The Validation Gap: A Mechanistic Analysis of How Language Models Compute Arithmetic but Fail to Validate It

Leonardo Bertolazzi[*1] **Philipp Mondorf**[*2, 3] **Barbara Plank**[2, 3] **Raffaella Bernardi**[4]
[1]DISI, University of Trento, Italy
[2]MaiNLP, Center for Information and Language Processing, LMU Munich, Germany
[3]Munich Center for Machine Learning (MCML), Munich, Germany
[4]Free University of Bozen-Bolzano, Italy
leonardo.bertolazzi@unitn.it {p.mondorf, b.plank}@lmu.de raffaella.bernardi@unibz.it

Abstract

The ability of large language models (LLMs) to validate their output and identify potential errors is crucial for ensuring robustness and reliability. However, current research indicates that LLMs struggle with self-correction, encountering significant challenges in detecting errors. While studies have explored methods to enhance self-correction in LLMs, relatively little attention has been given to understanding the models' internal mechanisms underlying error detection. In this paper, we present a mechanistic analysis of error detection in LLMs, focusing on simple arithmetic problems. Through circuit analysis, we identify the computational subgraphs responsible for detecting arithmetic errors across four smaller-sized LLMs. Our findings reveal that all models heavily rely on *consistency heads*—attention heads that assess surface-level alignment of numerical values in arithmetic solutions. Moreover, we observe that the models' internal arithmetic computation primarily occurs in higher layers, whereas validation takes place in middle layers, before the final arithmetic results are fully encoded. This structural dissociation between arithmetic computation and validation seems to explain why smaller-sized LLMs struggle to detect even simple arithmetic errors.

1 Introduction

In recent years, large language models have demonstrated notable performance across a variety of reasoning tasks, including arithmetic problem-solving (Sawada et al., 2023; Phan et al., 2025; Liu et al., 2024a; Achiam et al., 2023). However, a gap appears to exist between the models' ability to generate solutions and their capacity to validate them effectively (Huang et al., 2024; Hong et al., 2024; Jiang et al., 2024). Specifically, while LLMs are often able to correct mistakes once they have been identified, they struggle to detect errors in the first place (Tyen et al., 2024; Kamoi et al., 2024a,b).

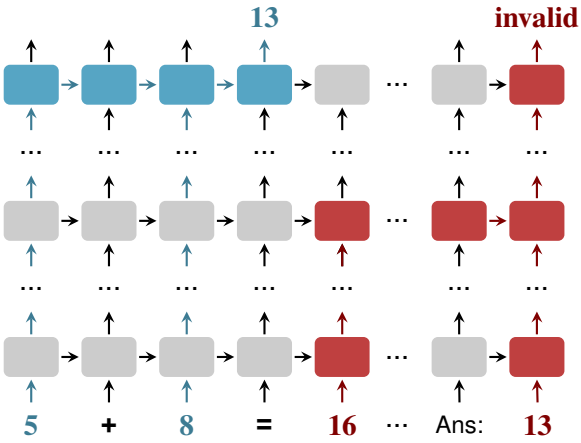


Figure 1: A schematic overview of the structurally dissociated circuits responsible for arithmetic computation and validation. While the models' internal arithmetic computation primarily occurs in higher layers, validation takes place in mid-to-lower layers, before the final arithmetic results are fully encoded (see Section 4.3).

Several studies have proposed methods to enhance LLMs' ability to detect errors and correct their own output (Welleck et al., 2023; Ye et al., 2023; Gou et al., 2024). However, comparatively little attention has been given to understanding why current models inherently struggle with error detection (Hong et al., 2024; Kamoi et al., 2024a; Li et al., 2024). In particular, few studies have examined the internal mechanisms responsible for error detection in LLMs (Liu et al., 2024b).

In this paper, we seek to bridge this gap by presenting a *mechanistic* analysis of error detection in LLMs, focusing on math word problems involving basic addition. We examine four LLMs—Qwen-2.5-(Math)-1.5B-Instruct (Yang et al., 2024a,b), Llama-3.2-3B-Instruct (Dubey et al., 2024), and Phi-3-Mini-4k-Instruct (Abdin et al., 2024)—to understand how these models detect arithmetic errors

[*]Equal contribution. Listing order is random.

Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, pages 29376–29413
November 4-9, 2025

and why they struggle with this task. Specifically, we identify and analyze the computational subgraphs (or *circuits*) responsible for error detection, examining the role of identified modules within the broader task context. Additionally, we analyze how these circuits compare to those involved in computing arithmetic results, seeking to understand the structural differences between arithmetic computation and validation in LLMs. To the best of our knowledge, this is the first study to examine arithmetic error detection in LLMs through the lens of mechanistic interpretability. Our findings reveal:

- Circuits for detecting arithmetic errors are structurally similar across different models.
- The error detection process is governed by *consistency heads*—attention heads located in lower to middle layers that check for surface-level alignment of numerical values in the arithmetic solution. By patching a small subset of these heads, we can effectively control the models' error detection behavior.
- The mechanisms for arithmetic computation and validation appear to be structurally dissociated. While the models' internal arithmetic computation is predominantly conducted in higher layers, validation is performed in middle layers, before the final arithmetic results are fully encoded (see Figure 1).
- Adding latent activations from higher layers to the residual stream in lower layers significantly enhances the models' ability to detect errors, effectively closing the validation gap.[1]

Our analysis shows that mechanistic interpretability can offer valuable insights into how models detect—or fail to detect—arithmetic errors. While focused on smaller LLMs and basic math word problems, we hope that this work provides a foundation for future work to study error detection in larger models and more complex tasks.

