DEMYSTIFYING VARIANCE IN CIRCUIT DISCOVERY OF LARGE LANGUAGE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Circuit discovery is a key technique in mechanistic interpretability to pinpoint the model components that are crucial for performing a given task. Although the current state-of-the-art method (EAP-IG) performs well on the metric of (un)faithfulness, it suffers from substantial variability. This includes *resampling variance*, where the circuit changes when we probe with a new batch of data from the same distribution; *phrasing variance*, where the discovered circuit shifts when the prompts are rephrased; and *sample-wise variance*, where a circuit with low population unfaithfulness exhibits large fluctuations in unfaithfulness across individual samples. This paper studies the roots of these variances. We demonstrate that conductance-based EAP (CEAP), a more mathematically principled alternative to EAP-IG, lessens resampling variance. We further show that phrasing variance arises because prompts expressed in different templates tend to activate different circuits with limited cross-template generalizability, and that weight sparsity can mitigate this limitation. Regarding sample-wise variance, we find that extremely poor unfaithfulness scores often arise from how the metric is normalized, rather than from large discrepancies between the circuit and the full model's behavior.

1 INTRODUCTION

Circuit discovery aims to find the key pathways a model uses to perform a task. It typically scores model components by importance and selects the highest-scoring ones to form a circuit. In recent years, this area has advanced along two main axes: greater efficiency and broader generalizability.

On the efficiency side, methods are becoming more automated and easier to parallelize. To reduce manual work, Conmy et al. (2023) introduced an automated framework that iteratively tests the effects of removing model components. As such a method scales poorly, Marks et al. (2024); Syed et al. (2024); Nanda (2022) proposed gradient-based circuit discovery approaches that leverage the parallelism provided by modern deep learning hardware. In particular, edge attribution patching (EAP) (Syed et al., 2024; Nanda, 2022) can evaluate all the components of a model in two forward passes and one backward pass. However, this level of efficiency comes at the cost of attribution accuracy. To improve attribution accuracy while retaining computational efficiency, Hanna et al. (2024) averaged activation gradients along an interpolation path between corrupted and clean inputs, yielding EAP-IG, where IG stands for integrated gradients.

On the generalizability side, earlier work (Nanda, 2022; nostalgebraist, 2020) often focused on single-sample probing. However, findings from individual samples do not establish how broadly the identified mechanisms generalize across a task (Nanda, 2022). Later work assessed component importance across task datasets, typically by averaging scores over samples (Hanna et al., 2024; Wang et al., 2023; Hanna et al., 2023; Marks et al., 2024). By steering the discovered task circuits, Wang et al. (2025); Cheng et al. (2026) enhanced the model's performance on those tasks, highlighting the practical value of circuit discovery. Nevertheless, several studies have reported substantial variability in discovered circuits and their measured unfaithfulness, raising concerns about their reliability (Anani et al., 2026; Hadad et al., 2026; Palumbo et al., 2025).

In this work, we discuss three forms of such variance. The first is *resampling variance* (Méloux et al., 2025), where the identified circuit may change when prompts are resampled from the same distribution. We show that conductance-based EAP (CEAP), a more principled alternative to EAP-IG, reduces this variance and retains comparable computational efficiency.

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

The second type of variance is *phrasing variance* (Méloux et al., 2025), where alternative phrasings of the same input can produce substantially different circuits. We show that different prompt templates can activate distinct circuits with limited cross-template generalizability. This limits the usefulness of discovered circuits for understanding and controlling a model's behavior on a task beyond the templates used for discovery. We investigate weight sparsity as a potential remedy using the weight-sparse transformers trained by Gao et al. (2025), which are shown to have improved interpretability through more compact task circuits. We find that, as weight sparsity increases, circuits associated with different templates within the same task overlap more and generalize better across those templates.

The third form of variance is *sample-wise variance* (Miller et al., 2024), in which the circuit quality metric, unfaithfulness, varies sharply across samples. We show that extremely poor scores often arise from how the metric is normalized, rather than from large absolute discrepancies between the circuit and the full model's behavior. We connect this variation to a neural mechanism we term *selective contribution scaling*.

Taken together, our contributions are the following:

- We introduce CEAP, a more principled alternative to EAP-IG. We prove that conductance satisfies additive order preservation, recovering ground-truth importance orderings in a tractable setup, whereas IG can distort them. Empirically, CEAP reduces resampling variance.
- We show that template-dependent circuits have limited cross-template generalizability, hindering task-level understanding and control. We further demonstrate that increasing weight sparsity improves circuit overlap and generalizability across templates within the same task.
- We show that extreme sample-wise unfaithfulness often arises from normalization and provide a mechanistic explanation for this effect, offering a new perspective on interpreting unfaithfulness.

2 Background: Evaluating Subgraphs With (Un)faithfulness

Circuit discovery aims to identify a subgraph (circuit) from the model's full computational graph that best reproduces the full model's behavior (illustrated in Figure 8). A widely used measure of the circuit quality is faithfulness (Hanna et al., 2024). In the current setup, the *behavior* of a task is usually characterized by how a task-specific output metric changes when the input changes from a base point (referred to as corrupted input in previous literature) to the clean input. For instance, when examining subject-verb agreement (SVA) (Newman et al., 2021), we might quantify how much output probability shifts from plural verbs to singular verbs as the input changes from one that cues a plural verb (corrupted input) to one that cues a singular verb (clean input). To quantify this change in general, for a given input x, we use the metric `prob diff` $M(x; x_0, x_1) \coloneqq \mathbb{P}(C(x_1)|x) - \mathbb{P}(W(x_0)|x)$, where x_1 is a clean input, like *"The book on the table"*, and x_0 is a corrupted input like *"The books on the table"*. $C(x_1)$ is the set of tokens that x_1 encourages, and $W(x_0)$ is the set of competing tokens encouraged by x_0 instead. Continuing our example of SVA, $C(x_1)$ is singular verbs, and $W(x_0)$ is plural verbs. In the following, when the clean-corrupted pair is clear from context, we shorten $M(x; x_0, x_1)$ to $M(x)$. We discuss alternative metrics in previous literature in Appendix C.

We use subscripts G and G' to denote evaluation on the full computational graph and selected subgraph, respectively. The *behavior* of the full model and the selected circuit can be written as a subtraction of metrics,

$$Q_\Gamma(x_0, x_1) \coloneqq M_\Gamma(x_1) - M_\Gamma(x_0), \tag{1}$$

where $\Gamma \in \{G, G'\}$. The metrics for the full model in Equation (1) can each be obtained via two standard forward passes. By contrast, $M_{G'}(\cdot)$ is obtained by patching: edges outside G' are fixed to their activations under x_0, while edges inside G' remain responsive to the current input through the selected subgraph (Hanna et al., 2024), as visualized in Figure 8. By this design, $M_{G'}(x_0) = M_G(x_0)$.

Finally, the faithfulness (Hanna et al., 2024) of G' with respect to G is defined as $\Phi(G', G; x_0, x_1) = \frac{Q_{G'}(x_0,x_1)}{Q_G(x_0,x_1)}$. The best value for $\Phi(G', G; x_0, x_1)$ is 1, as it is the faithfulness yielded by choosing the entire graph. For the ease of discussion, in what follows, we use *unfaithfulness*, defined as $U(G', G; x_0, x_1) \coloneqq |1 - \Phi(G', G; x_0, x_1)|$, to evaluate the subgraph. The smaller this metric, the better, which clarifies the presentation.

3 CONDUCTANCE VERSUS INTEGRATED GRADIENTS

Circuit discovery begins by scoring the importance of a model's edges and selecting the most important ones to form a subgraph. Sundararajan et al. (2017) introduced integrated gradients (IG) to quantify the importance of input components. Dhamdhere et al. (2019) subsequently adapted IG to internal components by introducing conductance, matching the objective of circuit discovery. Nonetheless, EAP-IG (Hanna et al., 2024), the mainstream method for LLM circuit discovery, uses IG to score edges and achieves satisfactory results. More recently, Arora et al. (2026) noted that conductance is arguably the most principled scoring method for internal components. However, a direct comparison between IG and conductance is still missing, which we address below.

The metric M defined in Section 2 depends on the activations carried by the model's edges. Let E denote the set of all edges and $a_e(x) \in \mathbb{R}^{d_e}$ the activations on edge e for input $x \in \mathbb{R}^{d_{\text{in}}}$. We study how these activations contribute to the change in M as the input moves from x_0 to x_1. The importance we attribute to each component i of a_e is its conductance (Dhamdhere et al., 2019):

Definition 3.1. For a corrupted–clean input pair (x_0, x_1), the *conductance* of a scalar activation a_{ei} with respect to the metric M is defined as

$$I_{\text{Cond}}(a_{ei}) = \int_\gamma \frac{\partial M}{\partial a_{ei}} \mathrm{d}a_{ei} = \sum_{j \in [d_{\text{in}}]} (x_{1j} - x_{0j}) \int_0^1 \frac{\partial M(x_0 + \alpha(x_1 - x_0))}{\partial a_{ei}} \frac{\partial a_{ei}}{\partial x_j} \, \mathrm{d}\alpha, \tag{2}$$

where γ is the path that a_{ei} traverses when the input moves from x_0 to x_1 in a straight line. The conductance score of edge e is the sum over its activation components: $I_{\text{Cond}}(e) = \sum_{i \in [d_e]} I_{\text{Cond}}(a_{ei})$.

By contrast, EAP-IG (Hanna et al., 2024) scores each activation component using IG:

$$I_{\text{IG}}(a_{ei}) = [a_{ei}(x_1) - a_{ei}(x_0)] \int_0^1 \frac{\partial M(x_0 + \alpha(x_1 - x_0))}{\partial a_{ei}} \mathrm{d}\alpha. \tag{3}$$

The edge score is again the sum over activation components: $I_{\text{IG}}(e) = \sum_{i \in [d_e]} I_{\text{IG}}(a_{ei})$.

To compare conductance and IG in a principled way, we study a tractable class of functions with an additive branch structure, which we call sum-of-branches functions.

Definition 3.2. A sum-of-branches function $F(x; B)$ is a scalar function of the form $F(x; B) = \sum_{b \in B} f_b(y_b) = \sum_{b \in B} f_b(g_b(x))$, where B is the set of branches, and x and y_b may be vectors.

Consider moving the input x from x_0 to x_1. The resulting output change is $\Delta F(B) = F(x_1; B) - F(x_0; B)$, with each branch $b \in B$ contributing $\Delta f_b = f_b(g_b(x_1)) - f_b(g_b(x_0))$. By additivity, $\Delta F(B) = \sum_{b \in B} \Delta f_b$. We take $|\Delta f_b|$ as the ground-truth importance of branch b in this setting: it is exactly the magnitude of the change in $\Delta F(B)$ caused by patching the y_b to its corrupted-input value $y_b(x_0)$. We ask whether a scoring method, which assigns a scalar $I(y_b)$ to edge y_b, preserves the ordering of these magnitudes, a property we call *additive order preservation*:

Definition 3.3. A scoring method satisfies *additive order preservation* if, for every sum-of-branches function and every corrupted–clean input pair (x_0, x_1), its scalar edge scores satisfy

$$|\Delta f_{b_1}| > |\Delta f_{b_2}| \iff |I(y_{b_1})| > |I(y_{b_2})| \qquad \text{for all } b_1, b_2 \in B.$$

With the proof in Appendix E, we show:

Theorem 3.4. *Conductance satisfies additive order preservation, while IG does not.*

Remark 3.5. Beyond sum-of-branches functions, the notion of ground-truth importance is hard to establish because the effect of one component on the output might be influenced by whether other components are patched or not. The sum-of-branches structure decouples the effects of individual branches, exposing their ground-truth importance and enabling rigorous analysis. We regard additive order preservation as a minimal desideratum: a method intended to rank component contributions should recover their ordering at least in this analytically tractable setting.

The discretized formulas further clarify why IG is less principled. We elaborate in Appendix F and summarize the intuition here. In short, to numerically compute conductance and IG, we break down the straight-line movement of x into equally spaced discrete steps and evaluate gradients $\frac{\partial M}{\partial a_{ei}}$ at each

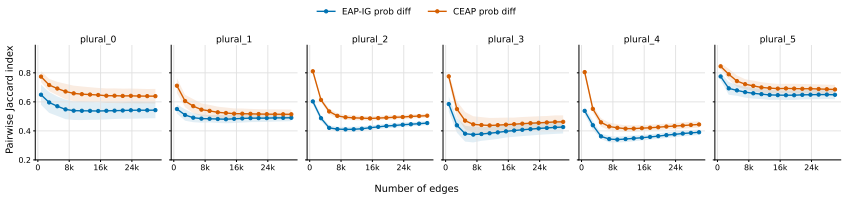


Figure 1: Comparison of PJI yielded by CEAP and EAP-IG (using GPT-2 XL for a subset of SVA)

step. Both scores are weighted sums of these gradients. Conductance weights each gradient by the corresponding activation change at that step, while IG uses a constant weight across all steps: the average activation change. Uneven activation changes due to the nonlinearity of $a_{ei}(x)$ therefore cause IG to mismatch gradients with activation movements, potentially distorting importance scores.

4 CEAP Reduces Circuit Variance Under Data Resampling

Méloux et al. (2025) noted that EAP-IG finds vastly different circuits when we resample the dataset from the same distribution. We show that replacing IG with conductance, yielding CEAP, mitigates this instability. We conducted experiments on GPT-2 small, GPT-2 XL, Pythia-160M, and Pythia-2.8B over three datasets: SVA (Newman et al., 2021), IOI (Wang et al., 2023), and greater-than (Hanna et al., 2023). The templates for generating these datasets are shown in Appendix D. Related setups are in Appendix B.

In our experiments, for each task, we used a full dataset of roughly 10000 samples and subsampled 4 smaller datasets (without replacement), each with 1000 samples. When scoring edges over a dataset, the edge scores are computed as the average over all samples. We performed circuit discovery for the 4 datasets and measured the stability of the found circuits using pairwise Jaccard index (PJI), defined in Appendix G. At each circuit size, the four resulting circuits yield six pairwise comparisons. We report the mean and standard deviation of these PJIs in Figure 1 for GPT-2 XL on a SVA subset. The full results are shown in Appendix H. The x-axis shows the circuit size. *Overall, we find that CEAP achieves PJI values that are higher than, or at least on par with, those of EAP-IG.*

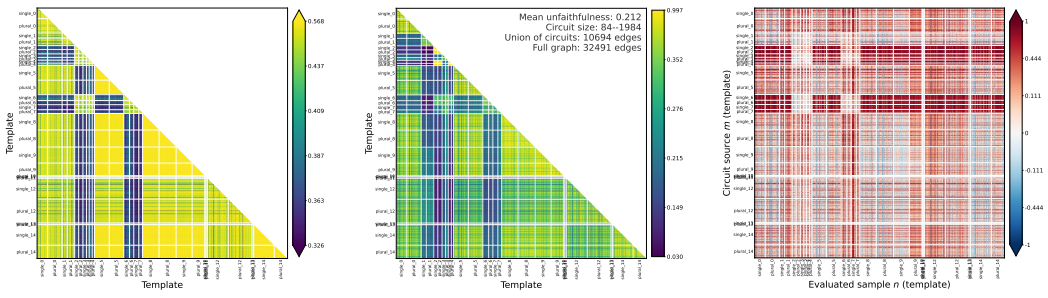
Although rigorously explaining CEAP's lower resampling variance in realistic models is challenging, we can establish this advantage for sum-of-branches functions (Appendix I). This result follows from conductance's exact recovery of ground-truth importance, whereas EAP-IG can distort importance rankings and select different circuits across resamples.

5 Template-induced Circuit Variance

Méloux et al. (2025) found that rephrasing the prompts used for circuit discovery can yield substantially different circuits, even when humans judge the prompts as representing the same task. We provide evidence that different templates might activate different circuits and, more importantly, that a circuit discovered for one template might yield deteriorated unfaithfulness on another within the same task. This challenges the assumption of task-level circuit generalizability: a circuit discovered using a finite set of templates may not faithfully explain or control the model's behavior on unfamiliar templates of the same task. We further show that weight sparsity, previously shown to improve model interpretability with more compact task circuits (Gao et al., 2025), mitigates this limitation.

5.1 Template Dependence of Circuit Hinders Generalizability

We take 1000 SVA samples generated from different templates and perform circuit discovery with CEAP for GPT-2 small on each sample *separately*. Each circuit thus contains edges that are the most important for the corresponding sample. To illustrate how circuits differ across templates and how well they generalize, Figure 2 compares the resulting edge-importance rankings, circuits, and cross-circuit unfaithfulness deterioration. In Figure 2a, we visualize pairwise ranking agreement using Kendall's τ_b, with edge importance defined as the absolute CEAP score. The heatmap exhibits a block structure aligned with template boundaries, showing that edge-importance rankings can



(a) Pairwise Kendall's τ_b. (b) Pairwise Jaccard index. (c) Cross-circuit $\Delta U_m(n)$.

Figure 2: Visualization of the template dependence of edge-importance rankings, circuits, and cross-circuit unfaithfulness deterioration for GPT-2 small on SVA. The heatmaps exhibit block structures that align well with template boundaries. Circuit size statistics are reported in (b).

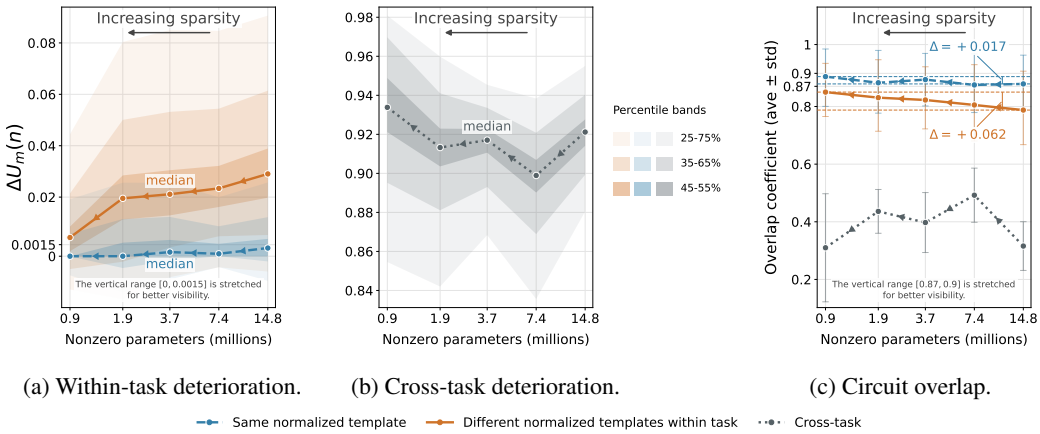
differ substantially across templates. For example, samples from templates with indices `2`, `3`, `4`, `6`, `7`, including both `single` and `plural` variants, have similar rankings among themselves but much lower ranking agreement with samples from the remaining templates. Circuit discovery then selects edges based on these rankings (Hanna et al., 2024). We visualize the PJIs of all the discovered circuits in Figure 2b, which also contains block patterns that signify the template-dependence of the circuits. Unlike Kendall's τ_b, which compares rankings over all edges, PJI ignores unimportant edges excluded from the circuits. This helps explain why Figure 2a and Figure 2b differ in detail while showing similar overall patterns.

Nonetheless, circuit overlap alone does not determine generalizability: shared edges may omit those crucial for another sample, and largely disjoint circuits may perform equally well on the same sample (Chen et al., 2026). We therefore evaluate generalizability directly through cross-circuit unfaithfulness deterioration over all ordered sample pairs (Figure 2c), defined as follows. Denote the full graph by G and the subgraph we obtained for sample m by G'_m. For a potentially different sample n with corrupted-clean pair (x_0^n, x_1^n), we evaluate the unfaithfulness of circuit G'_m on this sample as $U_m(n) \coloneqq U(G'_m, G; x_0^n, x_1^n)$. We measure $\Delta U_m(n) \coloneqq U_m(n) - U_n(n)$, which is the deterioration in unfaithfulness when replacing sample n's own circuit with sample m's circuit. A low $\Delta U_m(n)$ indicates that the circuit found for m generalizes well to sample n in steering its behavior, and a high $\Delta U_m(n)$ suggests otherwise. Figure 2c shows that the circuits found for templates with indices `2`, `3`, `4`, `6`, `7` generalize poorly to templates outside this group.

We produced these visualizations for three tasks (SVA, IOI, greater-than) on 2 models (GPT-2 small and Pythia-160M), shown in Appendix J. The same findings extend across the tested setups: samples from the same task may activate vastly different circuits, and a circuit found for one sample may perform poorly on another because their templates differ.

These observations challenge an important assumption underlying much prior work on circuit discovery: that a *single task circuit*, identified from representative samples, explains the model's behavior across different prompts expressing the same task (Hanna et al., 2024; Méloux et al., 2025; Miller et al., 2024; Nanda, 2022; Gao et al., 2025). This assumption also motivates steering task behavior by intervening on identified task circuits (Zhang et al., 2026; Wang et al., 2025; Cheng et al., 2026; Meng et al., 2022; 2023; Yao et al., 2024; Gao et al., 2025; Lindsey et al., 2025; Wang et al., 2023). Our results show that circuits discovered from one set of prompts may generalize poorly to other templates of the same task, raising concerns about their reliability for steering in the wild, where unseen prompt forms are unavoidable.

Remark 5.1. Complementary to our analysis, concurrent works address template-dependent circuits through a cluster-then-discover approach (Franco et al., 2026; Cho et al., 2026; Rai et al., 2026). They cluster prompts that appear to activate similar circuits and perform circuit discovery separately for each cluster. Although this can refine our understanding of how models handle the *observed* prompts, generalization beyond these prompts remains unresolved, as unseen prompts may not fit the identified clusters. This limits the approach's utility in the wild.



(a) Within-task deterioration. (b) Cross-task deterioration. (c) Circuit overlap.

Figure 3: Increasing weight sparsity improves circuit generalization and overlap across templates within the same task, while cross-task sample pairs show no corresponding trend. The horizontal axis denotes the number of non-zero weights in models with the same 110M-parameter architecture.

5.2 Weight Sparsity Boosts Cross-template Circuit Generalizability

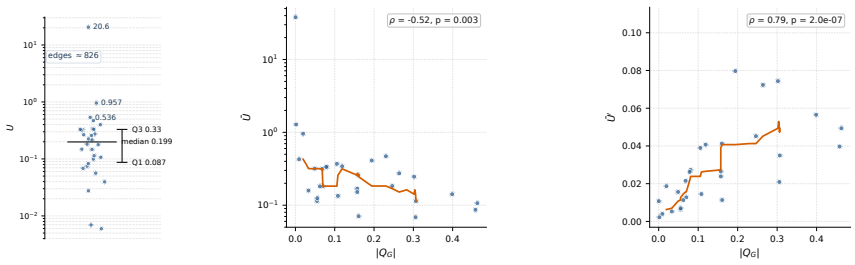
Gao et al. (2025) showed that imposing weight sparsity during training produces more compact task circuits. *Could it be that shrinking task circuits pushes template-specific circuits closer together, increasing their overlap and improving generalization across templates?* The following experiments suggest a positive answer.

