Where SPD can favour GPT-2's neurons

Will Sackfield, Open Interpretability · https://open-interpretability.com/research/spd-input-projections

Abstract

A parameter decomposition keeps whichever basis it starts in on GPT-2 small's down projections, so it says little about neurons there. The input projection is a sharper test, because GELU acts on each neuron separately. We decomposed both MLP projections for 50,000 steps from each of two starts. At about the same total of active subcomponents, the start from the neurons ends with a KL of 0.093 against 0.158. A random start does not turn towards the neurons: its closest-neuron cosine stays at 0.07–0.10, near a random direction's 0.07. So the losses favour the neurons, and training does not find them.

Status: Preprint. Work left: #4104 (one count per site, more seeds and a constant-rate continuation).

1 Introduction

Much interpretability work on language models explains the neurons of an MLP layer one at a time. That choice assumes that the neuron basis is special: that the model's computation is sparse in the neurons, and not only in some rotation of them. Most tests of this assumption come from the reader's side. They ask whether a neuron gets an explanation that holds. A test from the weights would ask a different question: whether the model's parameters themselves prefer the neurons.

A parameter decomposition gives such a test. Stochastic Parameter Decomposition, or SPD (Bushnaq et al., 2025), splits each weight matrix into rank-one pieces that sum to the matrix. We call these pieces *subcomponents*. SPD trains them so that few of them are needed on each token. If a decomposition that starts from no basis found the neurons on its own, that would show that the neuron basis is special. If it found some other basis, the neurons would be one choice among many.

Earlier work leaves this test open for GPT-2's MLPs. An MLP layer has two weight matrices. The *input projection* reads the residual stream and writes the neurons' pre-activations. The *output projection*, which we call the *down projection*, reads the neurons' activations and writes back to the residual stream. Our earlier runs decomposed only the down projections (the baseline write-up). There, a decomposition keeps whichever basis it starts in. Any rotation of the neurons sums to the down projection exactly, and nothing in its losses breaks that symmetry. So that test says nothing about the neurons.

The input projection is a sharper test, because GELU (Hendrycks & Gimpel, 2016) acts on each neuron separately. A neuron that is off on a token can then be dropped at no cost. A subcomponent that writes to many neurons cannot. We therefore decomposed both projections of GPT-2 small (Radford et al., 2019), from two starts, and asked two things. Do a parameter decomposition's losses favour GPT-2's neurons, and does training find them from a random start?

The question matters for audits of single neurons. We study each neuron one sign at a time, and call one sign of a neuron a *pole*. An explanation of a pole is *VERIFIED* when it passes every gate of an adversarial review, in which a second model writes probes meant to break it. Such audits assume that a neuron is a unit of the model, and not an arbitrary axis. The question bears on research question 3, *Do native neurons explain a model as well as learned dictionaries do?* (research questions).

The paper's contributions are these:

- Two 50,000-step SPD runs of both MLP projections in all 12 layers of GPT-2 small, one from random directions and one from the neurons, with one seed per start.
- A whole-model comparison at about the same total of active subcomponents. The start from the neurons ends with a KL of 0.093, against 0.158 from random.
- A comparison per site, through a shared neuron yardstick and a neuron baseline that picks each token's best neurons on an input projection.
- A measure of alignment, which shows that a random start does not turn towards the neurons, although its subcomponents move.
- A second reading of the earlier runs on the down projections alone. The objective favoured the neurons there too, by less.

2 Related work

Parameter decomposition. SPD (Bushnaq et al., 2025) follows attribution-based parameter decomposition, APD (Braun et al., 2025). It splits each matrix into rank-one subcomponents, trained to sum to the matrix. A causal importance says how far each subcomponent can be ablated on a token. Its losses ask that the subcomponents sum to the matrix, that the output stays close to the model's when subcomponents are masked at random by their importance, and that few subcomponents are important. The source tests SPD on toy models. Applying it to GPT-2, with the changes in the Method, is our step.

Privileged bases. Elhage et al. (2022) show in toy models that a network can store more features than it has dimensions, which they call superposition. In their models, superposition gives polysemantic neurons. They call a basis privileged when an activation function breaks the rotation symmetry of a layer. They also note that "having a privileged basis doesn't guarantee that features will be basis-aligned". Our test follows from their definition. The down projection's losses see no privileged basis, because its input is already the neurons' activations. The input projection's losses see one, because GELU follows it. The step from their toy models to SPD's objective on GPT-2 is ours.