2 Background

A common goal in interpretability research is to gain a deeper understanding of the internal mechanisms that drive the behavior of language models for a given task (Ferrando et al., 2024; Mueller et al., 2024; Bereska and Gavves, 2024). The *circuit* framework seeks to achieve this by identifying model components that causally influence the model's task output (Elhage et al., 2021; Wang et al., 2023; Hanna et al., 2023). In essence, a *circuit* refers to the computational subgraph $\mathcal{C} \subset \mathcal{G} = (\mathcal{V}, \mathcal{E})$ that represents the task-relevant flow of information across the model's layers (Conmy et al., 2023; Bhaskar et al., 2024). A node $v \in \mathcal{V}$ in this graph can represent different components, depending on the desired level of granularity—ranging from entire attention or MLP layers, to individual attention heads, to single neurons (Mueller et al., 2024). An edge $e_{ij} = (v_i, v_j) \in \mathcal{E}$ denotes a connection between two nodes, where the output of the source node v_i serves as the input to the destination node v_j. The total input received by a node v_j can be expressed as $\sum_{e_{ij}=(v_i,v_j)\in\mathcal{E}_{v_j}} \boldsymbol{z}_i$ where $\boldsymbol{z}_i$ represents the activation of node v_i and $\mathcal{E}_{v_j}$ denotes the set of incoming edges to v_j.

Circuit Identification. A method for identifying circuits in language models is *activation patching* (Vig et al., 2020; Geiger et al., 2021; Meng et al., 2022). The key idea is to intervene on the latent activations of components in the computation graph $\mathcal{G}$ to measure their indirect effect (Pearl, 2001) on the model's output. Adopting the terminology of Zhang and Nanda (2024), activation patching requires three forward passes to determine a component's indirect effect for a given input:

1. Clean run: run the model on a *clean* prompt X_{clean}, for which the model generates the desired task-specific output y_{clean}. Cache the component's latent activations, denoted as $\boldsymbol{z}_i$.
2. Corrupted run: run the model on a *corrupted* prompt $X_{corrupt}$, for which the model generates a related but altered output $y_{corrupt}$.
3. Patched run: run the model on $X_{corrupt}$, but this time, replace the component's activations associated with $X_{corrupt}$ with the cached activations $\boldsymbol{z}_i$ from the clean run.

Finally, the indirect effect is calculated by comparing the output of the *patched* run to that of the *corrupted* run using a predefined metric $\mathcal{P}$.[2] If the component under consideration causally influences the model's task output, the patched activations should shift the prediction $y_{corrupt}$ toward y_{clean}.

[1] By validation gap, we mean both the structural separation and the performance gap between arithmetic computation and validation in LLMs.

[2] This metric typically evaluates differences in logits or output probabilities relative to the clean output y_{clean}.