As the weight-sparse transformers trained by Gao et al. (2025) specialize in coding, we construct three coding task datasets adapted from their released data: single-double-quote, else-elif, and with-open. Each dataset comprises 16 templates, and details are provided in Appendix D. We then perform CEAP-based circuit discovery separately on each sample. Although these models still exhibit template-dependent circuits (see Appendix K), we find that *increasing weight sparsity improves circuit generalization across templates within the same task.*

More specifically, we use 1000 samples from each dataset. For each template, we select the smallest swept circuit size at which the average unfaithfulness within each template falls below 0.09. Experiments for other unfaithfulness thresholds are in Appendix Q. We measure the cross-sample unfaithfulness deterioration $\Delta U_m(n)$ for each ordered sample pair. These pairs fall into three groups: 187,536 same-template pairs, 2,812,464 same-task cross-template pairs, and 6,000,000 cross-task pairs. In Figure 3a, we visualize how $\Delta U_m(n)$ changes with sparsity for the first two groups. The distribution of $\Delta U_m(n)$ over same-task cross-template pairs compresses toward 0 as sparsity increases. This provides evidence that sparsity improves cross-template generalizability. For same-template pairs, $\Delta U_m(n)$ remains close to 0 throughout, indicating good generalizability within templates. Its median decreases slightly as sparsity increases, reaching almost exactly 0 at the highest sparsity level.

Improved cross-template generalizability is accompanied, and potentially explained, by greater overlap between the corresponding circuits. In Figure 3c, we visualize the Szymkiewicz–Simpson overlap coefficient[1] for the three groups of sample pairs, defined as $\frac{|E_{G'_m} \cap E_{G'_n}|}{\min\left(|E_{G'_m}|, |E_{G'_n}|\right)}$, where $E_{G'_m}$ and $E_{G'_n}$ are the edge sets of the circuits found for samples m and n, respectively. The coefficient becomes 0 when the circuits share no edges and 1 when one circuit is contained in or equal to the other. Figure 3c shows that average circuit overlap increases with sparsity for same-task cross-template pairs. Same-template pairs exhibit a similar, though milder, increase. This smaller increase is expected, as same-template circuits already have high overlap even at the lowest sparsity level studied.

[1]Note that PJI is less reflective of circuitry sharing, as it penalizes differences in circuit size. One example is provided by the greater-than circuits for GPT-2 small and Pythia-160M, shown in Figures 35 and 38. Template 9's circuits have low PJI with those of other templates but nonetheless generalize well to them. Their low PJI reflects a size mismatch: template 9's circuits are larger and largely contain those of other templates, as indicated by the high overlap coefficients in Figures 35d and 38d.



(a) U distribution. (b) $\bar{U}$ vs. $|Q_G|$. (c) $\bar{U}'$ vs. $|Q_G|$.

Figure 4: Circuit discovery results in great unfaithfulness variance. Here, we perform circuit discovery for GPT-2 small on SVA template `plural_1` with target circuit size 826 for each sample separately. Samples with smaller $|Q_G|$ tend to show larger pessimistic unfaithfulness $\bar{U}$, while the pessimistic unnormalized unfaithfulness $\bar{U}'$ is typically small when $|Q_G|$ is small. Orange curves show rolling-median trends as visual guides only. Boxes report Spearman rank correlation ρ and its p-value.

Interestingly, cross-task sample pairs do not exhibit the improvements observed within tasks. Cross-task deterioration $\Delta U_m(n)$ remains high, and its median reaches the highest at the greatest sparsity, while average circuit overlap fluctuates and reaches its lowest value there (Figures 3b and 3c). Together, these observations suggest task-oriented circuit compression: increasing weight sparsity promotes circuitry reuse and generalization within tasks, without indiscriminately merging circuits across tasks. The lack of improved cross-task transfer is not a drawback, since our aim is to understand or steer a given task rather than use its circuit for another.

Although weight-sparse transformers are significantly more expensive to train than standard models (Gao et al., 2025), they provide a proof of concept that certain training schemes *can* improve circuit generalizability across templates.

6 Variance of Unfaithfulness Across Samples

Miller et al. (2024) observed large sample unfaithfulness variance across samples in circuits found by averaging edge scores, even when they are faithful at the population level.[2] One might think this is because the averaged scores better match some samples while fitting others less well. However, large variance persists even without score averaging. We perform CEAP-based circuit discovery *separately* for each sample given a target circuit size, eliminating influence from other samples, the resulting unfaithfulness still varies greatly across samples. Figure 4a shows such an example, which we use throughout this section. One sample (sample 78) has unfaithfulness $U \approx 20$, while most samples from the same template (`plural_1`) have low unfaithfulness.

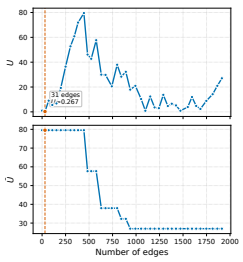
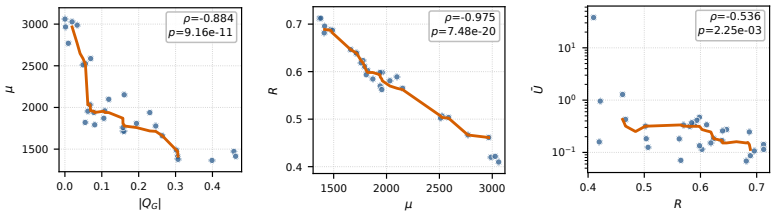


Figure 5: Normal and pessimistic unfaithfulness for sample 78 of SVA.

In this section, we show that, poor unfaithfulness scores typically stem from the normalization in the formulation of unfaithfulness, rather than from missing important edges. Comparing unfaithfulness values between samples does not reveal which sample's circuit is more sufficient for recovering the full model's behavior.

Single-sample unfaithfulness is difficult to interpret because it can vary highly non-monotonically with circuit size. Figure 5 shows how U changes with the circuit size for sample 78. It reaches 0.267 with just 31 edges but shoots up to around 80 when we increase the circuit size. Ideally, however, unfaithfulness should consistently decrease when we admit more edges into the circuit (Hadad et al., 2026). Single-sample unfaithfulness fails to meet this because it evaluates the behavior of the model at a single point in the input space, unrepresentative of the circuits broader behavior. We detail this in Appendix N. To avoid treating transient dips as evidence of faithful recovery, we define pessimistic unfaithfulness $\bar{U}$:

[2]The notion of population-level faithfulness has varied in previous works (Miller et al., 2024; Hanna et al., 2024; Wang et al., 2023), which we discuss in Appendix M.



(a) μ vs. $|Q_G|$. (b) R vs. μ. (c) $\bar{U}$ vs. R.

Figure 6: Spearman rank correlations reflecting Equation (4) that explain Figure 4b.

Definition 6.1. Let $U(m)$ denote the unfaithfulness of the circuit found by the algorithm with edge circuit size m. Pessimistic unfaithfulness is defined as $\bar{U}(m) = \max_{m \leq \hat{m} \leq |E|} U(\hat{m})$.

In experiments, we estimate $\bar{U}(m)$ by taking the maximum observed unfaithfulness over swept circuit sizes of at least m.

Next, we show that extremely large U need not indicate a large absolute discrepancy between the circuit and the full model. We define the unnormalized unfaithfulness $U' := |Q_{G'}(x_0, x_1) - Q_G(x_0, x_1)| = |M_{G'}(x_1) - M_G(x_1)|$ to directly measure the behavior gap between the circuit and the full model. We also define the pessimistic unnormalized unfaithfulness $\bar{U}'$ by enforcing monotonicity. In our experiments, we found that extremely high $\bar{U}$ is usually associated with near-zero $|Q_G|$, as shown in Figure 4b, where we obtained a negative ρ for Spearman rank correlation (SRC). Since Q_G is the normalization in (un)faithfulness, its being small may inflate $\bar{U}$ for certain samples. Supporting this explanation, samples with small $|Q_G|$ typically exhibit small unnormalized discrepancies $\bar{U}'$ (Figure 4c). *These observations suggest that extremely large $\bar{U}$ reflects amplification by a small normalization denominator rather than a large absolute gap in the task metric.*

6.1 The Correlation Between $\bar{U}$ and $|Q_G|$

A small denominator alone does not ensure a large ratio, because the numerator may also shrink. We therefore ask why, at a fixed circuit size, samples with small $|Q_G|$ exhibit much larger $\bar{U}$. An intuitive explanation is that, as $|Q_G|$ decreases, the excluded edges may become more important relative to those included in the circuit, which is supported by our analysis below.

Given a model with each edge scored on sample n, we arrange the absolute-valued scores of all its edges into a nonincreasing vector s^n. Let $S^n := \sum_{\ell=1}^{|E|} s^n_\ell$, and $u^n = s^n / S^n$ so that all u^n components sum up to 1. We define the normalized momentum to be $\mu^n := \sum_{\ell=1}^{|E|} u^n_\ell \times \ell$, which quantifies how heavy-tailed s^n is. From here on, for clarity, we introduce a superscript of sample index n for Q_G. We found that the samples with lower $|Q^n_G|$ have larger μ^n (Figure 6a). The lower panel of Figure 7 embodies this tendency for two samples with $|Q^{n_1}_G| > |Q^{n_2}_G|$. u^{n_1} has more mass at the head, while u^{n_2} has more at the tail.

Given an edge number m, our circuit discovery algorithm approximately selects the edges with the highest absolute scores to form the circuit. Define $C^n(m) := \sum_{\ell \in L} s^n_\ell$ where L is the set of indices of s^n that correspond to the edges admitted into the circuit. Then, $C^n(m)/S^n := R^n(m)$ can be interpreted as the ratio of score mass included in the circuit. Since the normalized scores sum to 1, more mass in the excluded tail leaves less in the selected head. We therefore expect heavier-tailed score distributions to have smaller $R^n(m)$, consistent with the negative correlation between μ^n and $R^n(m)$ in Figure 6b. A smaller $R^n(m)$ means that the circuit retains a smaller fraction of total edge importance. This suggests higher unfaithfulness, consistent with the negative correlation between $R^n(m)$ and $\bar{U}$ in Figure 6c. Based on these correlations, we derive the following relationships that account for the trend observed in Figure 4b.

$$|Q^n_G| \searrow \longrightarrow \mu^n \nearrow \longrightarrow R^n(m) \searrow \longrightarrow \bar{U} \nearrow \tag{4}$$

While the last two arrows in Equation (4) appear intuitive, the mechanism for the first one is unclear. We explain it through a neural mechanism we term *selective contribution scaling* in Section 6.2.

We verify all the correlation trends in this section, including the two in Figure 4 and the three in Figure 6, for all templates of SVA/IOI/greater-than, for both GPT-2 small and Pythia-160M, and for all graph sizes that we swept. The trends hold up well. Moreover, we check these trends using U in place of $\bar{U}$, in case our design of $\bar{U}$ introduces undesirable artifacts. U-based plots generally show the same trends, despite being noisier. All the results are presented in Appendix O.

6.2 A Mental Image for Selective Contribution Scaling

Previous results show that, if a set of edges is necessary and sufficient to produce the output (Figure 8), their conductance must sum up to Q_G^n (known as partition consistency (Dhamdhere et al., 2019)). In this sense, one can interpret the score as the *contribution* of the edge to the output. Suppose we have a pair of samples with $|Q_G^{n_1}| > |Q_G^{n_2}|$. It is reasonable to expect that the entries in s^{n_1} are larger than those in s^{n_2}, given partition consistency. Since we know in hindsight that u^{n_2} is more heavy-tailed than u^{n_1} (Figure 6a), we can infer that s^{n_1} is considerably higher than s^{n_2} at low indices, while the differences between them at high indices are mild. In this picture, moving from n_2 to n_1 amplifies the scores of important edges disproportionately, while the low-scoring tail changes comparatively little.[3] This is what we call selective contribution scaling, as illustrated in the upper panel of Figure 7. We verify the mental image of Figure 7 with a concrete sample pair in Appendix P.

That there exist specialized parts of the model reacting to structurally specific stimuli (tasks expressed in templates) and other parts responding only weakly to them aligns with the notion of sparse coding, a widely endorsed framework of information processing in neuroscience (Barlow, 1961; Olshausen & Field, 1996; 1997; Vinje & Gallant, 2000; Lonnqvist et al., 2023). Nonetheless, in our case, it links unfaithfulness to the insignificant magnitude of Q_G, which weakens unfaithfulness as a definitive marker of performance recovery. By the same token, we also anticipate great cross-sample variance in faithfulness-style normalization-dependent metrics measured on systems that exhibit sparse coding.

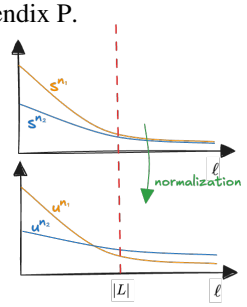


Figure 7: Mental image of selective contribution scaling.

7 Conclusion, Discussion and Future Work

In this paper, we systematically investigate the three types of variance in circuit discovery algorithms. We find that resampling and phrasing variance can be mitigated with more principled scoring and weight sparsity, respectively. Sample-wise variance of unfaithfulness is largely benign, reflecting a normalization effect.

Our findings also inform future practice. First, conductance is a more principled way of scoring compared to IG, as it satisfies additive order preservation and lowers resampling variance. Second, as of now, template-induced circuit variance is not completely eliminated yet, despite being mitigated by weight sparsity. Hence, statements involving task circuits should be carefully tested for robustness under changes in template (Meng et al., 2022; 2023). Third, cross-sample comparisons of (un)faithfulness require care: normalization can amplify differences associated with selective contribution scaling. Nonetheless, (un)faithfulness remains informative for tracking how circuit quality improves as we increase the permitted circuit size. In this regard, pessimism serves as a useful complement to reduce interpretational noise.

Notably, as weight-sparse transformers are extremely expensive to train, developing more practical training schemes to improve cross-template circuit generalizability constitutes a meaningful direction for future research. Conceptually, our findings in Section 5 also offer a circuit-level perspective on the view of intelligence as compression (Wolff, 2013; Schmidhuber, 2008): weight sparsity encourages circuitry reuse across templates rather than redundant template-specific mechanisms. Under this view, weight sparse training may foster a more intelligent class of models, beside its benefits for circuit interpretability and steering. We elaborate on this connection in Appendix L.

[3] Note that we are implicitly assuming that the edge orderings in s^{n_1} and s^{n_2} coincide. While this is not strictly true, they should be very similar, because, as we show in Section 5, samples sharing the same template tend to exhibit similar scoring patterns.

AI USE STATEMENT

In this work, we used generative AI tools for generating synthetic datasets, assisting in the writing and refinement of proofs based on ideas developed by the authors, and supporting data analysis through visualization. The proof assistance included clarifying assumptions and checking mathematical arguments. We have not used generative AI tools for method implementation or translation, and we do not designate any additional required-disclosure tasks as not applicable to this work. Additionally, we used generative AI tools for refactoring software code and editing the manuscript to improve language and readability. We have reviewed all AI-assisted work, including the generated datasets, mathematical arguments, and visualizations, and checked all AI-assisted code for correctness. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Alaa Anani, Tobias Lorenz, Bernt Schiele, Mario Fritz, and Jonas Fischer. Certified circuits: Stability guarantees for mechanistic circuits. *arXiv preprint arXiv:2602.22968*, 2026.

Aryaman Arora, Zhengxuan Wu, Jacob Steinhardt, and Sarah Schwettmann. Language model circuits are sparse in the neuron basis. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=OrviwFWcN4.

Horace Barlow. Possible principles underlying the transformations of sensory messages. *Sensory Communication*, 1, 01 1961. doi: 10.7551/mitpress/9780262518420.003.0013.

Niladri S. Chatterji, Philip M. Long, and Peter L. Bartlett. When does gradient descent with logistic loss find interpolating two-layer networks?, 2021. URL https://arxiv.org/abs/2012.02409.

Xi Chen, Mingyu Jin, Jingcheng Niu, Yutong Yin, Jinman Zhao, Bangwei Guo, Dimitris N. Metaxas, Zhaoran Wang, Yutao Yue, and Gerald Penn. All circuits lead to rome: Rethinking functional anisotropy in circuit and sheaf discovery for llms, 2026. URL https://arxiv.org/abs/2605.12671.

Stephen Cheng, Sarah Wiegreffe, and Dinesh Manocha. What drives representation steering? a mechanistic case study on steering refusal. *arXiv preprint arXiv:2604.08524*, 2026.

Hyunjin Cho, Youngji Roh, and Jaehyung Kim. Shared semantics, divergent mechanisms: Unsupervised feature discovery by aligning semantics and mechanisms, 2026. URL https://arxiv.org/abs/2606.08236.

Arthur Conmy, Augustine Mavor-Parker, Aengus Lynch, Stefan Heimersheim, and Adrià Garriga-Alonso. Towards automated circuit discovery for mechanistic interpretability. *Advances in Neural Information Processing Systems*, 36:16318–16352, 2023.

Kedar Dhamdhere, Mukund Sundararajan, and Qiqi Yan. How important is a neuron. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=SylKoo0cKm.

Gabriel Franco, Lucas M. Tassis, Azalea Rohr, and Mark Crovella. Finding highly interpretable prompt-specific circuits in language models, 2026. URL https://arxiv.org/abs/2602.13483v1.

Leo Gao, Achyuta Rajaram, Jacob Coxon, Soham V Govande, Bowen Baker, and Dan Mossing. Weight-sparse transformers have interpretable circuits. *arXiv preprint arXiv:2511.13653*, 2025.

Itamar Hadad, Guy Katz, and Shahaf Bassan. Formal mechanistic interpretability: Automated circuit discovery with provable guarantees. *arXiv preprint arXiv:2602.16823*, 2026.

Michael Hanna, Ollie Liu, and Alexandre Variengien. How does GPT-2 compute greater-than?: Interpreting mathematical abilities in a pre-trained language model. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=p4PckNQR8k.

Michael Hanna, Sandro Pezzelle, and Yonatan Belinkov. Have faith in faithfulness: Going beyond circuit overlap when finding model mechanisms. In *ICML 2024 Workshop on Mechanistic Interpretability*, 2024. URL https://openreview.net/forum?id=grXgesr5dT.

Marcus Hutter. A theory of universal artificial intelligence based on algorithmic complexity, 2000. URL https://arxiv.org/abs/cs/0004001.

Jack Lindsey, Wes Gurnee, Emmanuel Ameisen, Brian Chen, Adam Pearce, Nicholas L. Turner, Craig Citro, David Abrahams, Shan Carter, Basil Hosmer, Jonathan Marcus, Michael Sklar, Adly Templeton, Trenton Bricken, Callum McDougall, Hoagy Cunningham, Thomas Henighan, Adam Jermyn, Andy Jones, Andrew Persic, Zhenyi Qi, T. Ben Thompson, Sam Zimmerman, Kelley Rivoire, Thomas Conerly, Chris Olah, and Joshua Batson. On the biology of a large language model. *Transformer Circuits Thread*, 2025. URL https://transformer-circuits.pub/2025/attribution-graphs/biology.html.

Ben Lonnqvist, Zhengqing Wu, and Michael H Herzog. Latent noise segmentation: How neural noise leads to the emergence of segmentation and grouping. *arXiv preprint arXiv:2309.16515*, 2023.

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. *arXiv preprint arXiv:2403.19647*, 2024.

David Marr. *Vision: A Computational Investigation into the Human Representation and Processing of Visual Information*. The MIT Press, 07 2010. ISBN 9780262514620. doi: 10.7551/mitpress/9780262514620.001.0001. URL https://doi.org/10.7551/mitpress/9780262514620.001.0001.

Maxime Méloux, François Portet, and Maxime Peyrard. Mechanistic interpretability as statistical estimation: A variance analysis of eap-ig. *arXiv preprint arXiv:2510.00845*, 2025.

Kevin Meng, David Bau, Alex J Andonian, and Yonatan Belinkov. Locating and editing factual associations in GPT. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho (eds.), *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=-h6WAS6eE4.

Kevin Meng, Arnab Sen Sharma, Alex J Andonian, Yonatan Belinkov, and David Bau. Mass-editing memory in a transformer. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=MkbcAHIYgyS.

Joseph Miller, Bilal Chughtai, and William Saunders. Transformer circuit evaluation metrics are not robust. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=zSf8PJyQb2.

Neel Nanda. Attribution patching: Activation patching at industrial scale. urlhttps://www.neelnanda.io/mechanistic-interpretability/attribution-patching, February 2022. Last accessed July 25, 2025.

Benjamin Newman, Kai-Siang Ang, Julia Gong, and John Hewitt. Refining targeted syntactic evaluation of language models. In Kristina Toutanova, Anna Rumshisky, Luke Zettlemoyer, Dilek Hakkani-Tur, Iz Beltagy, Steven Bethard, Ryan Cotterell, Tanmoy Chakraborty, and Yichao Zhou (eds.), *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 3710–3723, Online, June 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.naacl-main.290. URL https://aclanthology.org/2021.naacl-main.290/.

nostalgebraist. Interpreting gpt: The logit lens. https://www.lesswrong.com/posts/AcKRB8wDpdaN6v6ru/interpreting-gpt-the-logit-lens, 2020. LessWrong blog post, accessed 2026-05-07.

Bruno A. Olshausen and David J. Field. Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature*, 381:607–609, 1996. URL https://api.semanticscholar.org/CorpusID:4358477.

Bruno A. Olshausen and David J. Field. Sparse coding with an overcomplete basis set: A strategy employed by v1? *Vision Research*, 37(23):3311–3325, 1997. ISSN 0042-6989. doi: https://doi.org/10.1016/S0042-6989(97)00169-7. URL https://www.sciencedirect.com/science/article/pii/S0042698997001697.

Nils Palumbo, Ravi Mangal, Zifan Wang, Saranya Vijayakumar, Corina S. Pasareanu, and Somesh Jha. Validating mechanistic interpretations: An axiomatic approach. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=qBtomgvLwn.

Daking Rai, Mor Geva, and Ziyu Yao. Data-driven circuit discovery for interpretability of language models, 2026. URL https://arxiv.org/abs/2605.09129.

Jürgen Schmidhuber. Driven by compression progress: A simple principle explains essential aspects of subjective beauty, novelty, surprise, interestingness, attention, curiosity, creativity, art, science, music, jokes. In *Workshop on anticipatory behavior in adaptive learning systems*, pp. 48–76. Springer, 2008.