The neuron basis as a unit. Opinions differ on whether neurons are good units. Elhage et al. (2022) write that in large language models "it actually seems rare for neurons to correspond to clean features". Marks et al. (2025) build circuits from learned features in place of what they call "polysemantic and difficult-to-interpret units" such as neurons. Recent work finds more structure in neurons than that view expects. Arora et al. (2026) find that circuits are sparse in the neuron basis, and they call neurons "as sparse a feature basis as SAEs". They do not compare the neuron basis with a random rotation of the same layer. Shabalin and Piotrowski (2026) find that many transcoder features are neuron features, whose encoder and decoder both align with one neuron. They report that the other features move less from where training started. When they train a transcoder from

a neuron start, the neuron-aligned features carry 95% of the activation mass, against 65% from a random start. Every model that they study uses a gated MLP, and GPT-2's MLP is GELU. Our study asks the same kind of question of a parameter decomposition: how much the result depends on the start, and whether the objective prefers the neurons.

3 Method

Sites. Each MLP layer of GPT-2 small computes, for an input x from the residual stream,

$$h = \mathrm{GELU}(xW_{\mathrm{in}} + b_{\mathrm{in}}), \qquad y = hW_{\mathrm{out}} + b_{\mathrm{out}}$$

Here W_{in} is the input projection, W_{out} is the down projection, and neuron i is coordinate i of h. Each of the two matrices in each of the 12 layers is a *site*, so there are 24 sites.

Subcomponents. SPD writes the matrix W of each site as a sum of C rank-one subcomponents,

$$W = \sum_{c=1}^{C} \mathbf{v}_c \mathbf{u}_c^{\top}$$

where $\mathbf{v}_c$ is the input direction of subcomponent c and $\mathbf{u}_c$ is its output direction. On a token with centred input $\tilde{x}$, subcomponent c writes $(\tilde{x} \cdot \mathbf{v}_c)\, \mathbf{u}_c$. The write of the mean input goes to the bias.

Causal importance and masks. A gate gives each subcomponent a causal importance $g_c \in [0, 1]$ on each token. The masked output of a site is

$$\widehat{y} = \sum_{c=1}^{C} m_c\, (\tilde{x} \cdot \mathbf{v}_c)\, \mathbf{u}_c$$

with a mask m_c. In the *stochastic* mask, $m_c = g_c + (1 - g_c)\, r_c$, with r_c drawn uniformly from $[0, 1]$. In the *CI-masked* output, $m_c = g_c$ exactly. A subcomponent is *active* on a token when $g_c > 0$. A subcomponent is *alive* when it is active on some held-out token.

Losses. SPD's losses ask for three things. The subcomponents sum to W, which is faithfulness. The model's output with stochastic masks stays close in KL to the model's own output. The importance loss keeps few subcomponents important. On one token it is

$$L_{\mathrm{imp}} = \sum_{c=1}^{C} (g_c + \epsilon)^p$$

with an exponent $p > 0$. The source fixes p for each model. Our runs lower it during training (Method, *Training*), which is our choice. The *objective* is the sum of a run's weighted losses.

Input projections. The output loss and the fit are measured after GELU. A neuron that is off on a token can then be dropped at no cost, and a subcomponent that writes to many neurons cannot. That is the reason to expect a preference. It is our argument, not the source's.

Starts. A run starts either from random directions or from the neurons. From the neurons, on an input projection, subcomponent i starts with $\mathbf{v}_i = W_{:,i}$ and $\mathbf{u}_i = \mathbf{e}_i$. On a down projection it starts with $\mathbf{v}_i = \mathbf{e}_i$ and $\mathbf{u}_i = W_{i,:}$. Here $\mathbf{e}_i$ is the unit vector of neuron i . A start from the neurons thus sums to W exactly at step 0.

Measures. The *fit* of a site is the explained variance (EV) of its CI-masked output, the share of the output's variance over tokens that the masked output keeps. The *KL* is that of the model's output with every mask at its causal importance, from the model's own output. The *loss gap left* is the share of the loss rise from ablating every decomposed projection that the CI-masked model keeps,

$$\text{loss gap left} = \frac{L_{\text{CI}} - L_{\text{model}}}{L_{\text{ablated}} - L_{\text{model}}}$$

where L is the language-model loss. For alignment, we read each subcomponent's direction in the neurons' space: $\mathbf{u}_c$ on an input projection and $\mathbf{v}_c$ on a down projection. Its *closest-neuron cosine* is

$$\max_i \left|\cos(\mathbf{u}_c, \mathbf{e}_i)\right|$$

on an input projection. On this measure a neuron scores 1 and a random direction about 0.07. The *cosine with start* is the cosine of the same direction with its value at the start.

Neuron baselines. On an input projection, each token keeps the k neurons that move furthest from their ablated value. For this squared error, that is the best choice of k neurons. On a down projection, the ranking is $|h_i - \bar{h}_i| \, \|W_{i,:}\|$, with $\bar{h}_i$ the mean of neuron i. That ranking ignores cancellation, so that baseline is easier to beat.