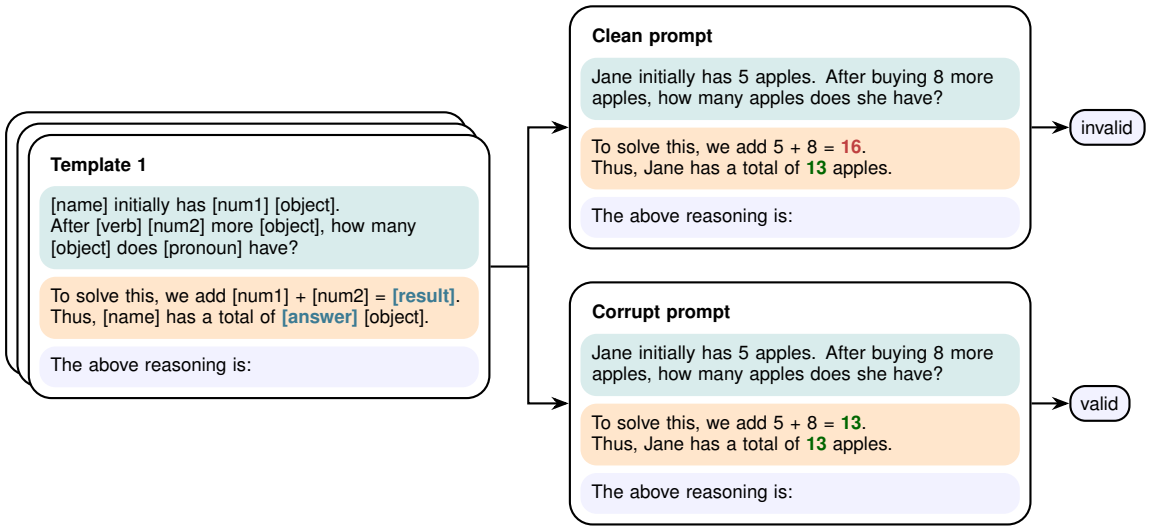


Figure 2: Our data generation setup. We use eight templates to generate samples that consist of a simple arithmetic problem, its corresponding solution, and a final statement assessing the solution's validity. Words enclosed in [square brackets] serve as placeholders for components that are substituted with specific content. For each generated sample, a pair of (*clean*, *corrupt*) prompts is derived. Counterintuitively, *clean* prompts *contain* errors, as they represent prompts for which the model exhibits the desired error detection behavior (predicting "invalid").

As performing these steps for every model component and sample can become computationally expensive, several approximations trade off computational cost against accuracy (Syed et al., 2024; Nanda, 2024; Hanna et al., 2024). In this work, we consider ***edge attribution patching*** **(EAP)** (Syed et al., 2024), a linear approximation of activation patching requiring only two forward and one backward pass. EAP focuses on the indirect effect of ***edges*** $e_{ij} \in \mathcal{E}$, which represent inputs to a node v_j from earlier nodes v_i. Specifically, the causal influence is approximated using the *absolute attribution score* $|\nabla_{\boldsymbol{z}}\mathcal{P}|$, which measures the change in $\mathcal{P}$ under the intervention (for further details, see Appendix B). Once these scores are computed, the top-k edges with the highest absolute attribution values are selected to define the circuit $\mathcal{C}$. Although EAP is only a linear approximation of activation patching, it has been successfully employed in studies to identify circuits within language models for tasks such as indirect object identification, subject-verb agreement, and greater-than attribution (Syed et al., 2024; Hanna et al., 2024; Miller et al., 2024).

3 Circuits for Arithmetic Error Detection

In this section, we present the dataset used to study arithmetic error detection in LLMs. Additionally, we outline our use of *edge attribution patching* to identify circuits responsible for the task.

Dataset. In this study, we focus on simple math word problems. As illustrated in Figure 2, we employ templates to systematically generate data (Wang et al., 2023; Hanna et al., 2023). Each sample consists of a basic arithmetic problem, its corresponding solution, and a final statement that evaluates the solution's validity. Samples derived from the same template maintain a consistent sentence structure but incorporate ***variable*** components such as [names] or numerical [values] (left box in Figure 2). To analyze the models' error detection mechanisms, we introduce simple arithmetic errors into the sample's solution statement. Specifically, we consider *two* types of errors separately: *i)* a miscalculation of the arithmetic result, and *ii)* an incorrect final numeric answer. Note that the ***perturbed*** sample forms our *clean prompt*, for which models can ***successfully detect*** the arithmetic error (see upper-right box in Figure 2, showing an error for the arithmetic result). Additionally, we construct a *corrupt prompt* without errors, for which models predict the solution to be "valid" (lower-right box). We use single-digit numerical values that sum to a two-digit arithmetic result across all templates and samples. The introduced errors always correspond to a different, incorrect two-digit number ranging from 10 to 19.

For each template, we generate 6,000 pairs of (*clean*, *corrupt*) prompts. We use eight different

Error Type	Qwen-2.5-1.5B	Qwen-2.5-Math-1.5B	Llama-3.2-3B	Phi-3-Mini-3.8B
Arithmetic Result	60.53 ± 10.92	98.99 ± 2.18	87.67 ± 12.51	89.51 ± 26.75
Numeric Answer	59.03 ± 10.72	98.53 ± 3.17	86.44 ± 13.35	89.44 ± 26.84

Table 1: Accuracy of models in correctly classifying the solutions' validity of (*clean*, *corrupt*) prompt pairs. Values represent the mean accuracy across all templates, reported with their corresponding standard deviation.

templates that vary in syntactic structure and token length while preserving the fundamental task logic. Each type of error (arithmetic result vs. final numeric answer) is examined separately. In total, we obtain a dataset of 6,000 (*clean*, *corrupt*) prompt pairs for each template $\mathcal{T}_{\{1:8\}}$ per error type. For further details on the data generation process and templates, please refer to Appendix C.

Method. As described in Section 2, we employ *edge attribution patching* (EAP) (Syed et al., 2024) to identify circuits responsible for arithmetic error detection in LLMs. For each template $\mathcal{T}_i$, we use 5,000 pairs of (*clean*, *corrupt*) prompts to determine the circuit—specifically, the set of edges $\mathcal{C}_i = \mathcal{E}_i \subset \mathcal{E}$—that causally influences the model's error detection behavior (see Section 2 for more details). Since all samples within a template contain the same number of tokens, we apply ***token-wise*** EAP, which allows us to assess the causal impact of edges at each token position of the prompt. Following Syed et al. (2024), the *absolute attribution score* $|\nabla_{\boldsymbol{z}}\mathcal{P}|$ is computed using the average logit difference related to the models' answer tokens ("valid", "invalid") as metric $\mathcal{P}$ (see Appendix B for further details). Once the attribution scores are obtained, we use the template's remaining 1,000 (*clean*, *corrupt*) prompt pairs to find the minimal set of top-k edges for which the circuit achieves a faithfulness score between 99%–101%. For a more detailed explanation of the search procedure and the computation of the faithfulness score, please refer to Appendix B.2 and B.3.

