^p$
    $\mathcal{L}_{\text{stochastic-recon}} \leftarrow 0$
    $\mathcal{L}_{\text{stochastic-recon-layerwise}} \leftarrow 0$
    Let $G^l = \sigma_{H,\text{lower}}(G^l_{\text{raw}})$ for all l.
    for $s = 1, \ldots, S$ **do**
        Sample $R^l_s \sim \mathcal{U}(0,1)^{B \times C}$ for each layer l.
        Compute masks $M^l_s \leftarrow G^l + (1 - G^l) \odot R^l_s$.
        Construct masked weights $\{W'^{(s)}_b\}_{b=1}^B$ with $W'^{(s),l}_b = U^l \cdot \text{Diag}(M^l_{s,b}) \cdot V^l$.
        $Y_{\text{masked}} \leftarrow f(X, W'^{(s)})$
        $\mathcal{L}_{\text{stochastic-recon}} \leftarrow \mathcal{L}_{\text{stochastic-recon}} + D(Y_{\text{masked}}, Y_{\text{target}})$
        for layer $l' = 1, \ldots, L$ **do**
            Construct layerwise masked weights $\{W'^{(s,l')}_b\}_{b=1}^B$ as follows:
                $W'^{(s,l'),l}_b = U^l \cdot \text{Diag}(M^l_{s,b} \text{ if } l = l' \text{ else } \mathbf{1}) \cdot V^l$
            $Y_{\text{masked-layerwise}} \leftarrow f(X, W'^{(s,l')})$
            $\mathcal{L}_{\text{stochastic-recon-layerwise}} \leftarrow \mathcal{L}_{\text{stochastic-recon-layerwise}} + D(Y_{\text{masked-layerwise}}, Y_{\text{target}})$
        end for
    end for
    Normalize $\mathcal{L}_{\text{stochastic-recon}} \leftarrow \mathcal{L}_{\text{stochastic-recon}}/S$
    Normalize $\mathcal{L}_{\text{stochastic-recon-layerwise}} \leftarrow \mathcal{L}_{\text{stochastic-recon-layerwise}}/(S \cdot L)$
    $\mathcal{L}_{\text{SPD}} \leftarrow \mathcal{L}_{\text{faithfulness}} + \beta_1 \mathcal{L}_{\text{stochastic-recon}} + \beta_2 \mathcal{L}_{\text{stochastic-recon-layerwise}} + \beta_3 \mathcal{L}_{\text{importance-minimality}}$
    Update parameters of $\{U^l, V^l, \Gamma^l\}_{l=1}^L$ using gradients of $\mathcal{L}_{\text{SPD}}$.
end for