Yardstick. *Neurons per active* lets two runs at different counts share one scale. Let k^* be the neurons per token that reach a run's fit, interpolated on the baseline's curve, and let a be the run's active count. Then

$$\text{neurons per active} = \frac{k^*}{a}$$

Above 1, SPD beats the baseline. The yardstick assumes the curve is a fair exchange rate, which we did not test.

Setup.

- **Model.** GPT-2 small, with 12 layers (Radford et al., 2019, Table 2). Its released weights have 3,072 GELU neurons (Hendrycks & Gimpel, 2016) per MLP. Each layer has an input projection `c_fc` and a down projection `c_proj`, so there are 24 sites.
- **Decomposition.** SPD (Bushnaq et al., 2025), after APD (Braun et al., 2025). Each site has $C = 3,072$ subcomponents.
- **Our changes.** One gate per site reads the centred input. The importance loss adds $\epsilon = 10^{-3}$, and gate gradients are clipped to norm 1. Each site also has a loss on its CI-masked output. The baseline write-up gives the failed pilots behind each change.
- **Runs.** Run C draws each subcomponent at random, then runs a faithfulness warmup. Run D starts each one as a neuron. Runs A and B are the same pair on the down projections only.
- **Training.** 50,000 steps on a Colab A100, 3.6 hours each, with 8 pieces of 128 tokens per batch. The exponent p falls from 2 to 0.4 by step 5,000, and the learning rate follows a cosine to zero. The seed is 0.
- **Data.** Training uses `NeelNanda/pile-10k`, the first 10,000 documents of the Pile (Gao et al., 2020), without the 518 documents of the scan corpus. That gives 113,546 pieces, with 1,096 held out. Its card gives the licence as `bigscience-bloom-rail-1.0`.

- **Evaluation.** The per-site tables read 64 held-out pieces (8,128 tokens). The KL, the loss gap and the training figure read the run's own 8 held-out pieces. The objective is averaged over the run's last 11 evaluations.
- **Code.** `scripts/train_spd_colab.ipynb` trains the runs, `scripts/spd_up_neurons.py` gives the neuron baseline of the input projections, and `scripts/spd_input_results.py` writes the results. The Appendices give the commands.
- **Figures.** Figure 2 moves. Each input projection's dot follows its measured fit and active count from step 5,000 to the end of training, at every 2,000 steps.

4 Does the start from the neurons fit better at equal sparsity?

The first question is whether the start from the neurons fits the model better when both starts keep about the same number of subcomponents active. If the losses were indifferent to the basis, two runs at the same total sparsity would end at about the same KL. Table 1 gives the four runs, and Figure 1 shows the KL of each through training.

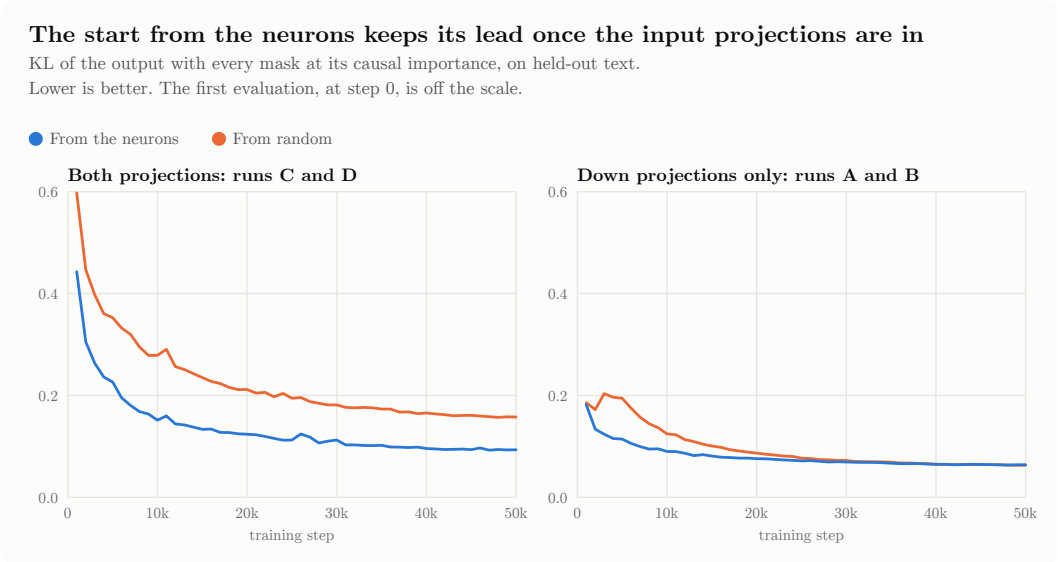

Figure 1. The KL of the model's output with every mask at its causal importance, on the run's held-out pieces, through 50,000 steps of training. The left panel is runs C and D, which decompose both projections. The right panel is runs A and B, which decompose the down projections alone. Lower is better.