Soft Intersection Circuit. After identifying a circuit $\mathcal{C}_i$ for each template $\mathcal{T}_i \in \{\mathcal{T}_1, \ldots, \mathcal{T}_8\}$, we aim to find a ***final*** subset of edges $\mathcal{E}_{\mathcal{C}} \subset \mathcal{E}$ that ***generalizes across all templates*** $\mathcal{T}_i$, ensuring high faithfulness. To achieve this, we compute the *soft intersection circuit*, which includes edges present in at least $\frac{1}{8} \leq \tau \leq \frac{8}{8}$ of the identified circuits $\{\mathcal{C}_1, \ldots, \mathcal{C}_8\}$. The soft intersection is defined through a *membership function* that determines the proportion of identified circuits in which a given edge $e \in \mathcal{E}$ appears:

$$f(e) = \frac{1}{8} \sum_{i=1}^{8} \mathbb{1}_{\mathcal{C}_i}(e) \tag{1}$$

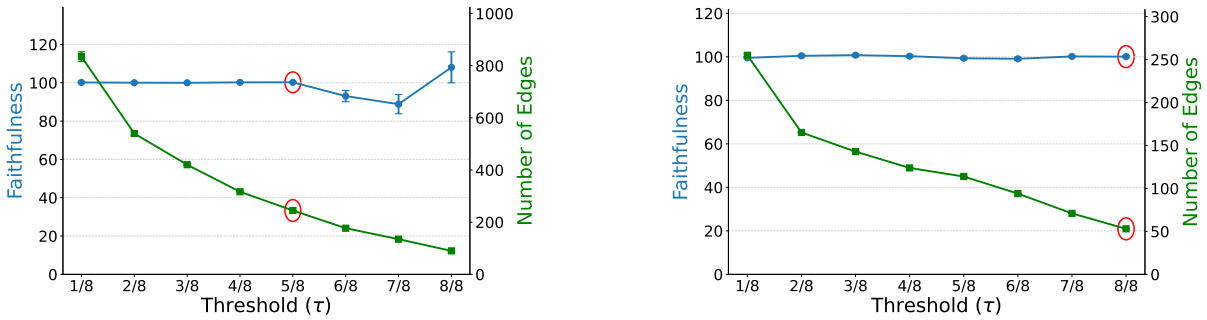
where $\mathbb{1}_{\mathcal{C}_i}(e)$ is an indicator function that assigns a value of 1 if $e \in \mathcal{C}_i$ and 0 otherwise. Consequently, $f(e)$ takes values in $\{0, \frac{1}{8}, \ldots, \frac{8}{8}\}$. The soft intersection circuit is then formally defined as $\mathcal{C}^{(\tau)} = \mathcal{E}_{\mathcal{C}^{(\tau)}} = \{e \in \mathcal{E} \mid f(e) \geq \tau\}$. This formulation allows for a flexible trade-off: setting $\tau = \frac{1}{8}$ yields the union of all identified circuits, while $\tau = \frac{8}{8}$ results in their strict intersection.[3] By varying τ from $\frac{1}{8}$ to $\frac{8}{8}$, we balance faithfulness against the numbers of edges considered, progressively filtering out template-specific redundant edges.

Models. In this study, we consider four different LLMs—Qwen-2.5-(Math)-1.5B-Instruct (Yang et al., 2024a,b), Llama-3.2-3B-Instruct (Dubey et al., 2024), and Phi-3-Mini-4k-Instruct (Abdin et al., 2024), to assess the influence of varying architectures, model scales, and fine-tuning procedures (particularly for math). Appendix D.1 provides details about models and prompts. Our code is publicly available at: https://github.com/mainlp/validation-gap.

4 Experiments

Before identifying circuits, we ensure that all models are capable of detecting the types of arithmetic errors described in Section 3. Specifically, we randomly generate 5,000 (*clean*, *corrupt*) prompt pairs for each template and evaluate whether the models predict tokens that correctly indicate the validity of the presented solutions (we expect predictions such as "invalid", "incorrect", or "wrong" for *clean* prompts, and "valid", "correct", or "right" for *corrupt* prompts). Table 1 summarizes the models' average accuracy along with the standard deviation across templates. A pair is considered correctly classified if the model's highest logit falls within

[3]Note that since we employ token-wise EAP to identify relevant edges *per token position*, we assign abstract yet meaningful labels to each token position, ensuring transferability across templates. Further details on this labeling process can be found in Appendix C.2.