Avanti Shrikumar, Jocelin Su, and Anshul Kundaje. Computationally efficient measures of internal neuron importance, 2018. URL https://arxiv.org/abs/1807.09946.

Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. In Doina Precup and Yee Whye Teh (eds.), *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pp. 3319–3328. PMLR, 06–11 Aug 2017. URL https://proceedings.mlr.press/v70/sundararajan17a.html.

Aaquib Syed, Can Rager, and Arthur Conmy. Attribution patching outperforms automated circuit discovery. In *Proceedings of the 7th BlackboxNLP Workshop: Analyzing and Interpreting Neural Networks for NLP*, pp. 407–416, 2024.

William Vinje and Jack Gallant. Sparse coding and decorrelation in primary visual cortex during natural vision. *Science (New York, N.Y.)*, 287:1273–6, 03 2000. doi: 10.1126/science.287.5456.1273.

Kevin Ro Wang, Alexandre Variengien, Arthur Conmy, Buck Shlegeris, and Jacob Steinhardt. Interpretability in the wild: a circuit for indirect object identification in GPT-2 small. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=NpsVSN6o4ul.

Xu Wang, Yan Hu, Wenyu Du, Reynold Cheng, Benyou Wang, and Difan Zou. Towards understanding fine-tuning mechanisms of LLMs via circuit analysis. In *ICLR 2025 Workshop on Building Trust in Language Models and Applications*, 2025. URL https://openreview.net/forum?id=Z9qzta1yiK.

Christian H. Weiß, Fukang Zhu, and Aisouda Hoshiyar. Softplus ingarch models. *Statistica Sinica*, 32(2):pp. 1099–1120, 2022. ISSN 10170405, 19968507. URL https://www.jstor.org/stable/27118810.

J Gerard Wolff. Computing as compression: the sp theory of intelligence, 2013. URL https://arxiv.org/abs/1303.2013.

Yunzhi Yao, Ningyu Zhang, Zekun Xi, Mengru Wang, Ziwen Xu, Shumin Deng, and Huajun Chen. Knowledge circuits in pretrained transformers. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 118571–118602. Curran Associates, Inc., 2024. doi: 10.52202/079017-3765. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/d6df31b1be98e04be48af8bedb95b499-Paper-Conference.pdf.

Hengyuan Zhang, Zhihao Zhang, Mingyang Wang, Zunhai Su, Yiwei Wang, Qianli Wang, Shuzhou Yuan, Ercong Nie, Xufeng Duan, Feijiang Han, Qibo Xue, Zeping Yu, Chenming Shang, Xiao Liang, Jing Xiong, Hui Shen, Chaofan Tao, Zhengwu Liu, Senjie Jin, Zhiheng Xi, Dongdong Zhang, Sophia Ananiadou, Tao Gui, Ruobing Xie, Hayden Kwok-Hay So, Hinrich Schütze,

Xuanjing Huang, Qi Zhang, and Ngai Wong. Locate, steer, and improve: A practical survey of actionable mechanistic interpretability in large language models, 2026. URL https://arxiv.org/abs/2601.14004.

A Circuit Graph and Activation Patching

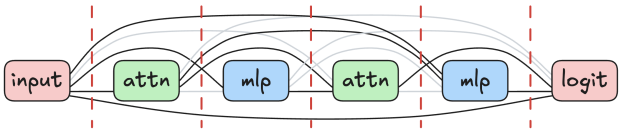


Figure 8: The computational graph of a mini Transformer. The residual addition is split into separate edges. The black edges are the ones included in the subgraph/circuit, while the grey ones are excluded. During a patched run, modules will take the results from previous ones through black edges but the grey edges will be fixed at activations corresponding to the corrupted input. Each red dotted line intercepts a set of edges that is necessary and sufficient for producing the output.

B Experimental Details

In this paper, we use the greedy method (Hanna et al., 2024), which first finds a circuit with a target edge count and then prunes the edges without parents or children.

Following previous conventions (Hanna et al., 2024; Nanda, 2022), we used `logit_diff` for tasks whose correct and incorrect answers are single logits (IOI, single-double-quote, and with-open). For other tasks, we used `prob_diff`.

For GPT-2 and Pythia experiments, we used integration step $K = 200$ for both EAP-IG and CEAP. For sparse-transformer experiments (Gao et al., 2025), we used 150.

C Numerical Issue Caused by KL Divergence

Besides `prob diff`, another commonly used metric is `logit diff` (Nanda, 2022; Hanna et al., 2024). For the SVA example in Section 2, this is the sum of the logits corresponding to singular verbs minus the sum for plural verbs.

Hanna et al. (2024); Conmy et al. (2023) also recommended using the KL divergence as a metric, as it is in principle applicable to all tasks. In practice, however, we found KL divergence to be restrictive: it cannot be used for models that assign zero probability to some tokens, since this leads to an infinite KL value, which we observe *e.g.*, for Pythia-160M. While $\mathrm{Softmax}(\cdot)$ should theoretically ensure that every token has nonzero probability, numerical underflow (which occurs for both `fp32` and `fp64`) frequently produces exact zeros in the probabilities. To keep the exposition uncluttered, we therefore report only the `logit`/`prob diff` metrics in the main paper. The experiments that we carried out for KL divergence do not show qualitatively different results from the conclusions we present here.

D Dataset Templates

We use template-generated datasets so that clean and corrupted prompts differ in a controlled way. Tables 1 to 6 summarize the templates used in our experiments. Placeholders in square brackets are filled with names, nouns, variable names, or code literals during dataset construction. For the tasks customized for sparse-transformers (Gao et al., 2025), each base template generates a paired contrast. For `single_double_quote`, we show both paired directions explicitly using the `_s` and `_d` suffixes. For `else_elif`, we show one representative direction in the table and omit the `_else`/`_elif` suffixes from the displayed template names for space; the generated data and visualizations keep those suffixes. For `with_open`, we likewise show the read direction only; the generated data use `_r`/`_w` suffixes for the read and write directions.

Table 1: SVA dataset templates. Each row shows a prompt contrast where the correct answer is either a singular or plural verb form. In the placeholders, sg and pl denote singular and plural forms, respectively.

Template	Clean prompt template example	Corrupted prompt template example	Correct	Incorrect
single_0	The [Noun_{sg}] that [Verb_{sg}] the [Noun]	The [Noun_{pl}] that [Verb_{pl}] the [Noun]	singular verb	plural verb
plural_0	The [Noun_{pl}] that [Verb_{pl}] the [Noun]	The [Noun_{sg}] that [Verb_{sg}] the [Noun]	plural verb	singular verb
single_1	The [Noun_{sg}] from the [Noun_{pl}]	The [Noun_{pl}] from the [Noun_{pl}]	singular verb	plural verb
plural_1	The [Noun_{pl}] from the [Noun_{pl}]	The [Noun_{sg}] from the [Noun_{pl}]	plural verb	singular verb
single_2	The [Noun_{pl}] that the [Noun_{sg}]	The [Noun_{pl}] that the [Noun_{pl}]	singular verb	plural verb
plural_2	The [Noun_{pl}] that the [Noun_{pl}]	The [Noun_{pl}] that the [Noun_{sg}]	plural verb	singular verb
single_3	The [Noun_{pl}] the [Noun_{sg}]	The [Noun_{pl}] the [Noun_{pl}]	singular verb	plural verb
plural_3	The [Noun_{pl}] the [Noun_{pl}]	The [Noun_{pl}] the [Noun_{sg}]	plural verb	singular verb
single_4	The [Noun_{pl}] said the [Noun_{sg}]	The [Noun_{pl}] said the [Noun_{pl}]	singular verb	plural verb
plural_4	The [Noun_{pl}] said the [Noun_{pl}]	The [Noun_{pl}] said the [Noun_{sg}]	plural verb	singular verb
single_5	The [Noun_{sg}] the [Noun_{pl}] [Verb]	The [Noun_{pl}] the [Noun_{pl}] [Verb]	singular verb	plural verb
plural_5	The [Noun_{pl}] the [Noun_{pl}] [Verb]	The [Noun_{sg}] the [Noun_{pl}] [Verb]	plural verb	singular verb
single_6	The [Noun_{pl}] that the [Noun_{sg}]	The [Noun_{pl}] that the [Noun_{pl}]	singular verb	plural verb
plural_6	The [Noun_{pl}] that the [Noun_{pl}]	The [Noun_{pl}] that the [Noun_{sg}]	plural verb	singular verb
single_7	The [Noun_{pl}] the [Noun_{sg}]	The [Noun_{pl}] the [Noun_{pl}]	singular verb	plural verb
plural_7	The [Noun_{pl}] the [Noun_{pl}]	The [Noun_{pl}] the [Noun_{sg}]	plural verb	singular verb
single_8	The [Noun_{sg}] that the [Noun_{pl}] [Verb]	The [Noun_{pl}] that the [Noun_{pl}] [Verb]	singular verb	plural verb
plural_8	The [Noun_{pl}] that the [Noun_{pl}] [Verb]	The [Noun_{sg}] that the [Noun_{pl}] [Verb]	plural verb	singular verb
single_9	The [Noun_{sg}] that the [Noun_{pl}] [Verb]	The [Noun_{pl}] that the [Noun_{pl}] [Verb]	singular verb	plural verb
plural_9	The [Noun_{pl}] that the [Noun_{pl}] [Verb]	The [Noun_{sg}] that the [Noun_{pl}] [Verb]	plural verb	singular verb
single_10	The [Noun_{sg}]	The [Noun_{pl}]	singular verb	plural verb
plural_10	The [Noun_{pl}]	The [Noun_{sg}]	plural verb	singular verb
single_11	The [Noun_{sg}] [Verb_{sg}] [VP] and	The [Noun_{pl}] [Verb_{pl}] [VP] and	singular verb	plural verb
plural_11	The [Noun_{pl}] [Verb_{pl}] [VP] and	The [Noun_{sg}] [Verb_{sg}] [VP] and	plural verb	singular verb
single_12	The [Noun_{sg}] next to the [Noun_{pl}]	The [Noun_{pl}] next to the [Noun_{pl}]	singular verb	plural verb
plural_12	The [Noun_{pl}] next to the [Noun_{pl}]	The [Noun_{sg}] next to the [Noun_{pl}]	plural verb	singular verb
single_13	The [Noun_{sg}] [Verb_{sg}] and	The [Noun_{pl}] [Verb_{pl}] and	singular verb	plural verb
plural_13	The [Noun_{pl}] [Verb_{pl}] and	The [Noun_{sg}] [Verb_{sg}] and	plural verb	singular verb
single_14	The [Noun_{sg}] the [Noun_{pl}] [Verb]	The [Noun_{pl}] the [Noun_{pl}] [Verb]	singular verb	plural verb
plural_14	The [Noun_{pl}] the [Noun_{pl}] [Verb]	The [Noun_{sg}] the [Noun_{pl}] [Verb]	plural verb	singular verb

Table 2: IOI dataset templates. Each clean prompt is truncated immediately before the indirect-object answer; the corrupted prompt replaces the cueing name with a distractor [C].

Template	Clean prompt template example	Corrupted prompt template example	Correct	Incorrect
ABBA_00	Then, [A] and [B] went to the [PLACE]. [B] gave a [OBJECT] to	Then, [A] and [B] went to the [PLACE]. [C] gave a [OBJECT] to	[A]	[B]
ABBA_01	Then, [A] and [B] had a lot of fun at the [PLACE]. [B] gave a [OBJECT] to	Then, [A] and [B] had a lot of fun at the [PLACE]. [C] gave a [OBJECT] to	[A]	[B]
ABBA_02	Then, [A] and [B] were working at the [PLACE]. [B] decided to give a [OBJECT] to	Then, [A] and [B] were working at the [PLACE]. [C] decided to give a [OBJECT] to	[A]	[B]
ABBA_03	Then, [A] and [B] were thinking about going to the [PLACE]. [B] wanted to give a [OBJECT] to	Then, [A] and [B] were thinking about going to the [PLACE]. [C] wanted to give a [OBJECT] to	[A]	[B]
ABBA_04	Then, [A] and [B] had a long argument, and afterwards [B] said to	Then, [A] and [B] had a long argument, and afterwards [C] said to	[A]	[B]
ABBA_05	After [A] and [B] went to the [PLACE], [B] gave a [OBJECT] to	After [A] and [B] went to the [PLACE], [C] gave a [OBJECT] to	[A]	[B]
ABBA_06	When [A] and [B] got a [OBJECT] at the [PLACE], [B] decided to give it to	When [A] and [B] got a [OBJECT] at the [PLACE], [C] decided to give it to	[A]	[B]
BABA_00	Then, [B] and [A] went to the [PLACE]. [B] gave a [OBJECT] to	Then, [B] and [A] went to the [PLACE]. [C] gave a [OBJECT] to	[A]	[B]
BABA_01	Then, [B] and [A] had a lot of fun at the [PLACE]. [B] gave a [OBJECT] to	Then, [B] and [A] had a lot of fun at the [PLACE]. [C] gave a [OBJECT] to	[A]	[B]
BABA_02	Then, [B] and [A] were working at the [PLACE]. [B] decided to give a [OBJECT] to	Then, [B] and [A] were working at the [PLACE]. [C] decided to give a [OBJECT] to	[A]	[B]
BABA_03	Then, [B] and [A] were thinking about going to the [PLACE]. [B] wanted to give a [OBJECT] to	Then, [B] and [A] were thinking about going to the [PLACE]. [C] wanted to give a [OBJECT] to	[A]	[B]
BABA_04	Then, [B] and [A] had a long argument, and afterwards [B] said to	Then, [B] and [A] had a long argument, and afterwards [C] said to	[A]	[B]
BABA_05	After [B] and [A] went to the [PLACE], [B] gave a [OBJECT] to	After [B] and [A] went to the [PLACE], [C] gave a [OBJECT] to	[A]	[B]
BABA_06	When [B] and [A] got a [OBJECT] at the [PLACE], [B] decided to give it to	When [B] and [A] got a [OBJECT] at the [PLACE], [C] decided to give it to	[A]	[B]

Table 3: greater-than dataset template example. The task asks whether the next two-digit year suffix is greater than the suffix xy implied by the clean prompt.

Template	Clean prompt template example	Corrupted prompt template example	Correct	Incorrect
x	The [NOUN] lasted from the year 13xy to the year 13	The [NOUN] lasted from the year 1301 to the year 13	suffix $> xy$	suffix $\leq xy$

Table 4: single-double-quote dataset templates. Each base template is shown in both directions: _s expects the closing single-quote-parenthesis token, and _d expects the closing double-quote-parenthesis token.

Template	Clean prompt template example	Corrupted prompt template example	Correct	Incorrect
append_call_s	`if [date bounds fail]:` `  [RESULT_LIST].append('Invalid Input`	`if [date bounds fail]:` `  [RESULT_LIST].append("Invalid Input`	')	")
append_call_d	`if [date bounds fail]:` `  [RESULT_LIST].append("Invalid Input`	`if [date bounds fail]:` `  [RESULT_LIST].append('Invalid Input`	")	')
constructor_call_s	`[OBJECT_VAR] = Project(` `  '[PROJECT_NAME]`	`[OBJECT_VAR] = Project(` `  "[PROJECT_NAME]`	')	")
constructor_call_d	`[OBJECT_VAR] = Project(` `  "[PROJECT_NAME]`	`[OBJECT_VAR] = Project(` `  '[PROJECT_NAME]`	")	')

for_loop_s	`for [ITEM_VAR] in [ITEM_SET]:`
	`  if len([ITEM_VAR]) == 0:`
	`  print('invalid`
		`for [ITEM_VAR] in [ITEM_SET]:`
		`  if len([ITEM_VAR]) == 0:`
		`  print("invalid`
			')	")

for_loop_d	`for [ITEM_VAR] in [ITEM_SET]:`
	`  if len([ITEM_VAR]) == 0:`
	`  print("invalid`
		`for [ITEM_VAR] in [ITEM_SET]:`
		`  if len([ITEM_VAR]) == 0:`
		`  print('invalid`
			")	')

function_call_s	`[FUNC_NAME]('[FILE_NAME]`	`[FUNC_NAME]("[FILE_NAME]`	')	")
function_call_d	`[FUNC_NAME]("[FILE_NAME]`	`[FUNC_NAME]('[FILE_NAME]`	")	')

if_clause_1_s	`if [COND]:`
	`  print('There is no topological sort,`
	`  the graph has a cycle`
		`if [COND]:`
		`  print("There is no topological sort,`
		`  the graph has a cycle`
			')	")

if_clause_1_d	`if [COND]:`
	`  print("There is no topological sort,`
	`  the graph has a cycle`
		`if [COND]:`
		`  print('There is no topological sort,`
		`  the graph has a cycle`
			")	')

if_clause_2_s	`if degree not in range(3, [LIMIT_VAR]-1):`
	`  print('Degree should be in the range`
	`  [3, [LIMIT_VAR]-1]`
		`if degree not in range(3, [LIMIT_VAR]-1):`
		`  print("Degree should be in the range`
		`  [3, [LIMIT_VAR]-1]`
			')	")

if_clause_2_d	`if degree not in range(3, [LIMIT_VAR]-1):`
	`  print("Degree should be in the range`
	`  [3, [LIMIT_VAR]-1]`
		`if degree not in range(3, [LIMIT_VAR]-1):`
		`  print('Degree should be in the range`
		`  [3, [LIMIT_VAR]-1]`
			")	')

if_clause_3_s	`if [ALT_VAR] < [DIST_VAR].get(` `  [NODE_VAR], float('inf`	`if [ALT_VAR] < [DIST_VAR].get(` `  [NODE_VAR], float("inf`	')	")
if_clause_3_d	`if [ALT_VAR] < [DIST_VAR].get(` `  [NODE_VAR], float("inf`	`if [ALT_VAR] < [DIST_VAR].get(` `  [NODE_VAR], float('inf`	")	')
method_call_s	`[GRAPH_VAR].read_data(` `  '[GRAPH_FILE]`	`[GRAPH_VAR].read_data(` `  "[GRAPH_FILE]`	')	")
method_call_d	`[GRAPH_VAR].read_data(` `  "[GRAPH_FILE]`	`[GRAPH_VAR].read_data(` `  '[GRAPH_FILE]`	")	')

nested_loop_s	`for [OUTER] in [SET]:`
	`  for [INNER] in [OUTER]:`
	`  if ([INNER] == ()):`
	`  print('No [INNER] at`
	`  the current iteration`
		`for [OUTER] in [SET]:`
		`  for [INNER] in [OUTER]:`
		`  if ([INNER] == ()):`
		`  print("No [INNER] at`
		`  the current iteration`
			')	")

nested_loop_d	`for [OUTER] in [SET]:`
	`  for [INNER] in [OUTER]:`
	`  if ([INNER] == ()):`
	`  print("No [INNER] at`
	`  the current iteration`
		`for [OUTER] in [SET]:`
		`  for [INNER] in [OUTER]:`
		`  if ([INNER] == ()):`
		`  print('No [INNER] at`
		`  the current iteration`
			")	')

print_stmt_1_s	`print('[PRINT_TEXT]`	`print("[PRINT_TEXT]`	')	")
print_stmt_1_d	`print("[PRINT_TEXT]`	`print('[PRINT_TEXT]`	")	')
print_stmt_2_s	`print('[USAGE_TEXT]`	`print("[USAGE_TEXT]`	')	")
print_stmt_2_d	`print("[USAGE_TEXT]`	`print('[USAGE_TEXT]`	")	')
print_stmt_3_s	`print('[QUESTION_TEXT]`	`print("[QUESTION_TEXT]`	')	")
print_stmt_3_d	`print("[QUESTION_TEXT]`	`print('[QUESTION_TEXT]`	")	')
print_stmt_4_s	`  print('[STATUS_TEXT]`	`  print("[STATUS_TEXT]`	')	")
print_stmt_4_d	`  print("[STATUS_TEXT]`	`  print('[STATUS_TEXT]`	")	')
raise_error_s	`raise ValueError('[ERROR_TEXT]`	`raise ValueError("[ERROR_TEXT]`	')	")
raise_error_d	`raise ValueError("[ERROR_TEXT]`	`raise ValueError('[ERROR_TEXT]`	")	')

string_split_s	`def [FUNC]([ARG]):`
	`  [LEFT], [RIGHT] =`
	`  [ARG].split('-`
		`def [FUNC]([ARG]):`
		`  [LEFT], [RIGHT] =`
		`  [ARG].split("-`
			')	")

string_split_d	`def [FUNC]([ARG]):`
	`  [LEFT], [RIGHT] =`
	`  [ARG].split("-`
		`def [FUNC]([ARG]):`
		`  [LEFT], [RIGHT] =`
		`  [ARG].split('-`
			")	')

strftime_call_s	`[TIME_VAR] = datetime.datetime.now()` `  .strftime('%Y%m%d_%H%M%S`	`[TIME_VAR] = datetime.datetime.now()` `  .strftime("%Y%m%d_%H%M%S`	')	")
strftime_call_d	`[TIME_VAR] = datetime.datetime.now()` `  .strftime("%Y%m%d_%H%M%S`	`[TIME_VAR] = datetime.datetime.now()` `  .strftime('%Y%m%d_%H%M%S`	")	')

Table 5: else-elif dataset templates. The task contrasts Python branches where `else` should be followed by `:\n`, while `elif` should be followed by any other token. In visualizations, template labels additionally carry `_else` or `_elif` suffixes for the two directions; these suffixes are omitted here for space, and the prompt examples show one representative direction.