Run	Sites	Start	Active	KL	Loss gap left	Objective (SD)	Importance term	Output term
A	down	random	6,807	0.063	0.012	3.32 (0.09)	2.57	0.66
B	down	neurons	4,538	0.064	0.010	2.90 (0.09)	1.61	1.19
C	both	random	5,152	0.158	0.030	6.06 (0.20)	2.27	3.51
D	both	neurons	5,186	0.093	0.011	4.40 (0.12)	2.20	2.03

Table 1. The whole-model fit of each run after 50,000 steps. *Active* is the total of active subcomponents per token over the run's sites. The objective is the mean of the last 11 evaluations, with its SD over those evaluations. Active counts for A and B come from

their last evaluation on 8 pieces. Those for C and D come from the 64 pieces of Tables 2 and 3. On 8 pieces C and D keep 5,119 and 5,159.

With both projections, the start from the neurons fits better at about equal total sparsity. Runs D and C keep 5,186 and 5,152 subcomponents active across all 24 sites, 0.7% apart. Run D's KL is 0.093 against 0.158, and its share of the loss gap left is 0.011 against 0.030 (Table 1). This is one seed per start, so the comparison has no interval across seeds.

The terms of the objective show where the lead comes from. With both projections, the importance terms are about level, and 1.48 of the objective's 1.66 gap is the output term. This is the like-for-like reading that the runs allow: the same price for sparsity, and a much better fit from the neurons. On the down projections the pattern differs. Run B's lead there is in sparsity, and its fit is lower.

The start from the neurons does not start ahead. At step 0 its KL is 5.34 against 4.56 from random. It leads by step 1,000 and stays ahead to the end (Figure 1). On the down projections alone, the two starts end level in KL, at 0.064 and 0.063.

The objective favoured the neurons without the input projections too, by less. From the neurons it is 27% lower with both projections (4.40 against 6.06), and 13% lower with the down projections alone (2.90 against 3.32). So we cannot credit all of the lead to GELU. The earlier reading of runs A and B, that SPD's losses do not choose a basis, holds for where each run ends in KL, but not for the value of the objective.

5 How does each start fit the input projections, site by site?

The whole-model result could hide a trade between sites. The second question is how each run spends its budget of active subcomponents across the two kinds of projection, and how each input projection fits for its count. Figure 2 follows each input projection through training. Tables 2 and 3 give each site at the end, and Figure 3 puts both runs on the shared yardstick.

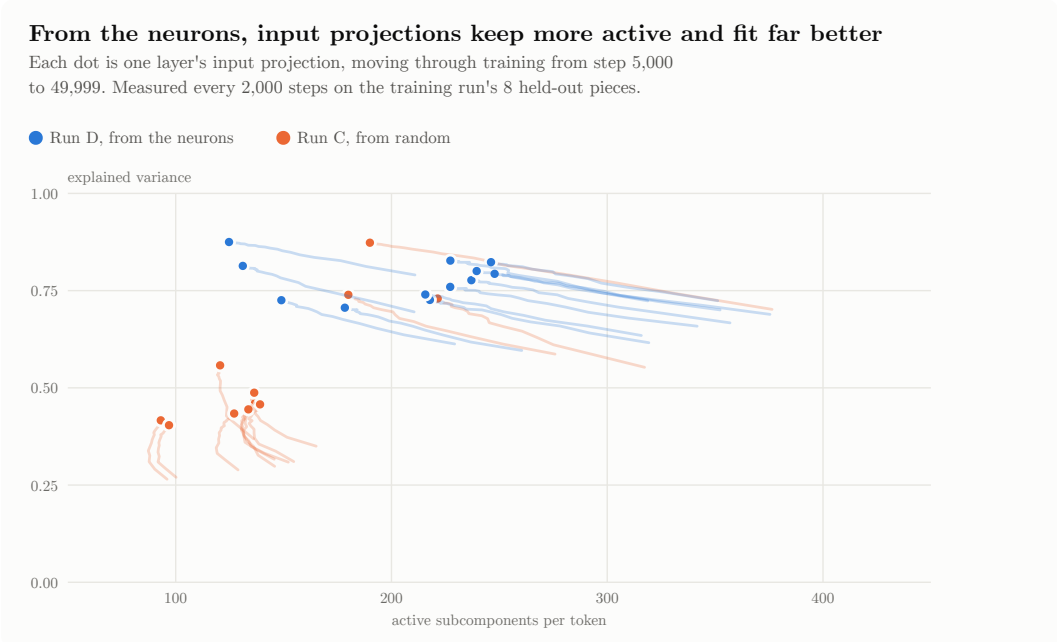

Figure 2. Each dot is one layer's input projection, moving through training from step 5,000 to the last step. The horizontal axis is the active subcomponents per token, and the vertical axis is the explained variance. Each faint line is the path of one dot. The dots rest on their last measured point.