(a) Error at the arithmetic result

(b) Error at the final numeric answer

Figure 3: The number of edges and faithfulness scores averaged across all templates (with standard deviation) of the *soft intersection circuit* for different τ values. Red circles mark the circuit that best balances size and faithfulness.

the set of correct labels for the clean and corrupt prompts, respectively. We observe that all models are able to detect the type of errors considered, with Qwen-2.5-Math-1.5B achieving near-perfect accuracy, while Qwen-2.5-1.5B performs worst with approximately 60% accuracy for both error types.

In the subsequent circuit identification process, we filter the generated prompt pairs to ensure that for all samples, the models predict the desired outputs, i.e., $y_{clean} \in$ {"invalid", "incorrect", "wrong"} and $y_{corrupt} \in$ {"valid", "correct", "right"} (see Section 2).

4.1 Identified Circuits

As described in Section 3, we employ edge attribution patching to identify circuits $\mathcal{C}_i = \mathcal{E}_i \subset \mathcal{E}$ that achieve faithfulness scores between 99% and 101% for each template $\mathcal{T}_i \in \{\mathcal{T}_1, \dots \mathcal{T}_8\}$. We find that for all models, only 100 to 900 edges (less than 0.1% of total edges) are sufficient to achieve around 100% faithfulness for the task. Due to space constraints, we present the faithfulness scores and the corresponding number of edges for each circuit in Table 12 in the Appendix—categorized by model, template, and error type. Once a circuit is identified for each template, we compute the *soft intersection circuit* $\mathcal{C}^{(\tau)}$ as outlined in Section 3. Figure 3 illustrates the faithfulness scores and associated edge counts of the *soft intersection circuits* for Qwen-2.5-1.5B across different threshold values $\tau \in \{\frac{1}{8}, \dots, \frac{8}{8}\}$. For errors at the position of the arithmetic result (e.g., "$5 + 8 = 16$"), the soft intersection circuit $\mathcal{C}^{(5/8)}_{\text{result}}$ achieves an average faithfulness of around 100%, effectively ***generalizing across all templates***, while retaining only 245 edges (red circles in Figure 3a). For errors at the position of the final numeric answer (e.g., "$\dots 5 + 8 = 13$. Thus, Jane has 16 apples."), even the strict intersection ($\tau = \frac{8}{8}$) yields an almost perfect average faithfulness score across all templates, with the number of relevant edges reduced to 53 (see Figure 3b). In subsequent analyses, we focus on $\mathcal{C}^{(5/8)}_{\text{result}}$ and $\mathcal{C}^{(8/8)}_{\text{answer}}$ for Qwen-2.5-1.5B. Notably, similar results can be found for ***all*** models, as presented in Appendix E.1, Figures 11 to 13.

A visualization of Qwen-2.5-1.5B's circuit $\mathcal{C}^{(8/8)}_{\text{answer}}$ for detecting errors at the position of the final numeric answer is shown in Figure 4. As mentioned in Section 3, we employ ***token-wise*** EAP. This means that we identify edges for each token position of the prompt. Figure 4 shows that most edges are concentrated at the position of the second digit of the final numeric answer ([answer-

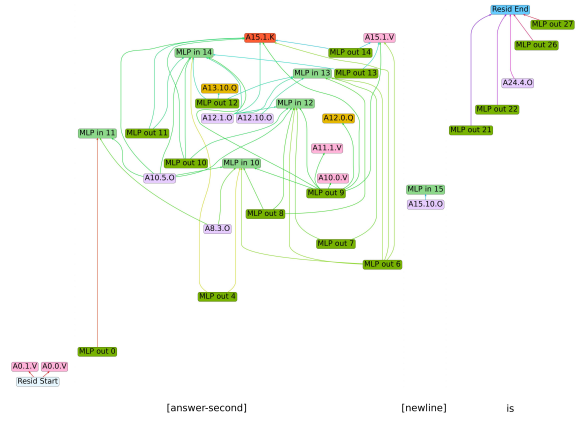


Figure 4: The soft intersection circuit $\mathcal{C}^{(8/8)}_{\text{answer}}$, representing the set of edges that causally influence the output of Qwen-2.5-1.5B when detecting errors at the position of the final numeric answer. Attention heads are abbreviated as *A.layer.head.K(ey)/V(alue)/Q(uery)/O(ut)*, while MLPs are represented as *MLP in/out layer*. Corresponding token positions are indicated by the labels at the bottom.