Template	Clean prompt template example	Corrupted prompt template example	Correct	Incorrect

return_result	`if [RESULT]:`
	`  [CACHE] = [RESULT]`
	`  return [RESULT]`
	`else`
		`if [RESULT]:`
		`  [CACHE] = [RESULT]`
		`  return [RESULT]`
		`elif`
			`:\n`	other tokens

answer_check	`if eval([PROBLEM]) == [ANSWER]:`
	`  print([MSG])`
	`elif`
		`if eval([PROBLEM]) == [ANSWER]:`
		`  print([MSG])`
		`else`
			other tokens	`:\n`

append_guard	`if [COUNT_MAP][[NODE].topic] >= [TARGET]:`
	`  [LIST].append([NODE])`
	`else`
		`if [COUNT_MAP][[NODE].topic] >= [TARGET]:\n`
		`  [LIST].append([NODE])`
		`elif`
			`:\n`	other tokens

program_dir_chain	`if [DIR] == 0:`
	`  [DIR] = [NEG]`
	`elif [DIR] == 1:`
	`  [DIR] = [POS]`
	`elif`
		`if [DIR] == 0:`
		`  [DIR] = [NEG]`
		`elif [DIR] == 1:`
		`  [DIR] = [POS]`
		`else`
			other tokens	`:\n`

schedule_branch	`if [TIME] > [LAST]:`
	`  [START] = [LAST] + 1`
	`else`
		`if [TIME] > [LAST]:`
		`  [START] = [LAST] + 1`
		`elif`
			`:\n`	other tokens

strategy_branch	`if [STRATEGY] == 'bfs':`
	`  return bfs(...)`
	`elif [STRATEGY] == 'dfs':`
	`  return dfs(...)`
	`elif`
		`if [STRATEGY] == 'bfs':`
		`  return bfs(...)`
		`elif [STRATEGY] == 'dfs':`
		`  return dfs(...)`
		`else`
			other tokens	`:\n`

equality_case	`if [CONSTRAINT].is_equality():`
	`  return [Z3] == -[COEFF]['1']`
	`else`
		`if [CONSTRAINT].is_equality():`
		`  return [Z3] == -[COEFF]['1']`
		`elif`
			`:\n`	other tokens

stats_update	`if [ROOT] not in [STATS]:`
	`  [STATS][[ROOT]] =`
	`  ([AREA], [COUNT])`
	`elif`
		`if [ROOT] not in [STATS]:`
		`  [STATS][[ROOT]] =`
		`  ([AREA], [COUNT])`
		`else`
			other tokens	`:\n`

inline_threshold	`if ([NUM] <= [LIMIT]):`
	`  return [FUNC_CALL]`
	`else`
		`if ([NUM] <= [LIMIT]):`
		`  return [FUNC_CALL]`
		`elif`
			`:\n`	other tokens

string_align	`if [CHAR] in [STRING]:`
	`  [LEFT], [RIGHT] =`
	`  [STRING].split([CHAR], 1)`
	`elif`
		`if [CHAR] in [STRING]:`
		`  [LEFT], [RIGHT] =`
		`  [STRING].split([CHAR], 1)`
		`else`
			other tokens	`:\n`

binary_search	`if [DATA][[INDEX]] < [VALUE]:`
	`  [RESULT] = [INDEX]`
	`  [LOW] = [INDEX] + 1`
	`else`
		`if [DATA][[INDEX]] < [VALUE]:`
		`  [RESULT] = [INDEX]`
		`  [LOW] = [INDEX] + 1`
		`elif`
			`:\n`	other tokens

truthy_call	`if [ROOM]:`
	`  set_parameter([BOARD], [PARAM],`
	`  [ROOM_NAME] + ' - ' + [ROOM])`
	`elif`
		`if [ROOM]:`
		`  set_parameter([BOARD], [PARAM],`
		`  [ROOM_NAME] + ' - ' + [ROOM])`
		`else`
			other tokens	`:\n`

compare_objects	`if isinstance([LEFT], (list, tuple)):`
	`  if isinstance([RIGHT], (list, tuple)):`
	`  compare_objects(...)`
	`  else`
		`if isinstance([LEFT], (list, tuple)):\n`
		`  if isinstance([RIGHT], (list, tuple)):`
		`  compare_objects(...)`
		`  elif`
			`:\n`	other tokens

cell_toggle	`if [BOARD][[ROW]][[COL]] == [OFF]:`
	`  [BOARD][[ROW]][[COL]] = 1`
	`elif`
		`if [BOARD][[ROW]][[COL]] == [OFF]:`
		`  [BOARD][[ROW]][[COL]] = 1`
		`else`
			other tokens	`:\n`

random_split	`if random.random() < 0.5:`
	`  [OUT].append(`
	`  f"ML,{[PEER]},{[ELEMENT]}")`
	`else`
		`if random.random() < 0.5:`
		`  [OUT].append(`
		`  f"ML,{[PEER]},{[ELEMENT]}")`
		`elif`
			`:\n`	other tokens

none_default	`if [END] is None:`
	`  [GOAL] = ([ROWS]-1, [COLS]-1)`
	`elif`
		`if [END] is None:`
		`  [GOAL] = ([ROWS]-1, [COLS]-1)`
		`else`
			other tokens	`:\n`

Table 6: with-open dataset templates. The clean prompt shown in each row contains `input.txt` and expects the next token `r`, whereas the corrupted prompt contains `output.txt` and expects `w`. The generated dataset also includes the reverse direction. Template labels carry `_r` or `_w` suffixes in the generated data; the suffixes are omitted here for space.

Template	Clean prompt template example	Corrupted prompt template example	Correct	Incorrect

diary_function	def process_diary([PATH]):
	  [ENTRIES] = []
	  with open("input.txt", "
		def process_diary([PATH]):
		  [ENTRIES] = []
		  with open("output.txt", "
			r	w

descriptor_class	class [DESCRIPTOR]:
	  def calculate(self):
	  with open("input.txt", "
		class [DESCRIPTOR]:
		  def calculate(self):
		  with open("output.txt", "
			r	w

cli_try	if __name__ == '__main__':
	  try:
	  with open("input.txt", "
		if __name__ == '__main__':
		  try:
		  with open("output.txt", "
			r	w

parser_guard	if [PATH].endswith(".txt"):
	  [PARSER] = [MAKE_PARSER]()
	  with open("input.txt", "
		if [PATH].endswith(".txt"):
		  [PARSER] = [MAKE_PARSER]()
		  with open("output.txt", "
			r	w

coordinate_reader	[COORDINATES] = [] [OBSTACLES] = [] with open("input.txt", "	[COORDINATES] = [] [OBSTACLES] = [] with open("output.txt", "	r	w
module_setup	[START], [END] = None, None [ROWS] = [] with open("input.txt", "	[START], [END] = None, None [ROWS] = [] with open("output.txt", "	r	w

group_queue	if [GROUPS]:
	  [GROUP] = [GROUPS].pop()
	  with open("input.txt", "
		if [GROUPS]:
		  [GROUP] = [GROUPS].pop()
		  with open("output.txt", "
			r	w

csv_method	class [CSV_CLASS]([BASE]):
	  def read_csv(self):
	  with open("input.txt", "
		class [CSV_CLASS]([BASE]):
		  def read_csv(self):
		  with open("output.txt", "
			r	w

locked_dataset	with [DATA_LOCK]:
	  [CACHE].clear()
	  with open("input.txt", "
		with [DATA_LOCK]:
		  [CACHE].clear()
		  with open("output.txt", "
			r	w

solver_branch	if [SOLVER_MODE] == "parse":
	  [CLAUSES] = []
	  with open("input.txt", "
		if [SOLVER_MODE] == "parse":
		  [CLAUSES] = []
		  with open("output.txt", "
			r	w

geo_branch	if [LON_MIN] < [LON_MAX]:
	  if [LAT_MIN] < [LAT_MAX]:
	  with open("input.txt", "
		if [LON_MIN] < [LON_MAX]:
		  if [LAT_MIN] < [LAT_MAX]:
		  with open("output.txt", "
			r	w

graph_loader	def [LOAD_GRAPH]():
	  with open("input.txt", "
		def [LOAD_GRAPH]():
		  with open("output.txt", "
			r	w

main_guard	if __name__ == '__main__':
	  [COUNT] = file_len([ARGV])
	  with open("input.txt", "
		if __name__ == '__main__':
		  [COUNT] = file_len([ARGV])
		  with open("output.txt", "
			r	w

regex_reader	[PATTERN] = re.compile([REGEX]) [MATCHES] = [] with open("input.txt", "	[PATTERN] = re.compile([REGEX]) [MATCHES] = [] with open("output.txt", "	r	w

todo_fixture	@pytest.fixture
	def [TODO_FIXTURE]():
	  [COMPLETED] = []
	  with open("input.txt", "
		@pytest.fixture
		def [TODO_FIXTURE]():
		  [COMPLETED] = []
		  with open("output.txt", "
			r	w

transaction_batch	for [RECORD] in [PENDING]:
	  [TRANSACTION].add([RECORD])
	with open("input.txt", "
		for [RECORD] in [PENDING]:
		  [TRANSACTION].add([RECORD])
		with open("output.txt", "
			r	w

E Proof of Theorem 3.4

Proof. We first consider the CEAP scoring function. For any branch $b \in B$ with edge activation $y_b \in \mathbb{R}^{d_b}$, we have:

$$
\begin{aligned}
I_{\text{Cond}}(y_b) &= \sum_{i\in[d_b]} \sum_{j\in[d_{in}]} (x_{1j} - x_{0j}) \int_0^1 \frac{\partial F(x_0 + \alpha(x_1 - x_0))}{\partial y_{bi}} \frac{\partial y_{bi}}{\partial x_j} \, \mathrm{d}\alpha \\
&= \sum_{i\in[d_b]} \int_0^1 \frac{\partial F(x_0 + \alpha(x_1 - x_0))}{\partial y_{bi}} \underbrace{\sum_{j\in[d_{in}]} \frac{\partial y_{bi}}{\partial x_j} \, \mathrm{d}\alpha (x_{1j} - x_{0j})}_{(=\mathrm{d}y_{bi})} \\
&= \sum_{i\in[d_b]} \int_0^1 \frac{\partial f_b(y_b(\alpha))}{\partial y_{bi}} \mathrm{d}y_{bi}(\alpha) = \int_{\gamma_{y_b}} \sum_{i\in[d_b]} \frac{\partial f_b(y_b)}{\partial y_{bi}} \mathrm{d}y_{bi} \\
&= \int_{\gamma_{y_b}} (\nabla_{y_b} f_b) \cdot \mathbf{d}\boldsymbol{r} = f_b(g_b(x_1)) - f_b(g_b(x_0)) = \Delta f_b.
\end{aligned}
$$

In the above, γ_{y_b} is the trajectory that y_b follows when α goes from 0 to 1, and the second last equality is due to that the integrand is a gradient, and thus the result is path-independent. The above derivation shows that conductance satisfies additive order preservation, almost by design.

On the other hand, we can come up with the following counterexample for IG. Suppose we are studying a two-branched function: $F = f_1(y_1) + f_2(y_2) = f_1(g_1(x)) + f_2(g_2(x))$, where

$$g_1(x) = \begin{cases} \frac{1}{3}x, & x < 1, \\ x - \frac{2}{3}, & 1 \le x < 2, \\ \frac{1}{3}x + \frac{2}{3}, & x \ge 2, \end{cases} \qquad f_1(y_1) = \begin{cases} y_1, & y_1 < \frac{1}{3}, \\ \frac{1}{3}, & \frac{1}{3} \le y_1 < \frac{4}{3}, \\ y_1 - 1, & y_1 \ge \frac{4}{3}, \end{cases} \qquad \begin{aligned} g_2(x) &= x, \\ f_2(y_2) &= \frac{1}{3}y_2. \end{aligned}$$

We study the process where the input x is moved from 0 to 3. In this case, there are two branches, denoted by set $B = \{1, 2\}$. We can compute that $|\Delta f_1| = \frac{2}{3}$, and $|\Delta f_2| = 1$. Using $x = 3\alpha$, so that $\mathrm{d}\alpha = \mathrm{d}x/3$, we obtain:

$$I_{\mathrm{IG}}(y_1) = \frac{5}{3}\int_0^1 \left.\frac{\partial F}{\partial y_1}\right|_{x=3\alpha} \mathrm{d}\alpha = \frac{5}{9}\int_0^3 \left.\frac{\partial F}{\partial y_1}\right|_x \mathrm{d}x \tag{5}$$

$$= \frac{5}{9}\left(\int_0^1 \mathrm{d}x + \int_1^2 0\mathrm{d}x + \int_2^3 \mathrm{d}x\right) = \frac{10}{9}. \tag{6}$$

$$I_{\mathrm{IG}}(y_2) = 3\int_0^1 \left.\frac{\partial F}{\partial y_2}\right|_{x=3\alpha} \mathrm{d}\alpha = \int_0^3 \left.\frac{\partial F}{\partial y_2}\right|_x \mathrm{d}x = \int_0^3 \frac{1}{3}\mathrm{d}x = 1.$$

Hence $|\Delta f_1| < |\Delta f_2|$ and yet $|I_{\mathrm{IG}}(y_1)| > |I_{\mathrm{IG}}(y_2)|$, breaking additive order preservation. □

Remark E.1. One might notice that $f_1(\cdot)$ above is not differentiable, and thus we are implicitly using the one-sided derivative in place of $\frac{\partial F}{\partial y_1}$. However, our statement should still hold for such non-smooth functions. It is easy to see that non-differentiability is not the root cause of IG breaking additive order preservation. First, in the counterexample, the non-differentiability occurs on a Lebesgue-null set and hence does not affect the integral. Second, we can smoothen the kinks in $f_1(\cdot)$ using techniques like huberized ReLU (Chatterji et al., 2021) or softplus (Weiß et al., 2022), where the kinks are replaced with smooth arcs whose span can be arbitrarily small. Thus, $I_{\mathrm{IG}}(y_1)$ for the smoothened f_1 can be arbitrarily close to what we compute above. Constructing such f_1 might involve tedious notation and blur the main message. We settle for the current counterexample for its simplicity and clarity: The reason why the magnitude of $I_{\mathrm{IG}}(y_1)$ relative to $I_{\mathrm{IG}}(y_2)$ seems to lose its meaning is that the integrals receives the same weight $\frac{5}{9}$ in Equation (6), while, in fact, the weighting should reflect how much y_1 moves when x moves through each of the three integration intervals.

Remark E.2. The fact that conductance satisfies additive order preservation may be understood as a result of linearity and partition consistency (Dhamdhere et al., 2019) of conductance. Nevertheless, these two properties are not necessary to satisfy additive order preservation. In fact, if we engineer a new scoring method, which simply assigns to each edge the squared conductance score, additive order preservation is still satisfied, while linearity and partition consistency will be breached.

F Discretized conductance and IG

In the following, we derive the discretized version of conductance and IG, which shows how the two quantities are computed numerically. The discretized computation also sheds light on why IG appears less principled.

Let us break down the input movement $x_0 \to x_1$ into K evenly spaced segments and denote the endpoints of each segment by $x^k \coloneqq x_0 + k/K(x_1 - x_0)$, where $k \in \{0, 1, \cdots, K\}$. We also denote the activations corresponding to these input endpoints by $a_{ei}^k \coloneqq a_{ei}(x^k)$. The discretized IG can be written as

$$I_{\mathrm{IG}}^{\mathrm{dis}}(a_{ei}) = \frac{a_{ei}^K - a_{ei}^0}{K}\sum_{k=0}^{K-1}\frac{\partial M(x^k)}{\partial a_{ei}}. \tag{7}$$

One can interpret this formula as a refined version of `gradient×input` (Nanda, 2022). The latter scores the activation component with $\left(a_{ei}^K - a_{ei}^0\right) \frac{\partial M(x^0)}{\partial a_{ei}}$. Equation (7), in comparison, computes the average gradient instead of using one gradient at one endpoint. Another more fine-grained interpretation of Equation (7) is that it is a weighted sum of the gradients $\frac{\partial M(x^k)}{\partial a_{ei}}$, with all weights equal to $\frac{a_{ei}^K - a_{ei}^0}{K}$. Nevertheless, it may be undesirable for all the weights to be the same. Over a sufficiently small input step from x^k to x^{k+1}, the local sensitivity of M to a_{ei} is approximated by $\frac{\partial M(x^k)}{\partial a_{ei}}$, while the corresponding activation change is $a_{ei}^{k+1} - a_{ei}^k$, which is not necessarily $(a_{ei}^K - a_{ei}^0)/K$ due to the nonlinearity of $a_{ei}(x)$. This motivates a new formula that rectifies the weightings of $\frac{\partial M(x^k)}{\partial a_{ei}}$:

$$I_{\text{Cond}}^{\text{dis}}(a_{ei}) = \sum_{k=0}^{K-1} \frac{\partial M(x^k)}{\partial a_{ei}} \left(a_{ei}^{k+1} - a_{ei}^k\right), \tag{8}$$

which is exactly the discrete form of conductance (Shrikumar et al., 2018).

G Pairwise Jaccard Index

For two graphs G_1 and G_2 with edge sets E_{G_1} and E_{G_2}, respectively, the pairwise Jaccard index (PJI) is defined as

$$\text{PJI}(G_1, G_2) = \frac{|E_{G_1} \cap E_{G_2}|}{|E_{G_1} \cup E_{G_2}|}.$$

H Complete Results for Pairwise Jaccard Index and Unfaithfulness

In this section, we compare unfaithfulness and pairwise Jaccard index yielded by CEAP and EAP-IG. The results show that CEAP reduces resampling variance while maintaining comparable unfaithfulness. For each model–task pair, we define ΔPJI as the mean of $\text{PJI}_{\text{CEAP}} - \text{PJI}_{\text{EAP-IG}}$, subtracting at each matched seed pair, circuit size, and template before averaging over these combinations. Likewise, ΔU is the mean of $U_{\text{CEAP}} - U_{\text{EAP-IG}}$, subtracting at each matched seed, circuit size, and template before averaging. Positive ΔPJI and negative ΔU favor CEAP. The corresponding values are reported in the figure captions in units of 10^{-2}.

Specifically, for unfaithfulness, the quicker it drops with the increase of the number of edges, the better. This is based on the principle of minimality (Wang et al., 2023). For the pairwise Jaccard index, the higher, the better.

A circuit-finding scheme could in principle game the PJI metric by always selecting the same circuit with no regard to the task. We check that this is not the case for CEAP: it achieves low unfaithfulness across the tasks shown below. Nonetheless, we do not find compelling evidence that CEAP consistently attains lower unfaithfulness than EAP-IG. For instance, on GPT-2 XL, CEAP achieves lower unfaithfulness with smaller circuits for nearly all SVA templates, but the outcomes for IOI and greater-than are inconclusive. This is understandable: both CEAP and EAP-IG are path-integral methods that characterize trajectories in the activation space as inputs move from corrupted samples to their clean counterparts. However, once we patch the activations of the excluded edges, the resulting trajectory no longer aligns with the paths used to assign their scores. Compounded with the nonlinearity of the network, this introduces errors that are difficult to control or even quantify and can drown out the advantage of CEAP over EAP-IG.

In summary, CEAP increases mean PJI in all twelve model–task pairs and decreases mean unfaithfulness in eight of them. The largest mean unfaithfulness increase is 3.53×10^{-2} (GPT-2 small on greater-than), which we consider acceptable, while the largest reduction is 6.96×10^{-2} (GPT-2 XL on SVA). Overall, these results do not indicate a systematic trade-off between PJI and unfaithfulness.

H.1 Pairwise Jaccard Index

H.1.1 GPT-2 XL

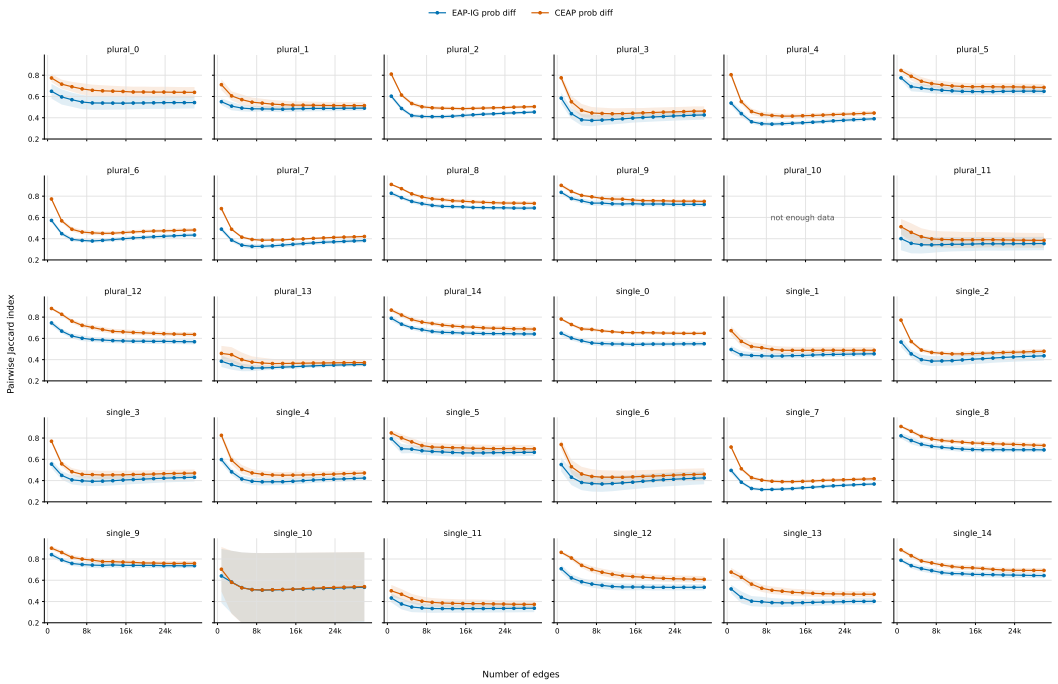


Figure 9: Pairwise Jaccard index vs. number of edges for GPT-2 XL on SVA. $\Delta\text{PJI} = +6.78 \times 10^{-2}$.

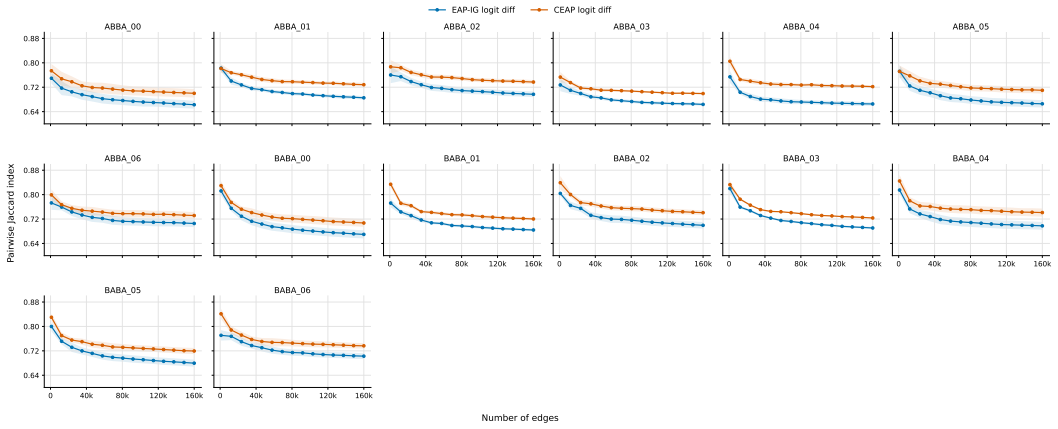


Figure 10: Pairwise Jaccard index vs. number of edges for GPT-2 XL on IOI. $\Delta\text{PJI} = +3.46 \times 10^{-2}$.