The two runs spend their budget differently. Run D keeps more active on the input projections (2,456 against 1,706) and fewer on the down projections (2,730 against 3,445). So no input projection is compared at the same count. In Figure 2, each layer's input projection fits better in run D than in run C at every evaluation from step 5,000. In most of those evaluations it also keeps more subcomponents active.

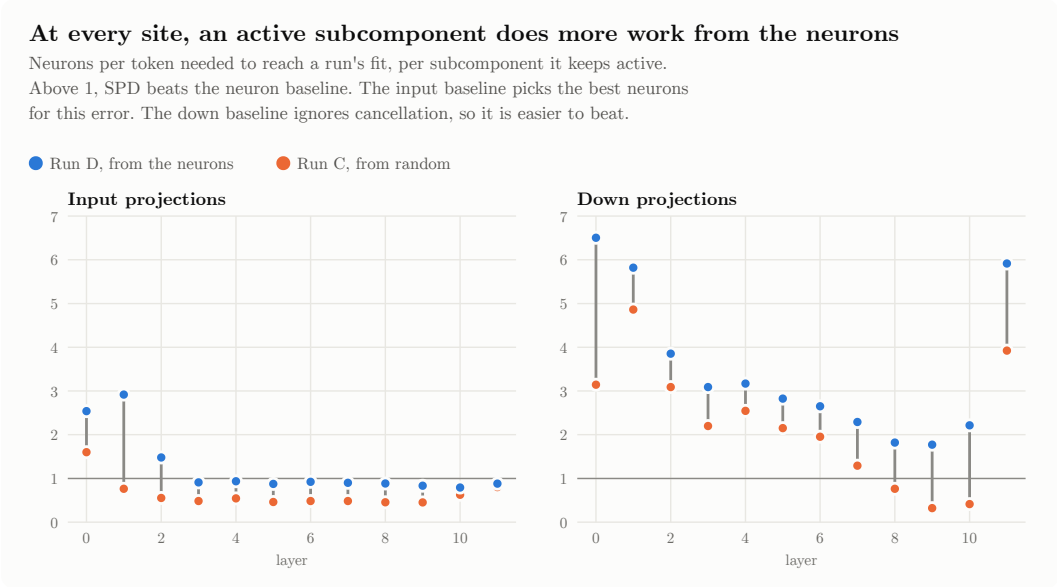

Figure 3. Neurons per active subcomponent for each site and run. It is the neurons per token that reach a run's fit on the baseline's curve, divided by the run's active count. Above 1, SPD beats the neuron baseline. The input baseline picks the best neurons for this error. The down baseline ignores cancellation, so it is easier to beat.

Layer	C: active	C: EV	D: active	D: EV	Best neurons, C's count	Best neurons, D's count	C: neurons per active	D: neurons per active
0	189	0.86	125	0.87	0.81	0.75	1.60	2.54
1	121	0.55	133	0.81	0.60	0.62	0.76	2.92
2	93	0.43	155	0.74	0.55	0.67	0.55	1.48
3	95	0.40	179	0.71	0.59	0.74	0.48	0.91
4	137	0.46	219	0.73	0.63	0.75	0.54	0.94
5	125	0.44	213	0.74	0.65	0.78	0.46	0.88
6	132	0.46	222	0.77	0.66	0.79	0.48	0.92
7	137	0.48	236	0.79	0.68	0.81	0.48	0.90
8	133	0.46	247	0.80	0.68	0.83	0.45	0.89
9	137	0.51	240	0.81	0.73	0.85	0.45	0.83
10	219	0.74	250	0.83	0.85	0.87	0.63	0.79
11	187	0.77	238	0.84	0.82	0.87	0.80	0.88

Table 2. The input projections of runs C and D, on 64 held-out pieces. *Active* is the active subcomponents per token, and *EV* is the fit after GELU. "Best neurons" is the fit of each token's best k neurons at that run's count.