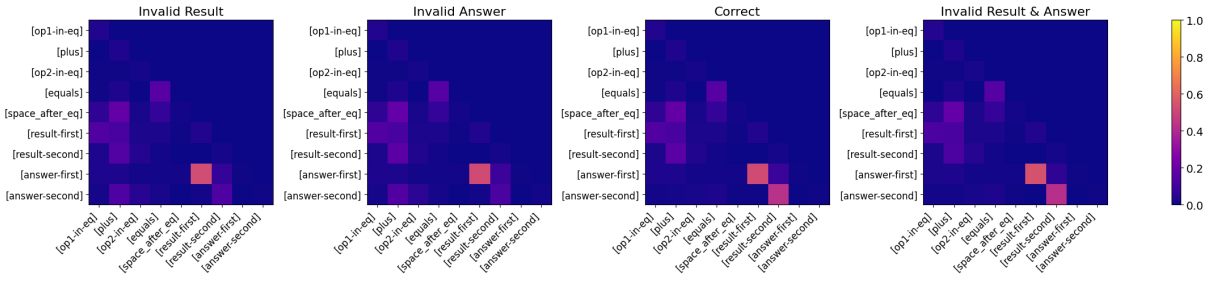


Figure 5: Attention patterns of a *consistency head* in Qwen-2.5-1.5B (head 2 in layer 12). Reported scores are averaged over 5,000 prompts where (*left*) an error is present at the position of the arithmetic result, (*second to left*) an error is present at the position of the final numeric answer, (*second to right*) no error is present, and (*right*) a consistent error is present at both considered positions.

second])[4] in the middle layers 4 to 15 of the model. For example, this corresponds to the position of the 6 in "...$5 + 8 = 13$. Thus, Jane has 16 apples." Additionally, a smaller number of edges appear in higher layers 21 to 27 at the final token position of the prompt ("is"), predominantly connecting MLP layers with the final residual output. Interestingly, this structural pattern appears ***consistent*** across all models (see Figures 31 to 33 in the Appendix) and even across error types (Figures 26 to 29 in the Appendix).

One Circuit for Arithmetic Error Detection. The structural similarity between circuits identified for the two distinct error types is further supported by their edge overlap. When computing the Intersection over Minimum (IoM)[5] between the circuits $\mathcal{C}^{(5/8)}_{\text{result}}$ and $\mathcal{C}^{(8/8)}_{\text{answer}}$, we obtain a value of 0.92, indicating that a substantial subset of edges of $\mathcal{C}^{(8/8)}_{\text{answer}}$ is also present in $\mathcal{C}^{(5/8)}_{\text{result}}$. Additionally, we compute the intersection of $\mathcal{C}^{(5/8)}_{\text{result}}$ and $\mathcal{C}^{(8/8)}_{\text{answer}}$, and evaluate the faithfulness of the resulting circuit. Despite comprising only 49 edges, the intersected circuit achieves an average faithfulness score of $78.60\% \pm 7.46\%$ on samples involving errors in the arithmetic result and $82.50\% \pm 5.08\%$ on samples with errors at the final numeric answer. Notably, these observations are consistent across all models considered. Corresponding results are provided in Appendix E.2, specifically in Table 5 and 6.

4.2 Decoding the Error Detection Process

Once we obtain a soft intersection circuit $\mathcal{C}^{(\tau)}$, we can analyze its components to gain deeper insights into the model's error detection mechanisms. For instance, Figure 4 illustrates that $\mathcal{C}^{(8/8)}_{\text{answer}}$ contains several edges that connect attention heads in the model's middle layers 8 to 15 at the position of the error ([answer-second]). To better understand the function of such attention heads, we compute their average attention scores over a set of input prompts. Figure 5 shows the average scores of the second attention head in layer 12 present in Qwen-2.5-1.5B's $\mathcal{C}^{(5/8)}_{\text{result}}$ circuit. Specifically, we visualize attention scores for four different sets of prompts: *i)* prompts with an error at the position of the arithmetic result, *ii)* prompts with an error at the position of the final numeric answer, *iii)* prompts without errors, and *iv)* prompts with a consistent error at ***both*** the arithmetic result and the final numeric answer (e.g., "...$5 + 8 = 16$. Thus, Jane has 16 apples."). Two notable attention patterns emerge. For prompts where an error is present either at the position of the arithmetic result (e.g., "...$5 + 8 = 16$. Thus, Jane has 13 apples.") or at the position of the final numeric answer (e.g., "...$5 + 8 = 13$. Thus, Jane has 16 apples."), we observe high attention scores between the first digit of the arithmetic result ([result-first]) and the first digit of the final numeric answer ([answer-first]), but ***not*** for the corresponding second digits ([result-second] and [answer-second], respectively). In contrast, for prompts *without* errors or those with *consistent* errors at both positions (e.g., "...$5 + 8 = 16$. Thus, Jane has 16 apples."), we observe high average attention scores for ***both*** the first and second digits of the result and the final numeric answer. In essence, we observe that the attention head exhibits high average attention scores when the digits of the arithmetic result and the final numeric answer align. We refer to such attention heads as *consistency heads*—attention heads that assess surface-level alignment of numerical values in the solution prompt. Notably, we

[4]Note that Qwen-2.5 uses a one-digit tokenization scheme; i.e., the number 16 is encoded into two separate tokens.

[5]Detailed information on the computation of the edge overlap between circuits can be found in Appendix D.2.