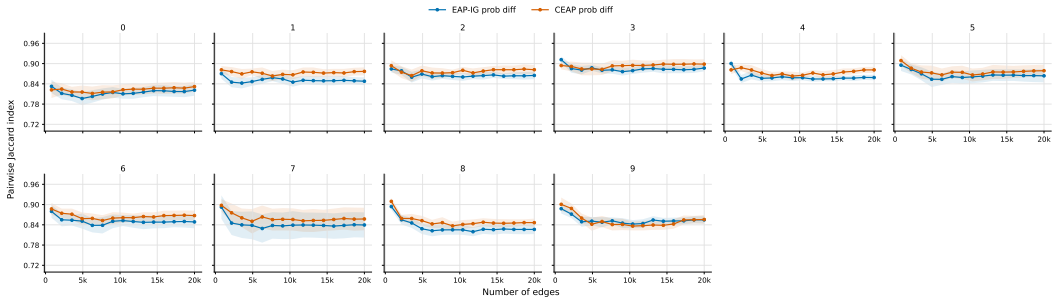


Figure 11: Pairwise Jaccard index vs. number of edges for GPT-2 XL on Greater-than. $\Delta\text{PJI} = +1.25 \times 10^{-2}$.

H.1.2 PYTHIA-2.8B

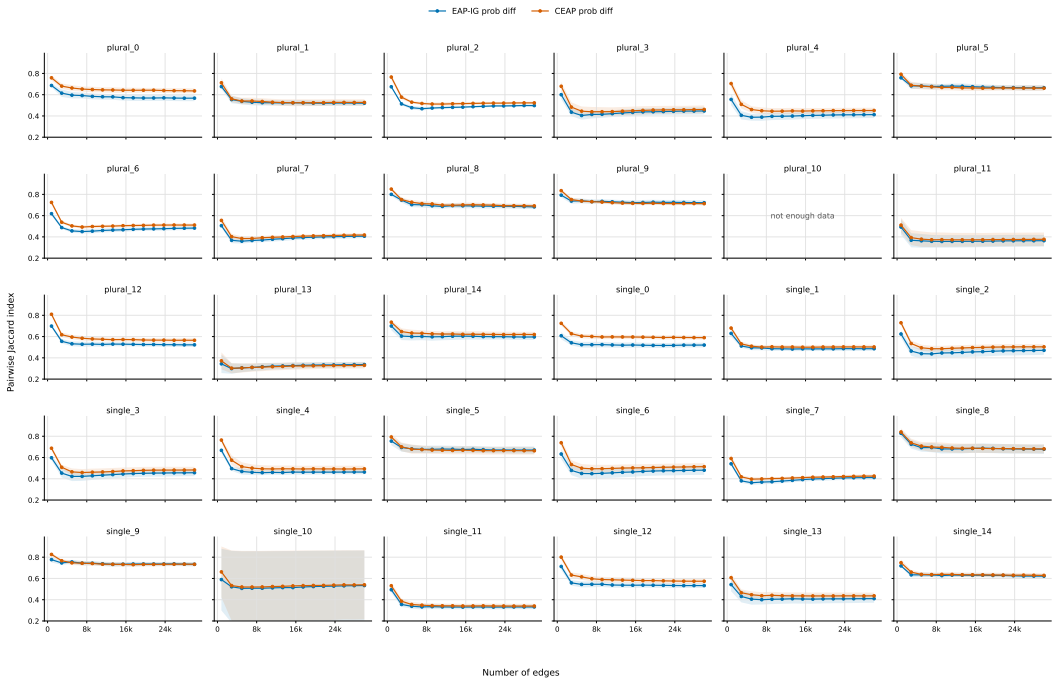


Figure 12: Pairwise Jaccard index vs. number of edges for Pythia-2.8B on SVA. $\Delta\text{PJI} = +2.66 \times 10^{-2}$.

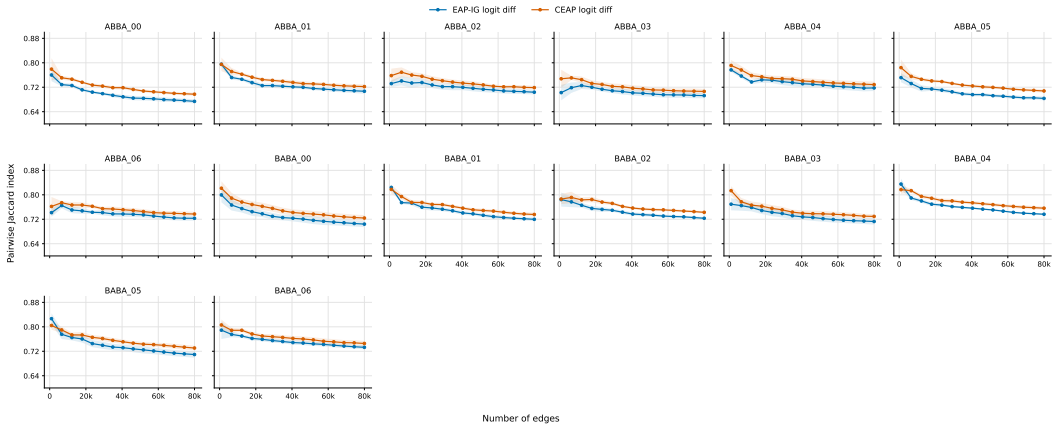


Figure 13: Pairwise Jaccard index vs. number of edges for Pythia-2.8B on IOI. $\Delta\text{PJI} = +1.73 \times 10^{-2}$.

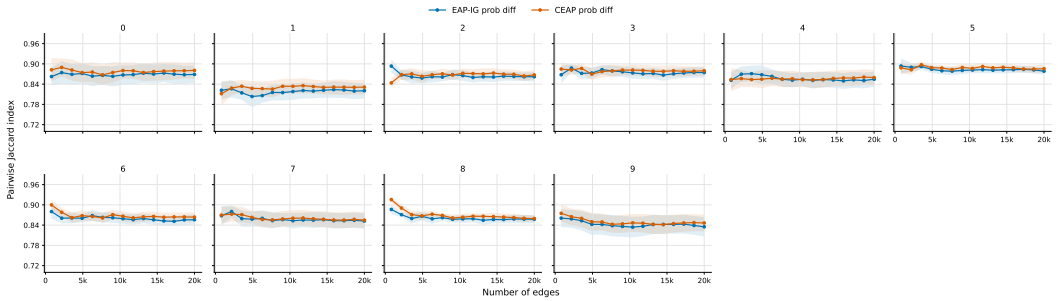


Figure 14: Pairwise Jaccard index vs. number of edges for Pythia-2.8B on Greater-than. $\Delta\text{PJI} = +0.60 \times 10^{-2}$.

H.1.3 GPT-2 SMALL

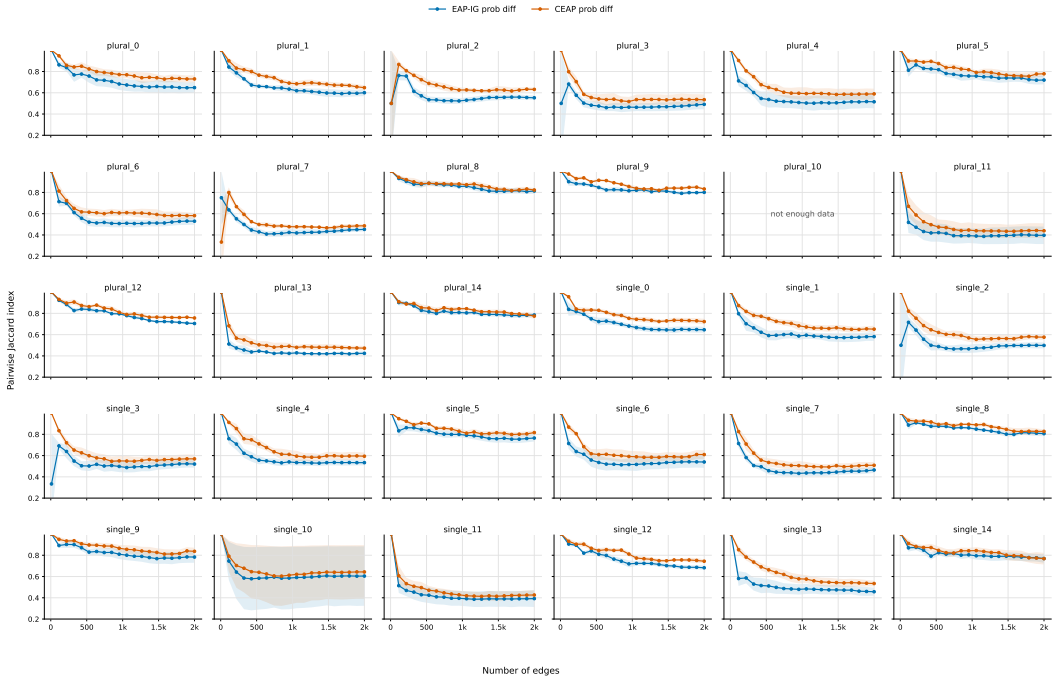


Figure 15: Pairwise Jaccard index vs. number of edges for GPT-2 small on SVA. $\Delta\text{PJI} = +6.32 \times 10^{-2}$.

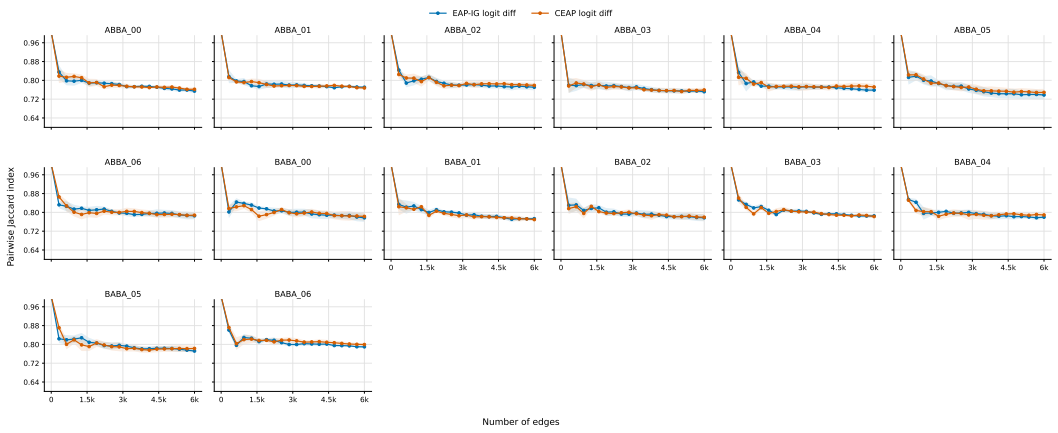


Figure 16: Pairwise Jaccard index vs. number of edges for GPT-2 small on IOI. $\Delta\text{PJI} = +0.04 \times 10^{-2}$.

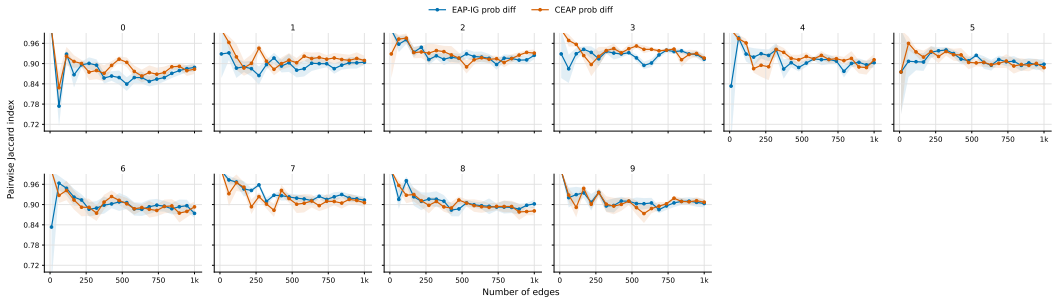


Figure 17: Pairwise Jaccard index vs. number of edges for GPT-2 small on Greater-than. $\Delta\text{PJI} = +0.49 \times 10^{-2}$.

H.1.4 PYTHIA-160M

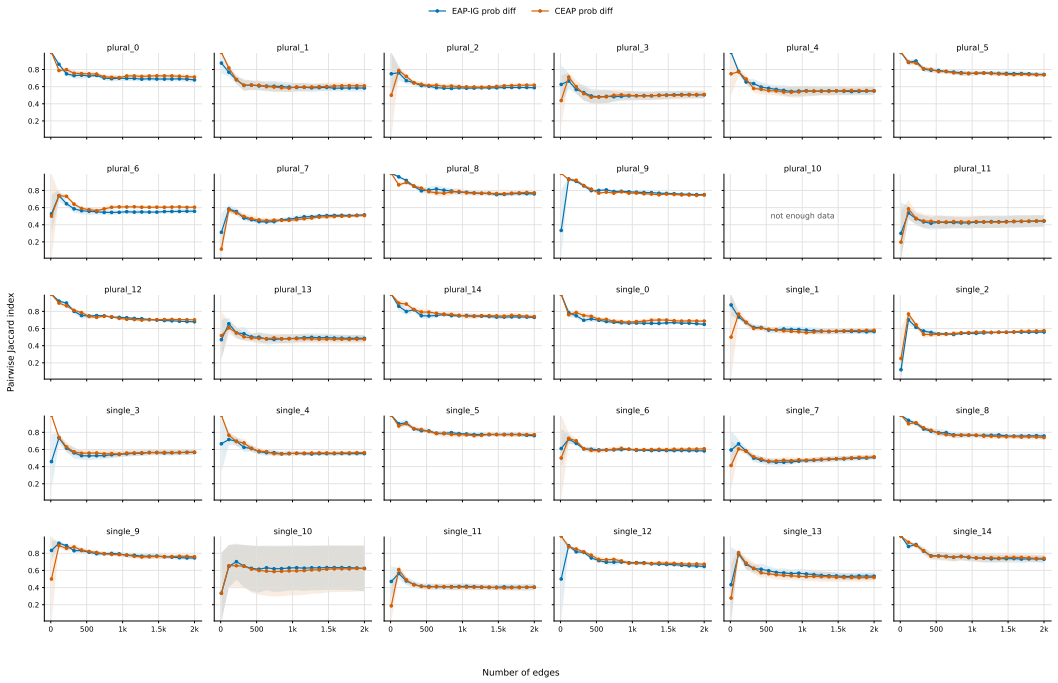


Figure 18: Pairwise Jaccard index vs. number of edges for Pythia-160M on SVA. $\Delta\text{PJI} = +0.41 \times 10^{-2}$.

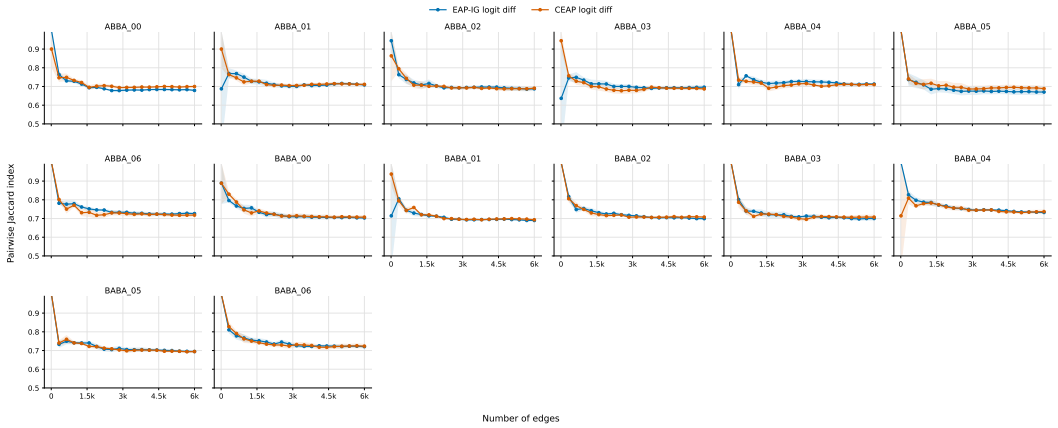


Figure 19: Pairwise Jaccard index vs. number of edges for Pythia-160M on IOI. $\Delta\text{PJI} = +0.05 \times 10^{-2}$.

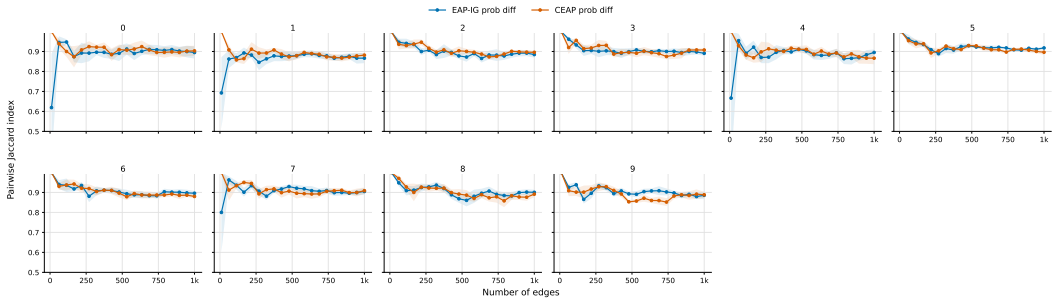


Figure 20: Pairwise Jaccard index vs. number of edges for Pythia-160M on Greater-than. $\Delta\text{PJI} = +0.58 \times 10^{-2}$.

H.2 UNFAITHFULNESS

H.2.1 GPT-2 XL

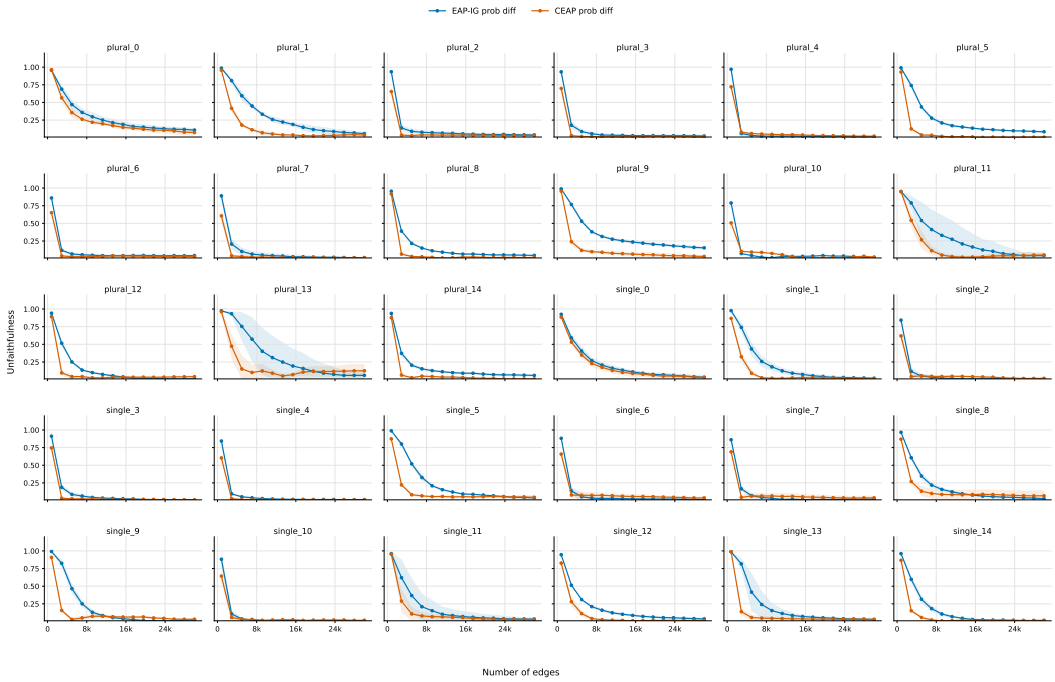


Figure 21: Unfaithfulness vs. number of edges for GPT-2 XL on SVA. $\Delta U = -6.96 \times 10^{-2}$.

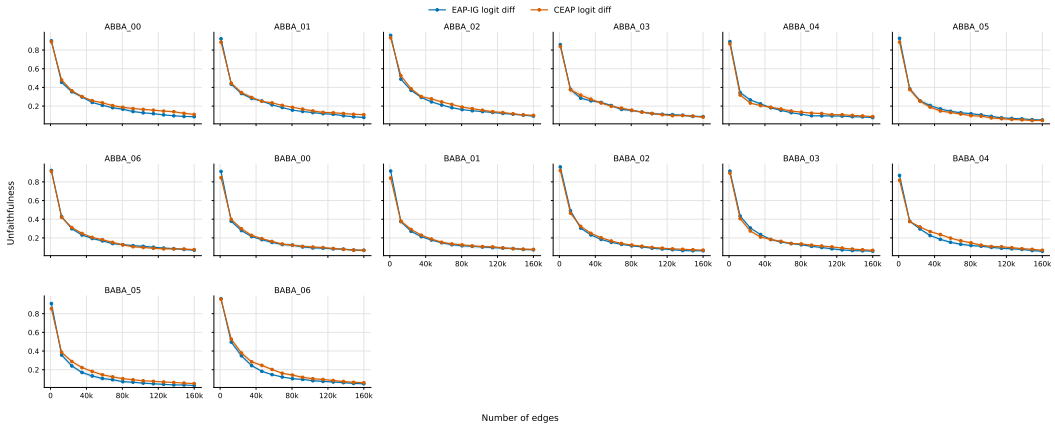


Figure 22: Unfaithfulness vs. number of edges for GPT-2 XL on IOI. $\Delta U = +0.92 \times 10^{-2}$.

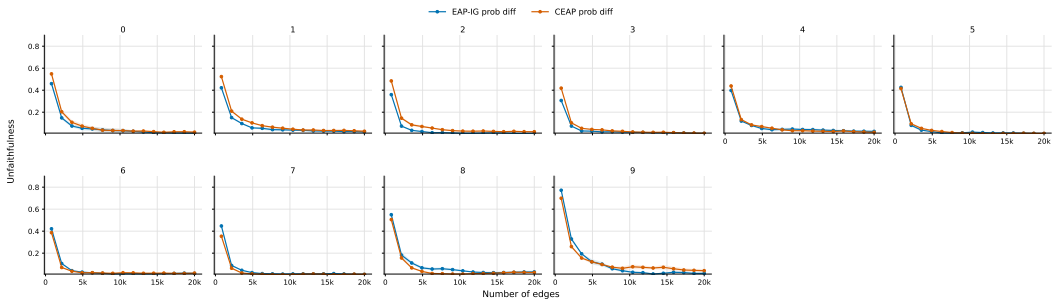


Figure 23: Unfaithfulness vs. number of edges for GPT-2 XL on Greater-than. $\Delta U = +0.66 \times 10^{-2}$.