Layer	C: active	C: EV	D: active	D: EV	Neurons, C's count	Neurons, D's count	C: neurons per active	D: neurons per active
0	200	0.91	107	0.92	0.77	0.68	3.14	6.50
1	230	0.77	184	0.75	0.21	0.16	4.86	5.82
2	274	0.77	217	0.77	0.15	0.07	3.09	3.85
3	291	0.76	221	0.78	0.31	0.14	2.20	3.09
4	339	0.77	264	0.76	0.09	-0.06	2.55	3.17
5	343	0.76	267	0.77	0.38	0.26	2.15	2.82
6	333	0.72	278	0.76	0.46	0.39	1.96	2.65
7	305	0.66	268	0.77	0.60	0.57	1.29	2.29
8	348	0.66	277	0.79	0.71	0.67	0.77	1.82
9	314	0.60	260	0.81	0.75	0.73	0.32	1.77
10	279	0.67	233	0.84	0.76	0.74	0.41	2.22
11	189	0.83	156	0.89	0.52	0.53	3.92	5.92

Table 3. The down projections of runs C and D, for comparison, on the same pieces. "Neurons" is the fit of each token's largest k neurons by the down projection's ranking, at that run's count.

On the shared neuron yardstick, run D is ahead at all 24 sites, by 1.10–5.51 times (Figure 3). At 10 sites run D has both the fewer active and the better fit, and run C has both at none. The yardstick is indirect, because it assumes that the baseline's curve is a fair exchange rate. On the down projections, run C's fit is higher in layers 1, 2 and 4, where it also keeps more active. Run D has the better fit and the lower count in the other nine layers (Table 3).

On most input projections, neither run beats the neurons themselves. At the same count, each token's best neurons fit better than run D in layers 3–11 and better than run C in layers 1–11 (Table 2). Run D beats its row's baseline only in layers 0–2, and run C only in layer 0. The soft gate gives SPD a lead on the down projections. But on most input projections it does not make up for the extra freedom of choosing neurons directly. On the down projections SPD is usually ahead, against a weaker baseline.

6 Does a random start turn towards the neurons?

The third question is whether training from random directions finds the neurons. If the losses favour the neurons, a random start might rotate towards them. Figure 4 and Table 4 give the closest-neuron cosine of each run's subcomponents after 50,000 steps.

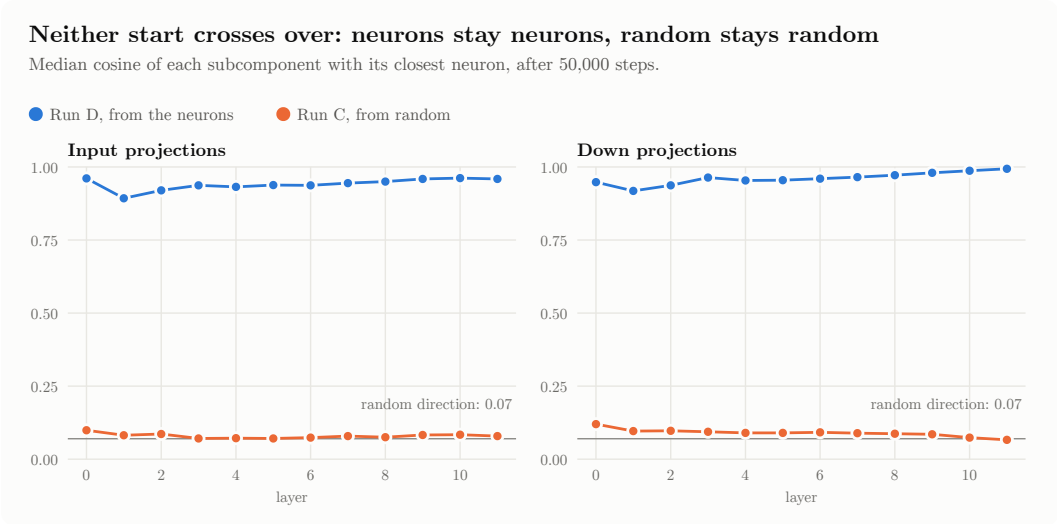

Figure 4. The median closest-neuron cosine of each layer's subcomponents after 50,000 steps, for the input projections and the down projections. The horizontal line is a random direction's 0.07.