Error Type	Qwen-2.5-1.5B	Qwen-2.5-Math-1.5B	Llama-3.2-3B	Phi-3-Mini-3.8B
Result & Answer	12.39 ± 6.00	3.37 ± 2.06	12.27 ± 9.08	40.98 ± 18.41

Table 2: Accuracy of models in correctly classifying the solutions' validity of (*clean*, *corrupt*) prompt pairs where *clean* prompts contain a ***consistent*** error at both the position of the arithmetic result and the position of the final numeric answer. Values represent the mean accuracy across all templates, reported with their standard deviations.

find several consistency heads across ***all*** models (see Figures 16a and 17a in the Appendix). Additionally, we observe *(in)*consistency heads, which display high attention scores for numerical values that misalign (Figures 14b to 17b in the Appendix).

Consistency Heads Govern Arithmetic Error Detection. We hypothesize that *consistency heads* in the lower to middle layers of the models play an important role in the arithmetic error detection process. This hypothesis has two major implications: *i)* models may struggle to distinguish between samples ***without*** errors and those containing a ***consistent*** error at ***both*** the position of the arithmetic result and the final numeric answer (e.g., "$\ldots 5 + 8 = 16$. Thus, Jane has 16 apples."); and *ii)* a small subset of *consistency heads* can significantly influence the models' arithmetic error detection behavior. To test our hypothesis, we first evaluate models on 1,000 (*clean*, *corrupt*) prompt pairs for each template, where *clean* prompts contain a ***consistent*** error at both error positions. As shown in Table 2, all models exhibit significant difficulties in detecting these type of errors. For instance, Qwen-2.5-1.5B achieves an average accuracy of only $12.39\% \pm 6.00\%$, indicating a strong bias toward labeling prompts with a consistent error as "valid". Even for Phi-3-Mini-4k-Instruct, the accuracy drops from about 89% when detecting errors at either the arithmetic result or the final numeric answer (see Table 1) to $40.98\% \pm 18.41\%$ when both positions contain a ***consistent*** error.

Next, we analyze the influence of individual consistency heads on the model's error detection behavior through two complementary experiments. First, when Qwen-2.5-1.5B is given a prompt with a ***consistent*** error at both positions (for which it incorrectly predicts "valid"), we *patch* the latent activations of ***six*** consistency heads (see Table 4 in the Appendix for the exact heads) with the corresponding activations from a prompt containing a ***single*** error at the arithmetic result (for which the model correctly predicts "invalid"). If consistency heads indeed govern error detection, the model should change its initial prediction from "valid" to "invalid" under this intervention. Second, we perform the reverse: running the model on a single-error prompt (correctly predicted as "invalid"), we patch the activations of the consistency heads with those from a consistent-error prompt. In this case, we would expect the intervention to reduce the error detection accuracy.

Figure 6 shows the resulting changes in detection accuracy (predicting "invalid") for Qwen-2.5-1.5B. As expected, the first intervention (ConsErr[heads := SingErr]) reliably flips the model's prediction from "valid" to "invalid," demonstrating that inconsistency signals injected via these heads can causally alter the model's output. By contrast, the reverse intervention (SingErr[heads := ConsErr]) changes the prediction from "invalid" to "valid" only in some cases. Similar results hold across ***all*** models (see Figure 18 in the Appendix). A plausible explanation might be that the model has more consistency heads than the ones we patch, which still signal an inconsistency for the given prompt. This touches upon the problem of completeness in circuit discovery, where circuits that can faithfully reproduce

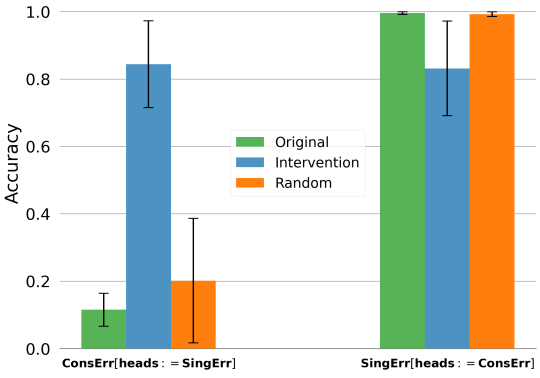


Figure 6: Error detection accuracy of Qwen-2.5-1.5B before and after patching six *consistency heads*. On the left, we evaluate the model on consistent-error prompts and patch activations from a single-error run; on the right, we evaluate on single-error prompts and patch with activations from a consistent-error run. As a control, we patch six randomly chosen attention heads.

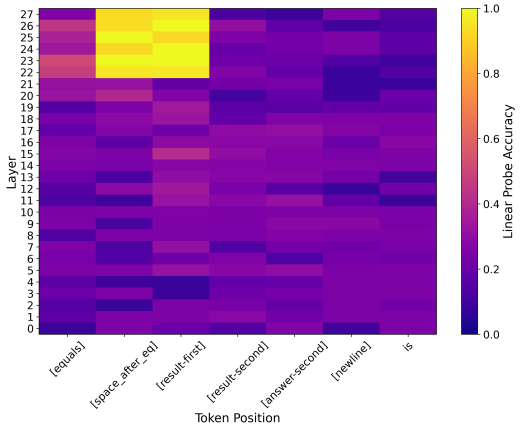


Figure 7: The linear probe's accuracy across all layers of Qwen-2.5-1.5B at selected token positions.

the model's task behavior are not guaranteed to include all components involved in the model's task behavior (Mondorf et al., 2025). For details on the patching procedure, we refer to Appendix D.4.