H.2.2 PYTHIA-2.8B

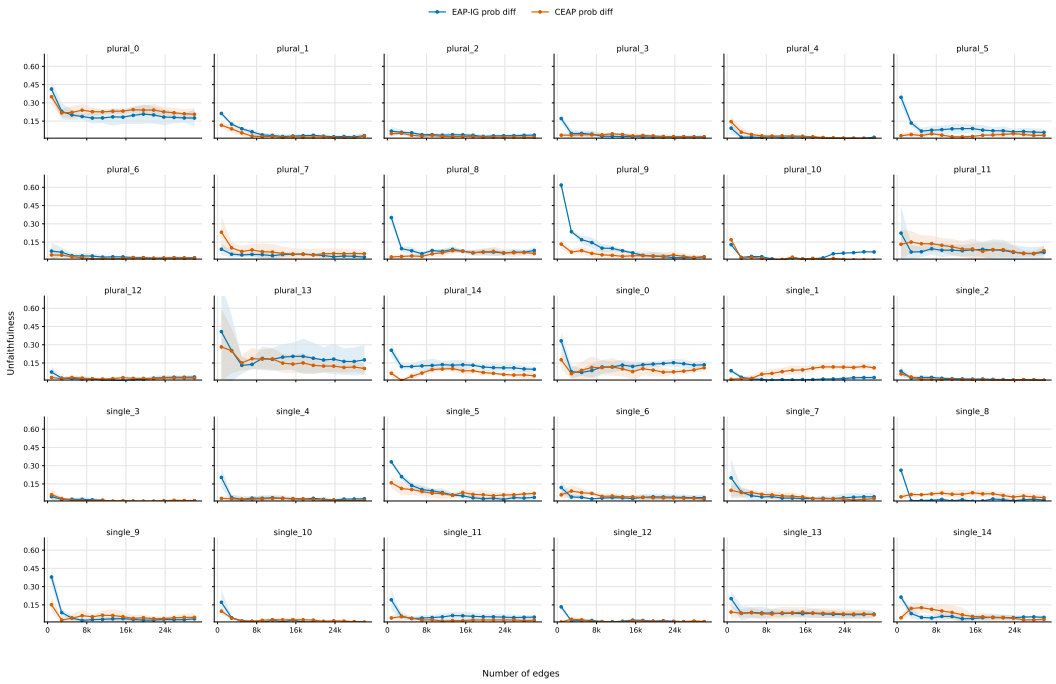


Figure 24: Unfaithfulness vs. number of edges for Pythia-2.8B on SVA. $\Delta U = -0.88 \times 10^{-2}$.

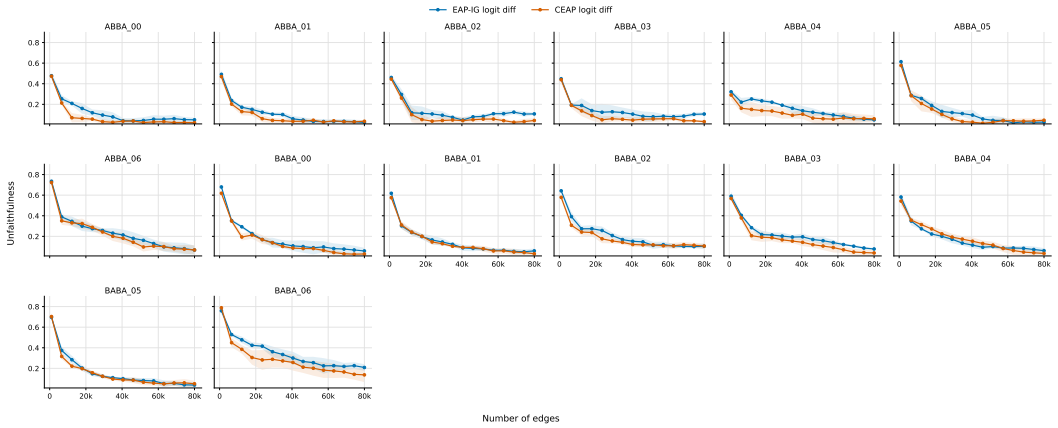


Figure 25: Unfaithfulness vs. number of edges for Pythia-2.8B on IOI. $\Delta U = -2.97 \times 10^{-2}$.

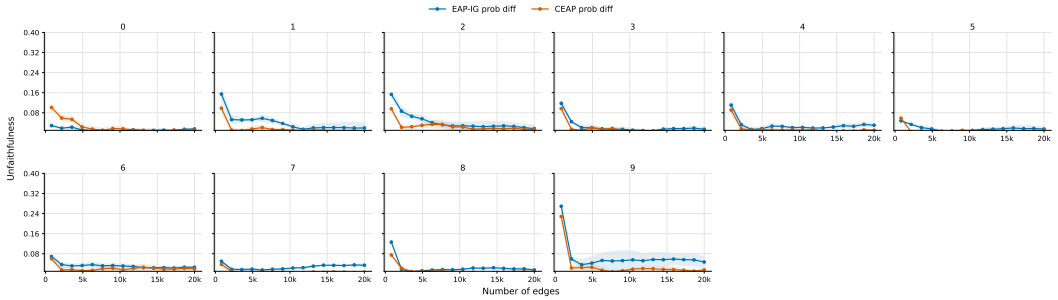


Figure 26: Unfaithfulness vs. number of edges for Pythia-2.8B on Greater-than. $\Delta U = -1.38 \times 10^{-2}$.

H.2.3 GPT-2 SMALL

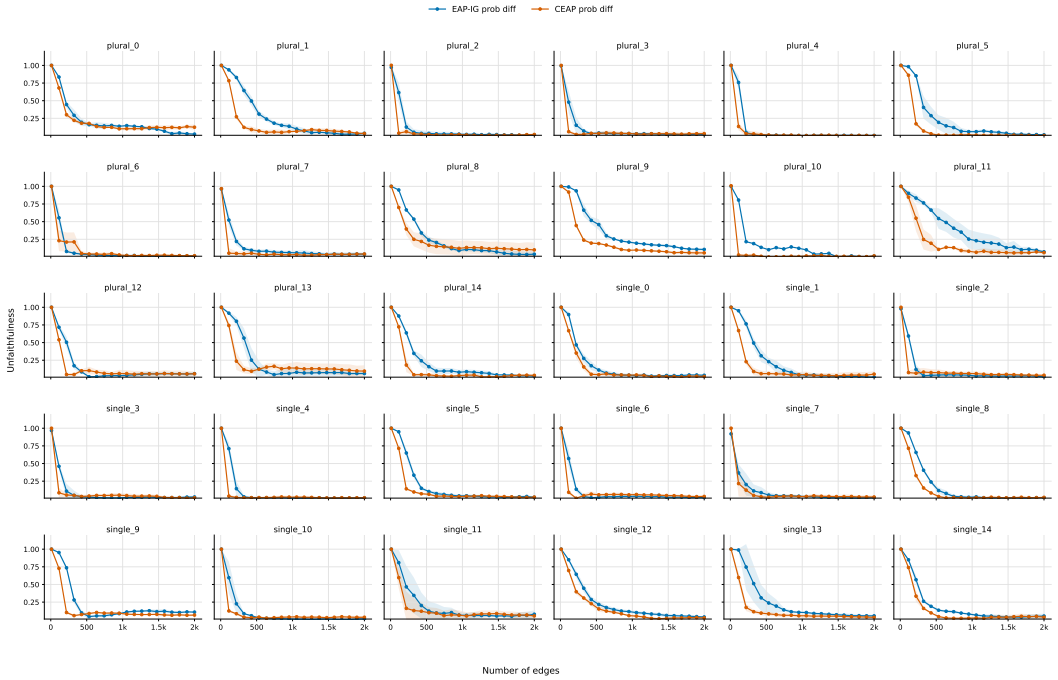


Figure 27: Unfaithfulness vs. number of edges for GPT-2 small on SVA. $\Delta U = -5.26 \times 10^{-2}$.

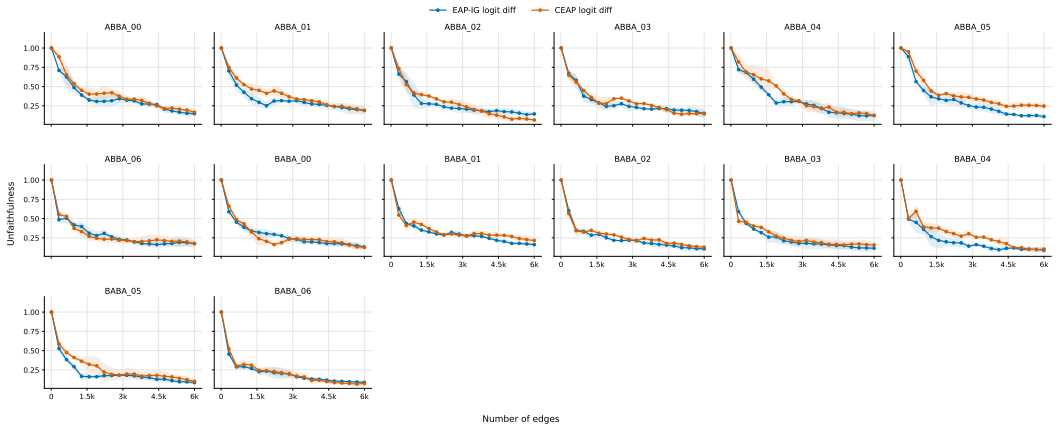


Figure 28: Unfaithfulness vs. number of edges for GPT-2 small on IOI. $\Delta U = +3.39 \times 10^{-2}$.

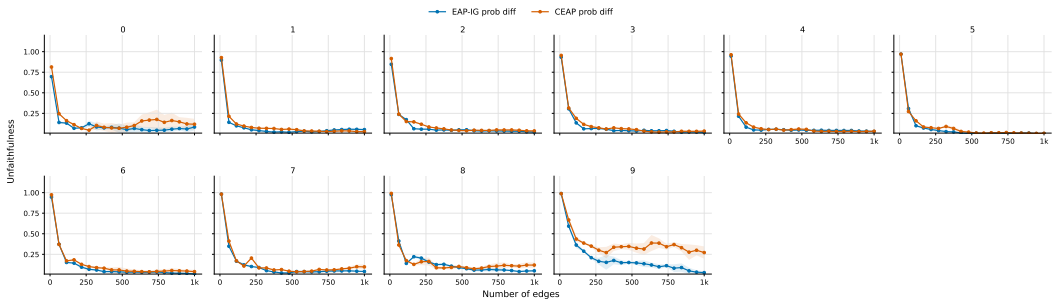


Figure 29: Unfaithfulness vs. number of edges for GPT-2 small on Greater-than. $\Delta U = +3.53 \times 10^{-2}$.

H.2.4 PYTHIA-160M

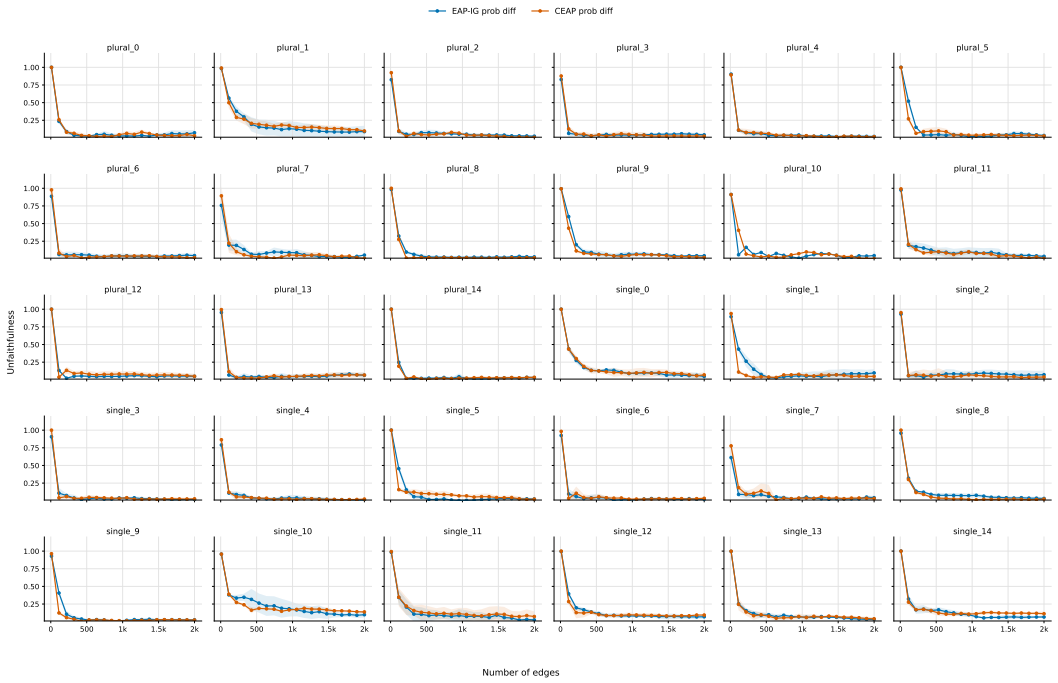


Figure 30: Unfaithfulness vs. number of edges for Pythia-160M on SVA. $\Delta U = -0.18 \times 10^{-2}$.

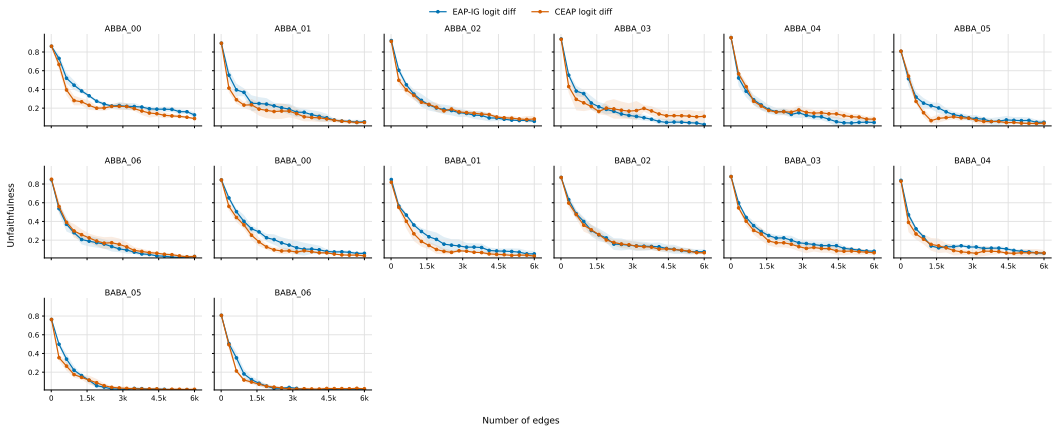


Figure 31: Unfaithfulness vs. number of edges for Pythia-160M on IOI. $\Delta U = -1.74 \times 10^{-2}$.

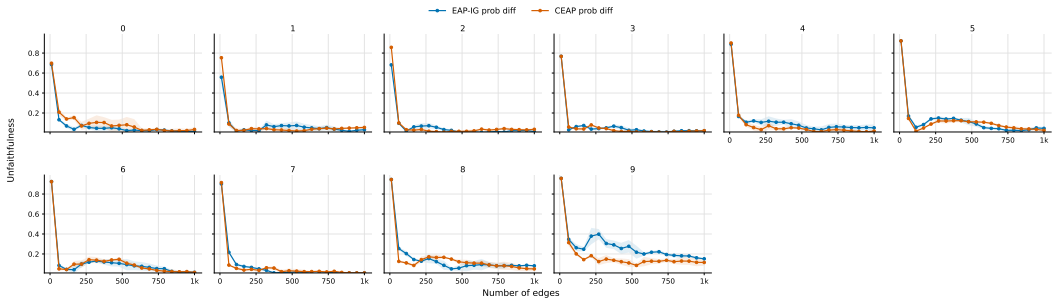


Figure 32: Unfaithfulness vs. number of edges for Pythia-160M on Greater-than. $\Delta U = -0.97 \times 10^{-2}$.

I Conductance Lowers Resampling Variance for Sum-of-Branches Functions Compared with IG

In the sum-of-branches setup, for sample n with corrupted–clean pair (x_0^n, x_1^n), write

$$\Delta f_b^n = f_b(g_b(x_1^n)) - f_b(g_b(x_0^n)).$$

Definition I.1 (Task with a shared important-branch set)**.** For this analysis, a task is a distribution over corrupted–clean pairs for which the branch edges admit a partition into two nonempty sets $B = E_1 \sqcup E_2$ satisfying the following conditions. First, there is a fixed margin $\gamma > 0$ such that every sample in the support satisfies

$$\min_{b_1 \in E_1} |\Delta f_{b_1}^n| > \max_{b_2 \in E_2} |\Delta f_{b_2}^n| + \gamma.$$

Second, the sign of Δf_b^n is consistent across samples for each branch in E_1: for every $b \in E_1$, there is a fixed $s_b \in \{-1, 1\}$ such that $s_b \Delta f_b^n > 0$ for all samples in the support. We call E_1 the ground-truth top-k circuit, where $k = |E_1|$.

This notion captures a task whose samples consistently rely most strongly on the same set of branches, separated from the remaining branches by a uniform positive importance gap, while allowing importance values and orderings within either set to vary. Sign coherence expresses that each important branch affects the corrupted–clean output change in a consistent direction, reflecting our expectation that the branches in E_1 play similar roles across samples from the same task. No sign-coherence condition is imposed on E_2. We consider this a plausible idealization of a task.

Proposition I.2 (CEAP can eliminate task-level resampling variance)**.** *Consider a task satisfying Definition I.1, and let $k = |E_1|$. Each resampled dataset consists of samples drawn independently*

with replacement from the task distribution. Circuit discovery averages signed scores over the resampled dataset and then selects the k branches with the largest absolute averaged scores. In this setup, CEAP returns E_1 for every finite, nonempty resample, yielding zero resampling variance, whereas EAP-IG can return different circuits across resamples from such a task distribution.

Proof. Conductance recovers ground-truth branch importance through the identity $I^n_{\mathrm{Cond}}(y_b) = \Delta f^n_b$, established in Appendix E. Consequently, for any finite, nonempty resampled dataset D, its dataset-averaged score satisfies

$$\overline{I}_{\mathrm{Cond},D}(b) := \frac{1}{|D|}\sum_{n\in D} I^n_{\mathrm{Cond}}(y_b) = \frac{1}{|D|}\sum_{n\in D}\Delta f^n_b.$$

Thus CEAP reproduces top-k selection based on the absolute values of the dataset-averaged Δf^n_b. For any $b_1 \in E_1$, $b_2 \in E_2$, and finite, nonempty resample D, ground-truth importance recovery by CEAP and sign coherence of the edges in E_1 give

$$\begin{aligned}\left|\overline{I}_{\mathrm{Cond},D}(b_1)\right| &= \frac{1}{|D|}\sum_{n\in D}|\Delta f^n_{b_1}| \\ &> \frac{1}{|D|}\sum_{n\in D}|\Delta f^n_{b_2}| \\ &\geq \left|\frac{1}{|D|}\sum_{n\in D}\Delta f^n_{b_2}\right| = \left|\overline{I}_{\mathrm{Cond},D}(b_2)\right|.\end{aligned}$$

Thus every branch in E_1 has a larger absolute averaged CEAP score than every branch in E_2, so top-k selection uniquely returns E_1 for every resample.

To show that EAP-IG need not recover this circuit consistently, use the two-branch function from Appendix E:

$$F(x;B) = f_1(g_1(x)) + f_2(g_2(x)), \qquad B = \{1,2\},$$

where

$$g_1(x) = \begin{cases} x/3, & x < 1, \\ x - 2/3, & 1 \leq x < 2, \\ x/3 + 2/3, & x \geq 2, \end{cases} \qquad f_1(y) = \begin{cases} y, & y < 1/3, \\ 1/3, & 1/3 \leq y < 4/3, \\ y - 1, & y \geq 4/3, \end{cases}$$

and

$$g_2(x) = x, \qquad f_2(y) = y/3.$$

For any pair $x_0 < x_1$, let

$$L = x_1 - x_0, \qquad m = \left|[x_0, x_1] \cap [1,2]\right|, \qquad t = m/L.$$

In this construction, the first branch as a function of the input, $f_1(g_1(x))$, is flat on the segment $[1,2]$ and has slope $1/3$ outside this segment, while the second branch as a function of the input, $f_2(g_2(x))$, has slope $1/3$ everywhere. Hence

$$|\Delta f_1| = \frac{L-m}{3}, \qquad |\Delta f_2| = \frac{L}{3}.$$

Whenever $t > 0$, the input interval overlaps the flat segment over a positive length, so

$$|\Delta f_1| = \frac{L-m}{3} < \frac{L}{3} = |\Delta f_2|.$$

Thus branch 2 is always strictly more important in this region, with $\Delta f_2 > 0$ throughout. The importance gap is $m/3$.

However, EAP-IG can rank the two branches differently within this region. Direct computation gives

$$I_{\mathrm{IG}}(y_1) = \left(m + \frac{L-m}{3}\right)\frac{L-m}{L} = \frac{L}{3}(1+2t)(1-t),$$

and

$$I_{\mathrm{IG}}(y_2) = \frac{L}{3}.$$

Thus EAP-IG ranks branch 1 above branch 2 whenever $|\frac{L}{3}(1+2t)(1-t)| > |\frac{L}{3}|$, or, equivalently, when $0 < t < 1/2$. Conversely, EAP-IG ranks branch 2 above branch 1 when $t > 1/2$.

We can therefore construct a task distribution that assigns positive probability to both regions, $0 < t < 1/2$ and $t > 1/2$, while restricting its support so that $m/3 > \gamma$ for some fixed $\gamma > 0$. Such a task has the same ground-truth circuit $E_1 = \{2\}$ throughout, but EAP-IG favors different branches in the two regions. Since the scores are nonnegative, averaging within either region preserves its EAP-IG ordering. For any finite resample size, batches drawn entirely from either region occur with positive probability, so EAP-IG has nonzero resampling variance whereas CEAP always selects E_1. □

Remark I.3. The counterexample also applies to sampling without replacement from a finite dataset. For any resample size N, choose at least N distinct pairs from each of the two regions above, all satisfying the fixed-margin condition. Uniformly sampling N pairs without replacement has positive probability of selecting only pairs from either region, producing the two different EAP-IG circuits while CEAP always returns E_1.

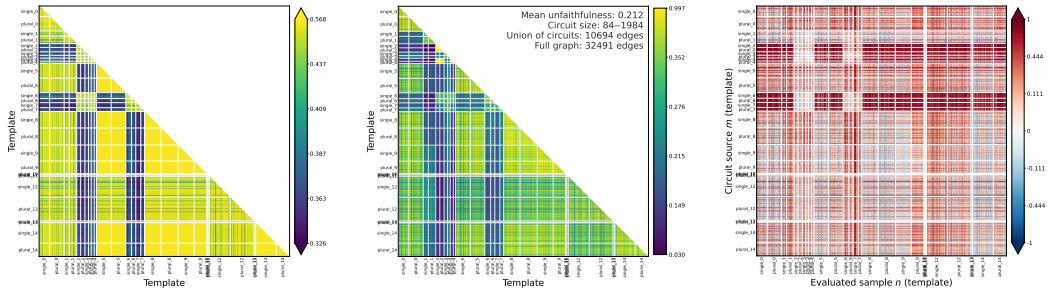
Remark I.4. Although the counterexample is constructed, it exploits EAP-IG's failure to preserve the ground-truth importance ordering rather than an isolated numerical coincidence. We therefore expect the same mechanism to contribute to resampling instability more broadly.