Layer	C: cosine	C: with start	C: alive	D: cosine	D: over 0.7	D: alive
0	0.10	0.47	2,985	0.96	0.989	2,732
1	0.08	0.47	2,853	0.89	0.973	2,751
2	0.09	0.50	2,788	0.92	0.981	2,645
3	0.07	0.50	2,607	0.94	0.988	2,488
4	0.07	0.49	2,428	0.93	0.979	2,560
5	0.07	0.51	2,232	0.94	0.988	2,553
6	0.07	0.50	2,264	0.94	0.982	2,674
7	0.08	0.48	2,464	0.94	0.995	2,814
8	0.07	0.49	2,400	0.95	0.994	2,861
9	0.08	0.48	2,566	0.96	0.996	2,886
10	0.08	0.43	2,954	0.96	0.997	2,946
11	0.08	0.47	2,935	0.96	0.996	3,013

Table 4. The alignment of the input projections, reading each subcomponent's output direction $\mathbf{u}_c$. *Cosine* is the median closest-neuron cosine, *with start* is the median cosine with the start, *over 0.7* is the share of subcomponents with a closest-neuron cosine above 0.7, and *alive* counts the subcomponents of 3,072 that stay alive. Run D's cosine with its start equals its closest-neuron cosine, so the table gives it once.

A random start does not find the neurons. Run C's input projections end at a median closest-neuron cosine of 0.07–0.10, and none has a cosine above 0.7 (Table 4). On this measure a neuron scores 1 and a random direction about 0.07. So run C's subcomponents end near chance.

Run C's subcomponents did move. Their cosine with their start fell to 0.43–0.51. Run C's input directions moved further from their start (0.43–0.51) than its down directions did (0.55–0.57), and still ended near chance. So staying random here is not only a failure to move. It does not show that no path to the neurons exists.

Started as neurons, they stay neurons. Run D's input projections end at 0.89–0.96, with 97.3–99.7% of subcomponents above 0.7 (Figure 4). The losses therefore hold a decomposition at the neurons when it starts there, but they do not pull one towards the neurons from a random start.

7 Discussion

The three experiments fit one picture. SPD's losses favour GPT-2's neurons, and its training does not find them. At about the same total of active subcomponents, the start from the neurons has the lower KL, the lower loss gap and the lower objective. On the shared yardstick it is ahead at every site. Yet a random start, which moves well away from its own directions, ends no nearer the neurons than a random direction. The objective thus ranks the neuron basis above the random one, but gradient descent from random does not reach it in 50,000 steps.

The input projection was meant to be the place where a preference could appear, because GELU acts on each neuron separately. The result agrees with that argument only in part. The lead from the neurons is larger with the input projections, 27% against 13% in the objective. But a lead remains on the down projections alone, where any rotation of the neurons sums to the matrix. There, run B's lead is in sparsity, and its fit is lower. So part of the preference comes from something other than GELU, and we did not test what causes that part.

These results do not show that the neurons are the best basis for GPT-2, or the global optimum of SPD's objective. They do not show that a neuron is a good unit of explanation. A longer run or a higher learning rate could still narrow the lead from a random start. The neuron basis is also not a fixed point that the runs share. Run D's subcomponents moved, and end near the neuron basis but not exactly on it.

Three other explanations need to be ruled out. First, the lead may be a head start. Neurons sum to the matrix exactly at step 0, and a random start must first reach that. Against this, the start from the neurons does not start ahead in KL, and it leads by step 1,000. Second, the runs differ in where they spend their budget, so the per-site comparison rests on the yardstick. That yardstick is indirect, and only a comparison at one count per site would settle it. Third, our changes to SPD could favour one start. The method reached this fit only after them, so another setup could rank the starts differently.

The result agrees with the idea of a privileged basis from Elhage et al. (2022). An activation function that acts on each neuron breaks the rotation symmetry, and SPD's objective does prefer the neurons where GELU acts. It also illustrates their caution that a privileged basis does not guarantee alignment. Here a privileged basis in the objective did not give alignment from a random start. Shabalin and Piotrowski (2026) also find that the start matters. Their transcoders from a random start still put 65% of the activation mass on neuron-aligned features, whereas run C ends near chance. A transcoder and a parameter decomposition differ in what they train, and their models are gated, so the two results are not directly comparable.

The result also adds a check that Arora et al. (2026) did not run. They find circuits sparse in the neuron basis, but do not compare the neurons with a random rotation. Our runs make that comparison in the weights, for one objective: the random basis reaches a worse objective at the same total sparsity. An audit-based comparison on Qwen2.5-0.5B

reaches the same conclusion from the reader's side (native neurons against learned dictionaries). Against the view of Marks et al. (2025) that neurons are difficult units, our result says only that the objective prefers them, not that they are easy to explain. In our earlier runs, SPD started from the neurons treated VERIFIED neurons no differently from other studied neurons (the baseline write-up).