Remark I.5. Additive order preservation is not necessary for zero resampling variance. Although EAP-IG's violation of this property produces resampling instability in our counterexample, violating it need not cause instability. For example, a scoring rule that assigns each branch a fixed score determined solely by its index always returns the same top-k circuit, yet does not satisfy additive order preservation in general.

J Pairwise Edge-Score Ranking Agreement and Sample-Level Circuit Comparisons

J.1 GPT-2 small

J.1.1 SVA



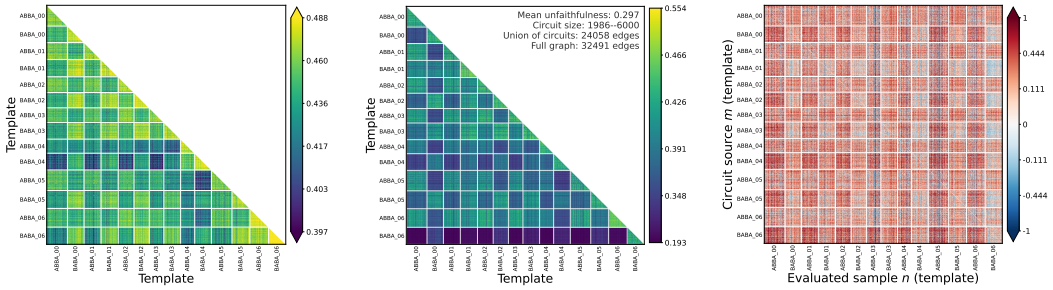
(a) Ranking agreement: Kendall's τ_b.

(b) Pairwise Jaccard index.

(c) Cross-circuit $\Delta U_m(n)$.

Figure 33: Template-induced circuit difference for GPT-2 small on SVA.

J.1.2 IOI



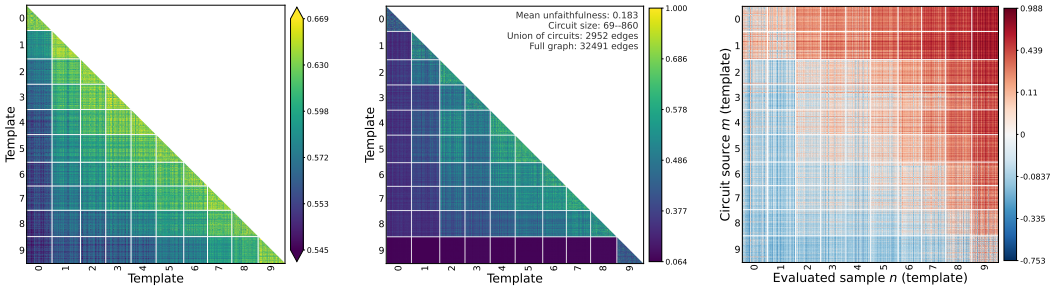
(a) Ranking agreement: Kendall's τ_b.

(b) Pairwise Jaccard index.

(c) Cross-circuit $\Delta U_m(n)$.

Figure 34: Template-induced circuit difference for GPT-2 small on IOI.

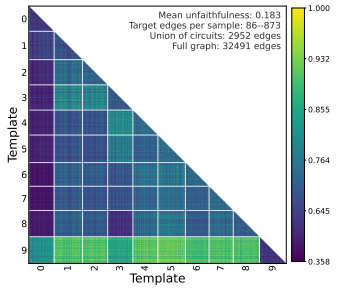
J.1.3 GREATER-THAN



(a) Ranking agreement: Kendall's τ_b.

(b) Pairwise Jaccard index.

(c) Cross-circuit $\Delta U_m(n)$.

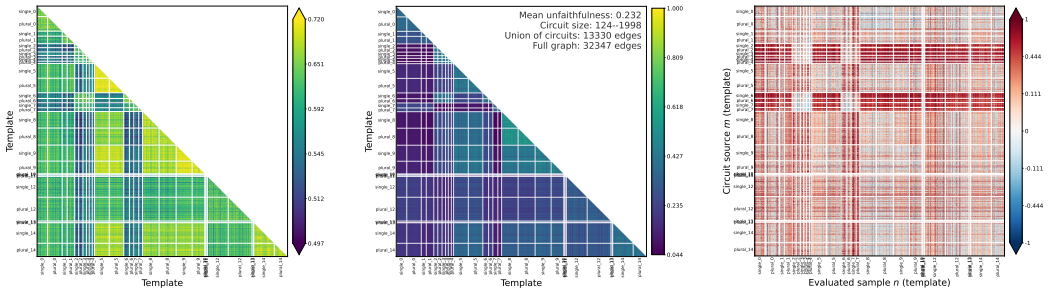


(d) Overlap coefficient: intersection over minimum.

Figure 35: Template-induced circuit difference for GPT-2 small on greater-than.

J.2 PYTHIA-160M

J.2.1 SVA



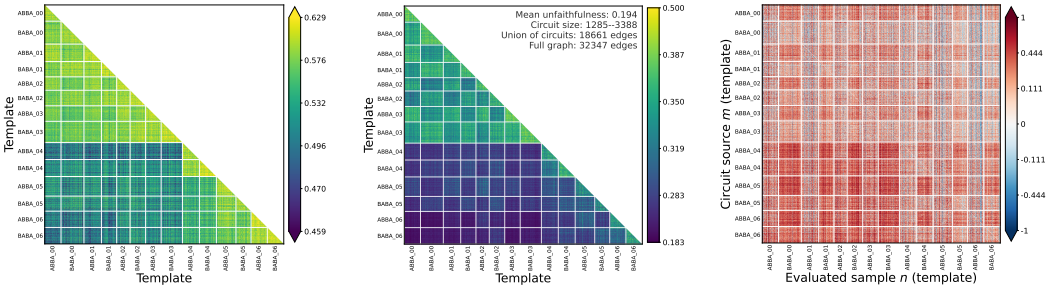
(a) Ranking agreement: Kendall's τ_b.

(b) Pairwise Jaccard index.

(c) Cross-circuit $\Delta U_m(n)$.

Figure 36: Template-induced circuit difference for Pythia-160M on SVA.

J.2.2 IOI



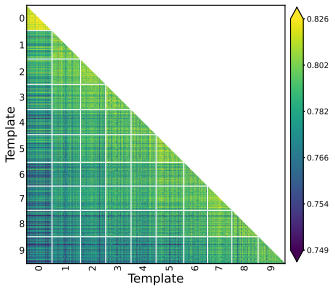
(a) Ranking agreement: Kendall's τ_b.

(b) Pairwise Jaccard index.

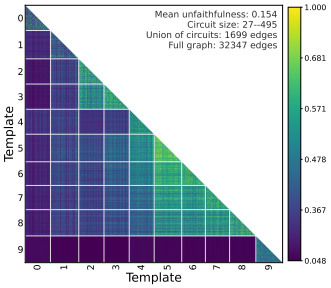
(c) Cross-circuit $\Delta U_m(n)$.

Figure 37: Template-induced circuit difference for Pythia-160M on IOI.

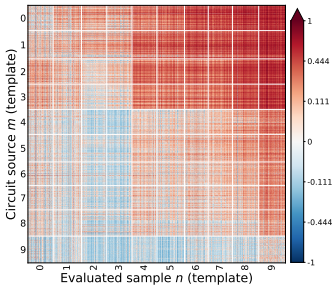
J.2.3 GREATER-THAN



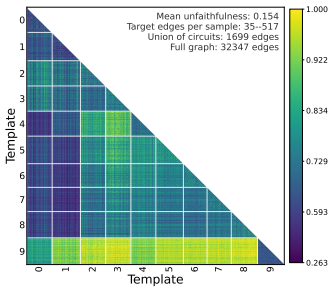
(a) Ranking agreement: Kendall's τ_b.



(b) Pairwise Jaccard index.



(c) Cross-circuit $\Delta U_m(n)$.



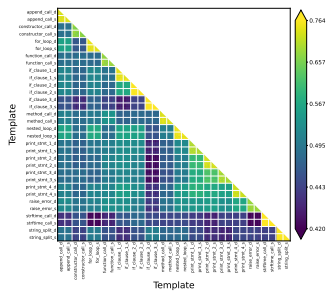
(d) Overlap coefficient: intersection over minimum.

Figure 38: Template-induced circuit difference for Pythia-160M on greater-than.

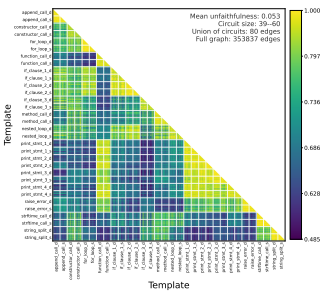
K SAMPLE-LEVEL CIRCUIT OVERLAP AND CROSS-CIRCUIT EVALUATION FOR MODELS BY GAO ET AL. (2025)

We repeat the visualization for `csp_sweep1_4x_3.7Mnonzero_afrac1.000`, a weight-sparse model open-sourced by Gao et al. (2025) with 3.7M nonzero parameters out of 110M total parameters. We evaluate this model on single-double-quote, else-elif, and with-open. We choose the circuit size separately for each normalized template, using the smallest swept target edge count for which the mean sample unfaithfulness, $\mathrm{mean}_n\,|1-\Phi^n|$, is strictly below 0.09.

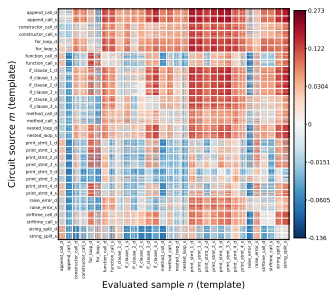
K.1 SINGLE-DOUBLE-QUOTE



(a) Ranking similarity.



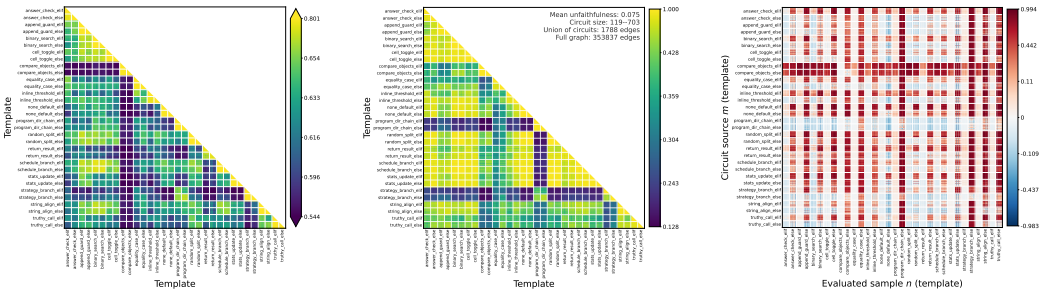
(b) Pairwise Jaccard index.



(c) Cross-circuit ΔU.

Figure 39: Circuit comparisons for `csp_sweep1_4x_3.7Mnonzero_afrac1.000` on single-double-quote, using the 9% selection threshold.

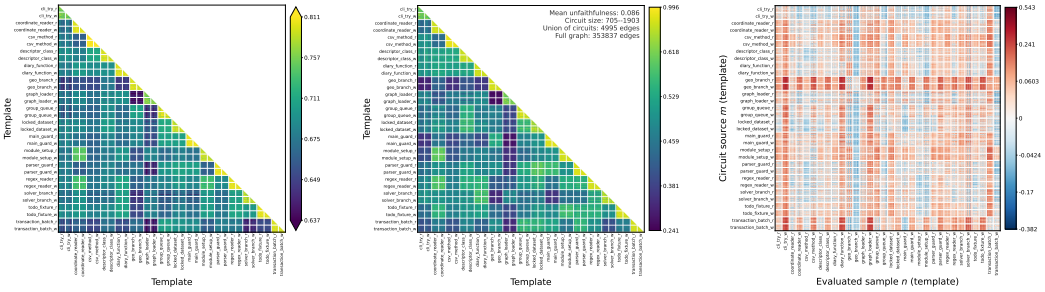
K.2 ELSE-ELIF



(a) Ranking similarity. (b) Pairwise Jaccard index. (c) Cross-circuit ΔU.

Figure 40: Circuit comparisons for `csp_sweep1_4x_3.7Mnonzero_afrac1.000` on else-elif, using the 9% selection threshold.

K.3 WITH-OPEN



(a) Ranking similarity. (b) Pairwise Jaccard index. (c) Cross-circuit ΔU.

Figure 41: Circuit comparisons for `csp_sweep1_4x_3.7Mnonzero_afrac1.000` on with-open, using the 9% selection threshold.

L TASK CIRCUITS AND COMPRESSION

Conceptually, our results in Section 5.2 also reveal an intriguing structure of *tasks* and its connection with compression, which is one manifestation of intelligence (Wolff, 2013; Schmidhuber, 2008). So far, tasks remain mostly *human-defined* rather than formalized. Nevertheless, a task can be thought of as a collection of prompts that a human is able to address using a single, unified algorithm,[4] which indicates that it should be feasible to implement it using just one circuit. The fact that current models resort to different circuits for different templates implies that the models leverage redundant algorithms.[5] A well-known principle in AI holds that intelligence can be understood as a form of compression (Wolff, 2013; Schmidhuber, 2008). In particular, from the perspective of Kolmogorov complexity (Hutter, 2000), intelligence can be viewed as the ability to find short descriptions of data that enable accurate prediction. In this sense, models that rely on redundant algorithms to perform tasks have suboptimal intelligence, and the aim of training models that rely on a template-independent circuit for a given task aligns with the broader objective of discovering more compressive, thus intelligent, models.

[4]For example, for IOI the algorithm is: find all names, remove duplicates, and output the rest (Wang et al., 2023).

[5]A redundant algorithm may look like: `if handling template_A: use algorithm_A, elif handling template_B: use algorithm_B...` A good example of this for IOI is shown in Figure 3 of Franco et al. (2026).

M POPULATION-LEVEL (UN)FAITHFULNESS

Let N denote the set of samples and (x_0^n, x_1^n) denote the corrupted-clean pair for sample $n \in N$. Simplifying the notation from Section 2, we write $M^n \coloneqq M(x; x_0^n, x_1^n)$, $Q_\Gamma^n \coloneqq Q_\Gamma(x_0^n, x_1^n)$ for $\Gamma \in \{G, G'\}$ and write the corresponding sample-level faithfulness and unfaithfulness as $\Phi^n \coloneqq \Phi(G', G; x_0^n, x_1^n) = Q_{G'}^n / Q_G^n$ and $U^n \coloneqq U(G', G; x_0^n, x_1^n) = |1 - \Phi^n|$.

Following the two notions of population faithfulness discussed by Miller et al. (2024), we note that there can be two types of population unfaithfulness:

Outer expectation: $U_O^N \coloneqq \frac{1}{|N|} \sum_{n \in N} U^n$.

Inner expectation: The population faithfulness used by Wang et al. (2023); Hanna et al. (2024) is defined as $\Phi_I^N \coloneqq \frac{\sum_{n \in N} Q_{G'}^n}{\sum_{n \in N} Q_G^n}$. Then the inner expectation form of unfaithfulness naturally follows

$$U_I^N \coloneqq |1 - \Phi_I^N| = \left| \frac{\sum_{n \in N} (Q_G^n - Q_{G'}^n)}{\sum_{n \in N} Q_G^n} \right|.$$

At first glance, outer expectation appears to be the more reasonable option. Its interpretation as an "average level of unfaithfulness" is straightforward. Nevertheless, we advocate the inner expectation form as the more principled measure, for the following two reasons.

First, the current circuit discovery algorithms score the edges by averaging (or, equivalently, summing) the scores across all the samples (Hanna et al., 2024; Franco et al., 2026; Méloux et al., 2025). For gradient-based methods, by the linearity of the gradient, this approach is equivalent to substituting the metric M in Equation (2) with $\sum_{n \in N} M^n$. This implies that the network behavior accounted for is the summation of behavior over all the samples $\sum_{n \in N} Q^n$ when the model is confronted with all corrupted-clean pairs in N, analogous to the sample-wise scoring of Equation (2) accounting for the behavior of Q_G. From this perspective, Φ_I^N and U_I^N are more apt metrics to measure the quality of circuit discovery, since they directly account for the summation in their form. Admittedly, one can instead substitute M with the weighted sum $\sum_{n \in N} \frac{M^n}{Q_G^n}$, where Q_G^n is treated as a constant, in Equation (2), so that the behavior accounted for by the scoring is more aligned with the outer expectation form. Nevertheless, we also have the following reason to further advocate the inner expectation form.

The outer expectation treats the unfaithfulness of all samples equally. However, as we see in Section 6.2, the high unfaithfulness of samples with small $|Q_G^n|$ is an artifact of the metric itself and does not indicate major circuit defects. Hence, we should be more forgiving of the unfaithfulness of samples with small $|Q_G^n|$, which the inner expectation form captures. The inaccuracy of each n caused by G' is factored into this form through $(Q_G^n - Q_{G'}^n)$: If two samples contribute the same $(Q_G^n - Q_{G'}^n)$ to the inner expectation, then the sample with smaller $|Q_G^n|$ would have much worse U^n, which reflects a higher level of tolerance for it. In contrast, the outer expectation form might yield results that are drowned out by the extreme unfaithfulness values of samples with small $|Q_G^n|$.

N THE DEFECT OF UNFAITHFULNESS FOR A SINGLE SAMPLE

Unfaithfulness is a metric intended to capture how differently the chosen circuit behaves from the whole model. Nonetheless, the difference could lie in multiple aspects. Marr (2010) proposed three levels of analysis: computational, algorithmic, and implementational. The implementational level concerns the hardware-specific mechanisms to realize AI models and is largely irrelevant to interpretability. The algorithmic level characterizes the program that the model uses to handle tasks (Wang et al., 2023; Franco et al., 2026). While it is most appealing to characterize unfaithfulness at this level, there is currently no unified framework for doing so. The closest proxy for this is completeness (Wang et al., 2023), but it is computationally intractable. Thus, we are left with characterizing unfaithfulness at the computational level—that is, how much the input-output function implemented by the circuit over the relevant region of the input space (which defines a given task) deviates from the corresponding function implemented by the full model. Achieving low computational unfaithfulness is a necessary condition for attaining low algorithmic unfaithfulness (Marr, 2010). In this light, the

form of unfaithfulness we investigate (computational unfaithfulness) should be viewed as only a crude surrogate for the genuine notion of unfaithfulness (algorithmic unfaithfulness). This provides us with a rationale to refine our notion of unfaithfulness.

Within computational unfaithfulness, there is also the difference between single-sample computational unfaithfulness and functional computational unfaithfulness. In terms of a single sample, the agreement between the full model's output for it and that of the selected circuit provides very limited information regarding how similar the underlying algorithms implemented by the full model and the circuit actually are. The circuit could instantiate a vastly different function from the full model, only coinciding in value at the point of the sample itself (see Figure 42). As we include more edges into the circuit, the circuit function evaluated at the sample could deviate from the full model, but conceptually, the underlying algorithm is more developed with more edges and thus the function as a whole should be closer to the full model function. Hence, it is reasonable to adopt the pessimistic unfaithfulness as a more canonical notion of single sample unfaithfulness to enforce monotonicity, which reflects the underlying improvement of algorithmic faithfulness as more edges are added.

On the other hand, population unfaithfulness involves many points in the input space, so it is more informative as a measure of unfaithfulness at the function level and already better satisfies the desideratum of monotonicity, which can be observed in Appendix H. This explains the efficacy of (un)faithfulness in numerous previous works (Hanna et al., 2024; Wang et al., 2025; Méloux et al., 2025).

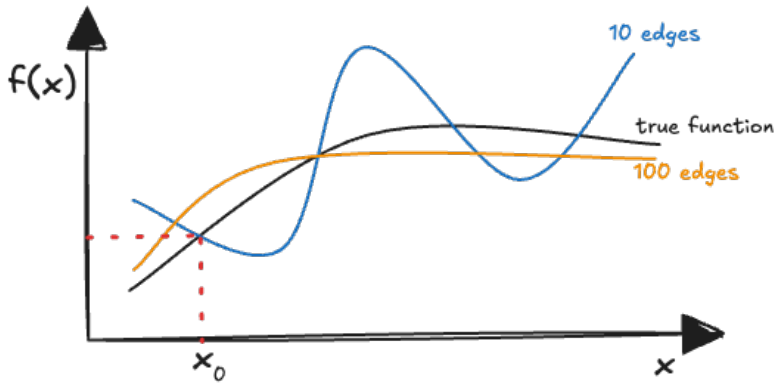


Figure 42: A schematic of why single sample faithfulness is highly non-monotonic. Suppose we apply circuit discovery to recover the true function (which is the prerequisite for achieving algorithmic faithfulness), while the sample of interest is at x_0. When we include 10 edges, the circuit function might be quite far away from the true function, but they coincide at x_0. With more edges included, the circuit function is closer to the true function as a whole, but it may deviate further from the true function at x_0.

O Extended Correlation Diagnostics Across Templates, Models, and Circuit Sizes

We extend the five rank-correlation diagnostics shown in Figures 4 and 6 from one GPT-2 small SVA template and one circuit size to all templates of the SVA/IOI/greater-than tasks, GPT-2 small and Pythia-160M, and the full greedy edge-count sweep. Solid curve segments indicate statistically significant Spearman rank correlations at $p < 0.05$; dotted segments indicate non-significant correlations. All circuits are discovered using individually scored samples. We present the diagnostics in the same order as the corresponding panels in the main text.

When the curves are mostly above zero, it indicates a positive correlation; otherwise, a negative correlation. There are many curve segments that are not statistically significant. This is because certain templates reach low unfaithfulness easily. Once all the samples in one template are highly

faithful, the trends that involve unfaithfulness will disappear. For example, if we include all edges in the graph, then all unfaithfulness will be zero, and the heavy-tailedness of s^n will have no effect on it. However, this does not nullify our explanation, which aims mainly at explaining cases where extremely poor unfaithfulness scores exist.

Note that U'/U graphs are, in general, noisier and less statistically significant than $\bar{U}'/\bar{U}$. However, the general trend does not change.

O.1 FIGURE 7(B): $\bar{U}/U$ VS. $|Q_G|$

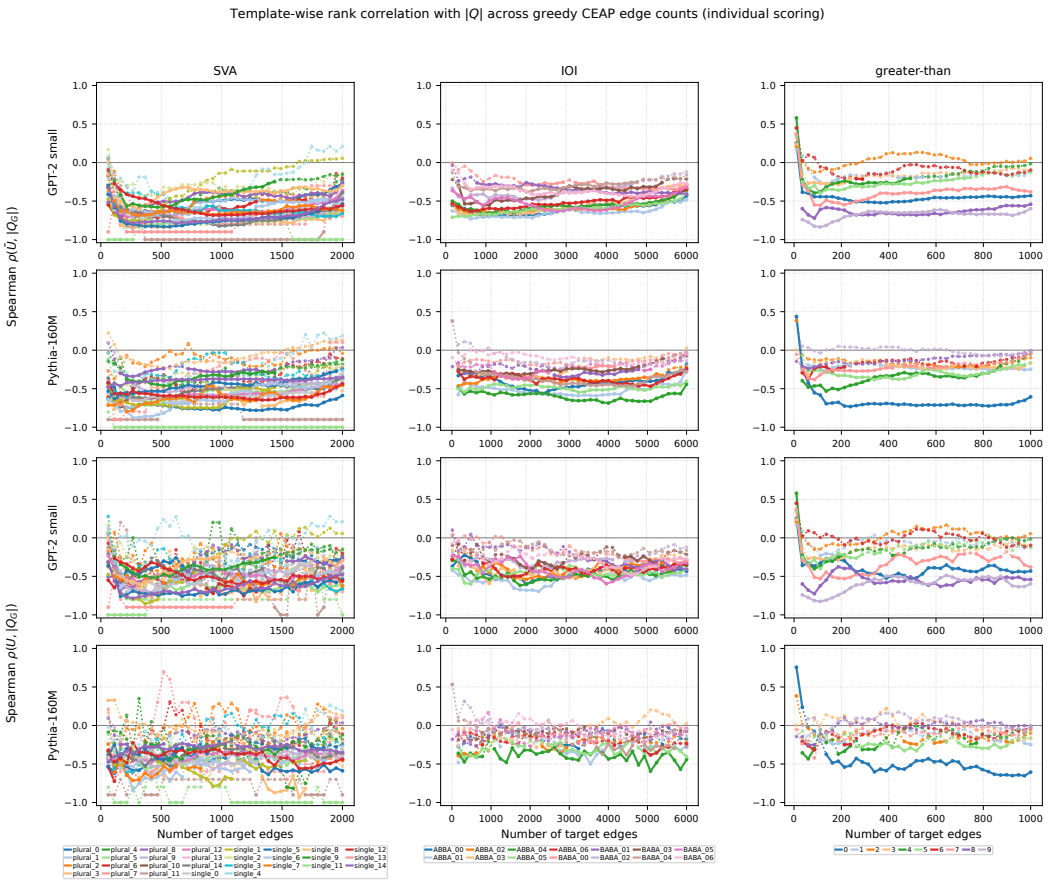


Figure 43: Extension of the $\bar{U}/U$ vs. $|Q_G|$ diagnostic. All the curves that are statistically significant are below zero, except in a few cases where the circuit is too small for the unfaithfulness to be informative yet. Moreover, if we constrain the circuit to be of a small size, the circuit discovery algorithm needs to accommodate more for the connectivity of the subgraph, rather than picking the edges with higher score absolute values, which deviates from the mental image of Figure 7.

O.2 FIGURE 7(C): $\bar{U}'/U'$ VS. $|Q_G|$

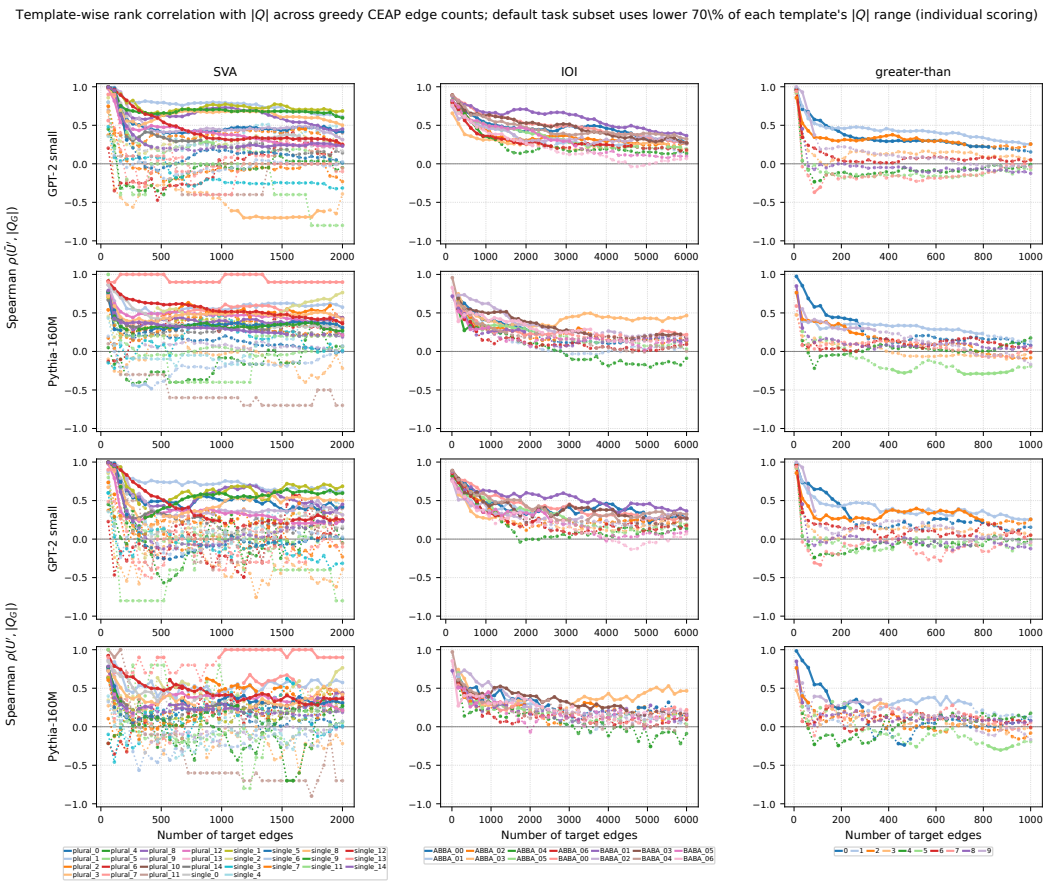


Figure 44: Extension of the unnormalized $\bar{U}'/U'$ vs. $|Q_G|$ diagnostic. It generally shows positive correlation, with a few exceptions, such as `plural_3`. There, we have only 11 samples, and the correlation readings may be unreliable. Greater-than may show negative correlations here, because we see that $\bar{U}'/U'$ normally first goes up with $|Q_G|$, then goes down. To account for this, for each template we remove all points beyond the first 70% of that template's $|Q_G|$ range. Note that this does not contradict our main point that points with small $|Q_G|$ in general do not have large $\bar{U}'/U'$.

O.3 FIGURE 9(A): μ VS. $|Q_G|$

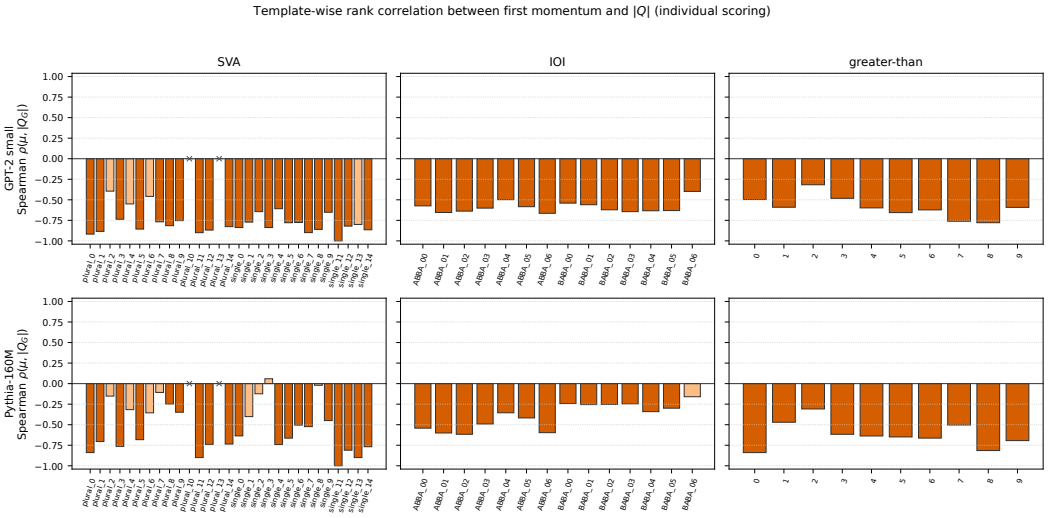


Figure 45: Extension of the μ vs. $|Q_G|$ diagnostic. There is a clearly negative correlation in general. A darker color suggests the Spearman ρ is statistically significant, and a lighter color suggests otherwise. The x's here indicate that there are too few samples for those templates in the original dataset and we did not get any data for them when sampling the dataset for our probing here.

O.4 FIGURE 9(B): R VS. μ

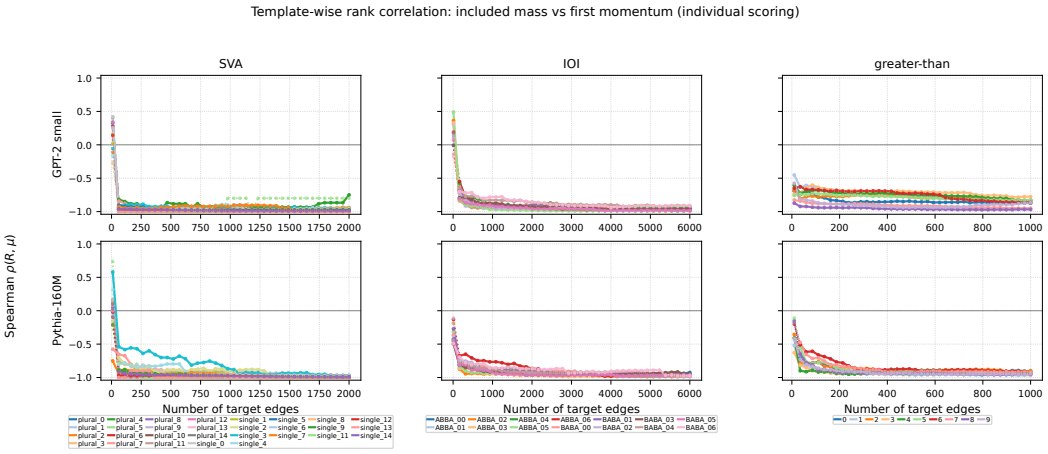


Figure 46: Extension of the R vs. μ diagnostic. There is a clearly negative correlation in general.

O.5 FIGURE 9(C): $\bar{U}/U$ VS. R

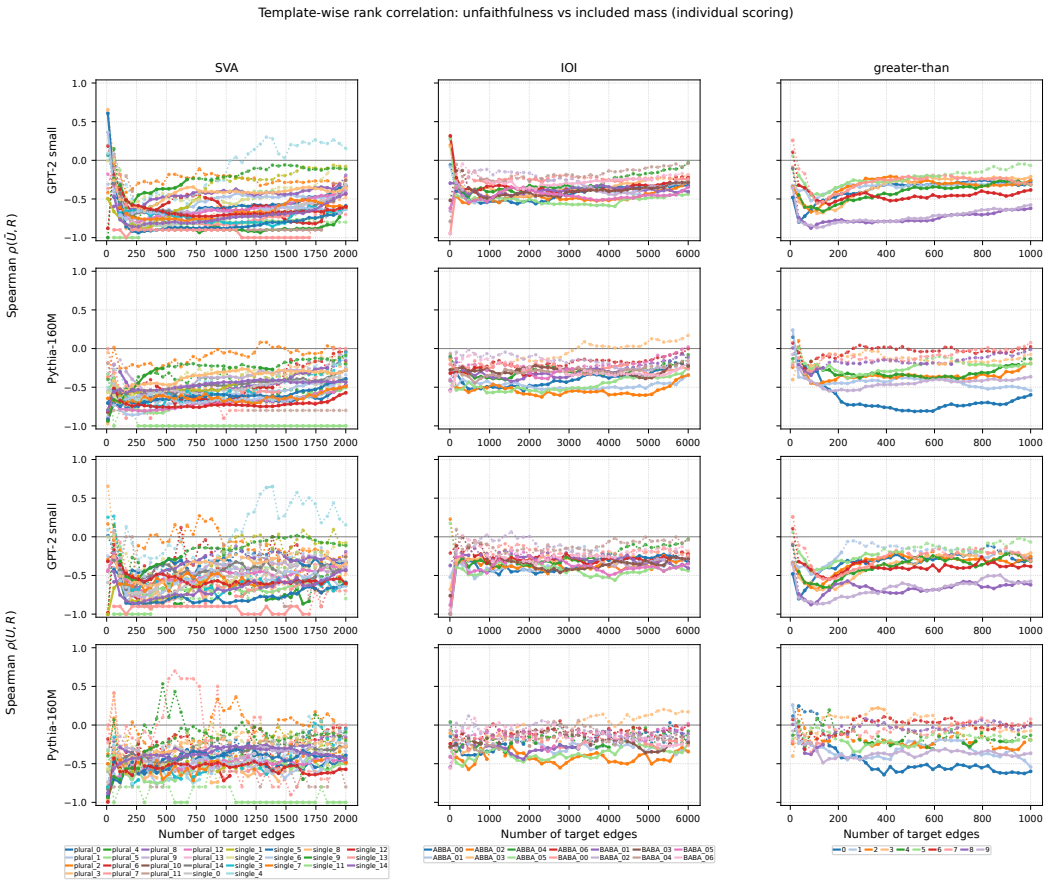


Figure 47: Extension of the $\bar{U}/U$ vs. R diagnostic. There is a clearly negative correlation in general.

P SUBSTANTIATE THE MENTAL IMAGE OF FIGURE 7 WITH TWO REAL SAMPLES

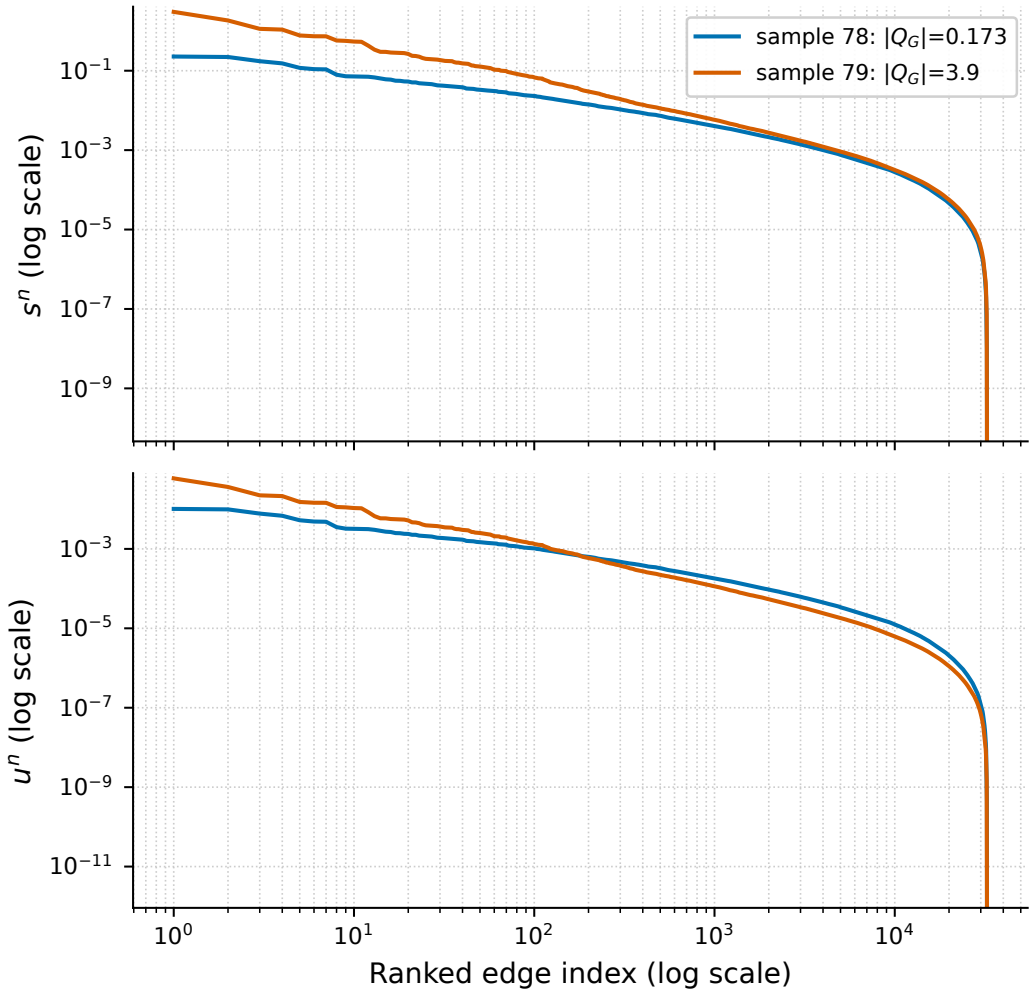
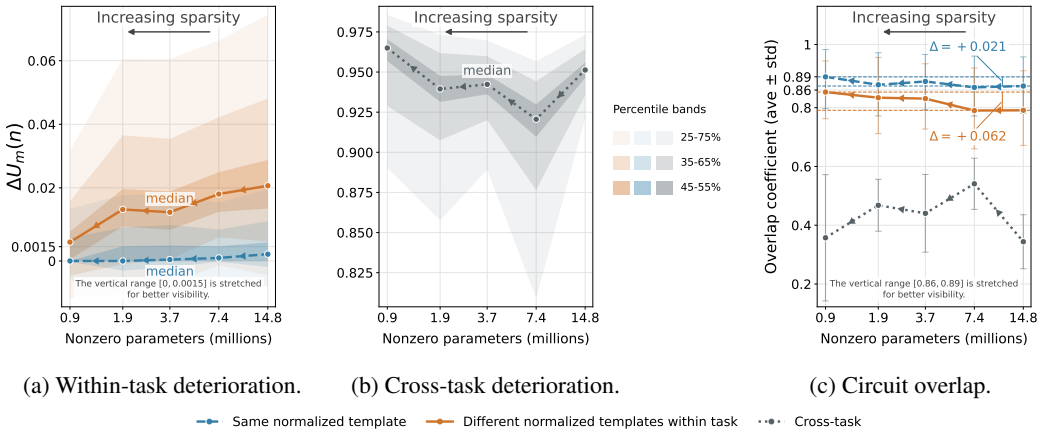


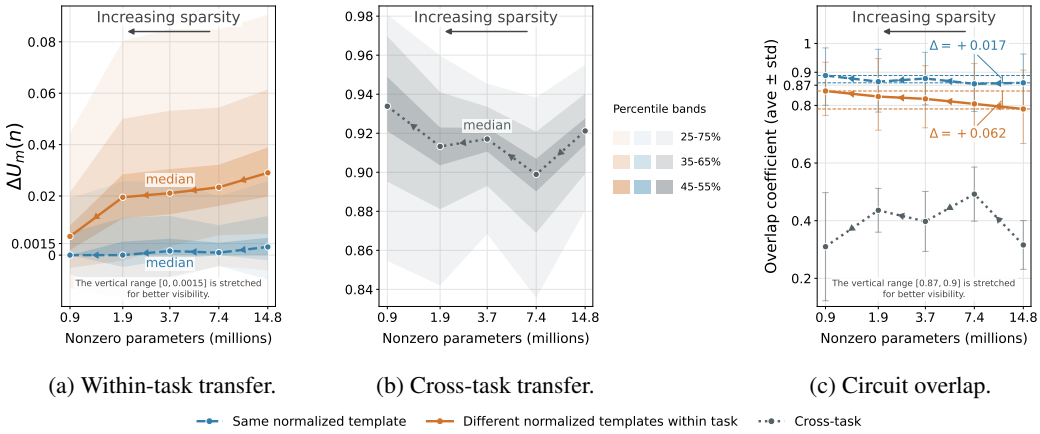
Figure 48: Redraw Figure 7 with two real samples of the same template. Sample 78 is the one with the poorest U reading in Figure 4a, and sample 79 is a sample of the same template but much higher $|Q_G|$. The curves are shown on logarithmic axes to make both the high-score region and long tail visible.

Q CIRCUIT OVERLAP ACROSS UNFAITHFULNESS THRESHOLDS



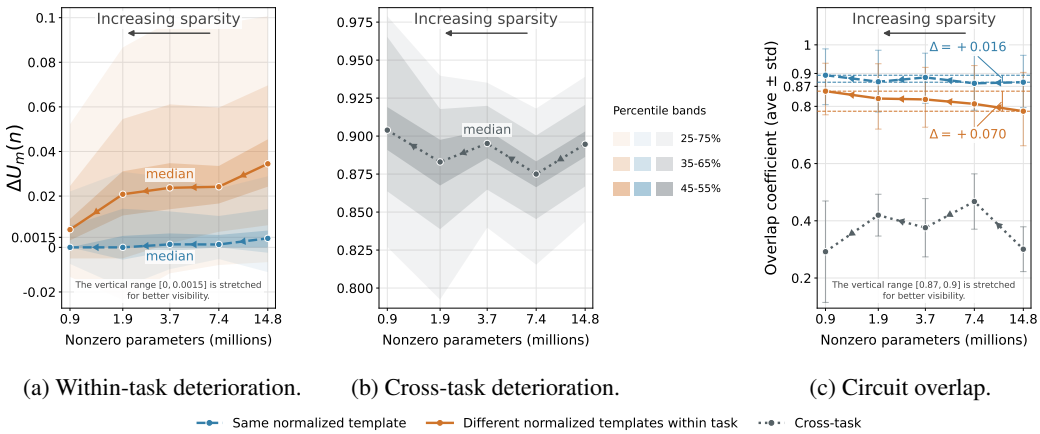
(a) Within-task deterioration. (b) Cross-task deterioration. (c) Circuit overlap.

Figure 49: Equivalent to Figure 3, with an unfaithfulness threshold of 5%.



(a) Within-task transfer. (b) Cross-task transfer. (c) Circuit overlap.

Figure 50: Equivalent to Figure 3, with an unfaithfulness threshold of 9%.



(a) Within-task deterioration. (b) Cross-task deterioration. (c) Circuit overlap.

Figure 51: Equivalent to Figure 3, with an unfaithfulness threshold of 13%.