8 Limitations

- **One seed per start.** The objective's SD is over evaluations of one run, not over seeds. It cannot give a confidence interval on the gap between starts.
- **A head start is still possible.** Neurons sum to the matrix exactly at step 0, and a random start must first reach that. Over the last 10,000 steps run C's input fit rose by about 0.008 and run D's by about 0.004. Meanwhile the learning rate fell to zero. The KL gap stopped narrowing near step 30,000, but a falling learning rate could cause that too.
- **No input projection is compared at one count.** The active ranges overlap in only a few layers, and only between a late state of one run and an early state of the other. Thresholding the gates to one count would be a direct test, and it needs the checkpoints.
- **Run D's subcomponents did move.** They end at 0.89–0.96 with their start on the input projections, so run D is near the neuron basis, not exactly on it.
- **Dead subcomponents.** Between 2,232 and 3,013 of 3,072 stay alive on the input projections, depending on the site. The runs differ here too, and we did not test how much of the fit this explains.
- **With C equal to the hidden width, SPD cannot split a neuron.** A polysemantic neuron would need more subcomponents than neurons.
- **Our changes matter.** The gate, the centring and the output loss are ours, and the method reached this fit only after them. Another setup could rank the starts differently.

9 Conclusion

SPD's losses favour GPT-2's neurons, and training does not find them from a random start. With both MLP projections decomposed, at about the same total of active subcomponents, the start from the neurons ends with a KL of 0.093 against 0.158. A random start ends with a closest-neuron cosine of 0.07–0.10, near a random direction's 0.07, although its subcomponents moved. The objective favoured the neurons on the down projections alone too, by less, so GELU does not explain all of the lead. Four further studies would sharpen this answer. Thresholding both runs' gates to one count per site would compare the input projections directly. Repeating both starts with more seeds would give the gap an interval. Continuing run C at a constant learning rate would show whether the lead closes. Decomposing with more subcomponents than neurons would test the case that bears on polysemantic neurons.

References

Arora, A., Wu, Z., Steinhardt, J., & Schwettmann, S. (2026). *Language model circuits are sparse in the neuron basis.* ICML 2026. arXiv:2601.22594.

Braun, D., Bushnaq, L., Heimersheim, S., Mendel, J., & Sharkey, L. (2025). *Interpretability in parameter space: Minimizing mechanistic description length with attribution-based parameter decomposition*. arXiv:2501.14926.

Bushnaq, L., Braun, D., & Sharkey, L. (2025). *Stochastic parameter decomposition*. arXiv:2506.20790.

Elhage, N., Hume, T., Olsson, C., Schiefer, N., Henighan, T., Kravec, S., Hatfield-Dodds, Z., Lasenby, R., Drain, D., Chen, C., Grosse, R., McCandlish, S., Kaplan, J., Amodei, D., Wattenberg, M., & Olah, C. (2022). *Toy models of superposition*. Transformer Circuits Thread. arXiv:2209.10652.

Gao, L., Biderman, S., Black, S., Golding, L., Hoppe, T., Foster, C., Phang, J., He, H., Thite, A., Nabeshima, N., Presser, S., & Leahy, C. (2020). *The Pile: An 800GB dataset of diverse text for language modeling*. arXiv:2101.00027.

Hendrycks, D., & Gimpel, K. (2016). *Gaussian error linear units (GELUs)*. arXiv:1606.08415.

Marks, S., Rager, C., Michaud, E. J., Belinkov, Y., Bau, D., & Mueller, A. (2025). *Sparse feature circuits: Discovering and editing interpretable causal graphs in language models*. ICLR 2025. arXiv:2403.19647.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., & Sutskever, I. (2019). *Language models are unsupervised multitask learners*. OpenAI.

Shabalin, S., & Piotrowski, M. (2026). *Comparing sparsified MLPs and transcoders: Many transcoder features are neurons in disguise*. NeurIPS 2026 workshop on interpretability as a science. arXiv:2610.24521.

Appendices

Reproduce

Open `scripts/train_spd_colab.ipynb` on a Colab A100, with `corpus.jsonl` in `MyDrive/openinterpretability/`. Run it with `MATRICES = "both"` and `STEPS = 50_000`, once per start, with the default seed of 0. Each run writes `compare.json` and `train.log` to its Drive folder. Copy each run's `compare.json` into `docs/analyses/`, and each `train.log` to `exchange/spd-logs/` as `A.log` to `D.log`. The four logs of this paper's runs are published: `uv run python scripts/exchange.py fetch spd-logs` downloads them. Then:

```
uv run python scripts/spd_up_neurons.py --device cpu \
    --compare docs/analyses/spd-baseline-colab-both-random.json \
    --compare docs/analyses/spd-baseline-colab-both-neurons.json \
    --out docs/analyses/spd-baseline-up-neurons.json
uv run python scripts/spd_input_results.py --logs exchange/spd-logs \
    --out papers/spd-input-projections/results.jsonl
uv run python scripts/paper_figures.py
```

The checkpoints and subcomponents are in the owner's Google Drive, and are not published.