4.3 Dissociation of Arithmetic Validation and Computation

Our findings presented in Section 4.2 suggest that the considered models tend to rely on surface-level consistency checks rather than re-evaluating the given arithmetic equation and comparing the result with the final numeric answer. Notably, we find that all models achieve 100% accuracy in predicting the correct result of the arithmetic equation provided in each prompt (e.g., $5+8=?$). However, we hypothesize that this correctly predicted result is not used for validation. To better understand the relationship between the models' arithmetic computation and validation procedures, we identify another set of circuits responsible for computing the correct arithmetic result at the position of the arithmetic equation, following a similar procedure as described in Section 3. For further details on the identification process, please refer to Appendix D.3. Interestingly, we find that for ***all*** models, the identified arithmetic circuits predominantly contain edges in higher layers (due to space constraints, visualizations of these circuits are provided in Figures 22 to 25 in the Appendix). This structural ***dissociation*** between the circuits responsible for arithmetic computation and those involved in validation seems to explain the models' difficulties in detecting basic arithmetic errors. Specifically, although the models can successfully re-compute the result of a given arithmetic equation, the final arithmetic outcome is not fully encoded when the model checks for numeric consistency in middle layers. To support this hypothesis, we train a linear probe to predict the correct arithmetic result based on the hidden states of the model's residual stream (for training details, see Appendix D.5). Figure 7 shows the probe's accuracy across different layers of Qwen-2.5-1.5B and selected token positions. Only in the higher layers (after the consistency check) does the model's residual stream linearly encode information about the correct arithmetic result. Interestingly, similar patterns are observed for other models, too (see Figure 20 in the Appendix).

We demonstrate that by "bridging" the gap between arithmetic computation and validation, the model's error detection capacity for prompts with consistent errors at both error positions can be significantly enhanced. Figure 8 shows the error detection accuracy of Qwen-2.5-1.5B before and after we ***add*** the hidden representation of the residual stream from layer 22 at token position [result-first] to the residual stream of layer 1 at token position [result-second] (essentially "moving" information from the top yellow layers in Figure 7 to lower layers). Notably, this approach improves the model's ability to detect ***consistent*** errors by 81%. Furthermore, accuracy on samples containing errors only at the position of the arithmetic result does not decline markedly, suggesting that the added representation enhances rather than overwrites existing information. A similar trend for other models is presented in Figure 21 in the Appendix.

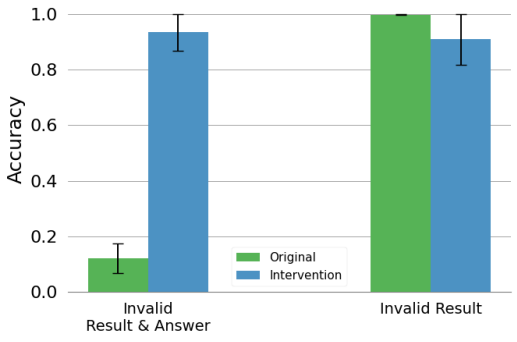


Figure 8: Error detection accuracy of Qwen-2.5-1.5B on (*clean*, *corrupt*) prompt pairs where (*left*) the *clean* prompt contains a consistent error at both error positions, and (*right*) an error is present only at the position of the arithmetic result. The blue intervention bar denotes the result after adding the hidden representation of the residual stream in layer 22 (at [result-first]) to the residual stream of layer 1 (at [result-second]).

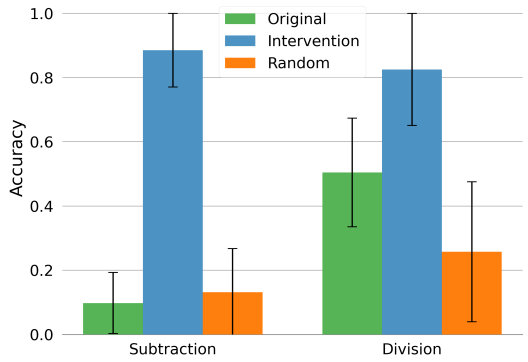


Figure 9: Error detection accuracy of Qwen-2.5-1.5B before and after patching six consistency heads. We evaluate the model on consistent-error prompts related to subtraction (*left*) or division (*right*) and patch activations from a single-error run. As a control, we patch six randomly chosen attention heads.

4.4 Consistency Heads in Other Arithmetic Operations

Given the important role of consistency heads in our experiments on addition, we conduct further analyses to investigate whether their influence extends to other arithmetic operations. To this end, we generate new math word problems involving subtraction, multiplication, and division, using the same template structure with the two key error positions at the arithmetic result and the final numeric answer.[6] As for addition, we first evaluate the models' error detection accuracy on 1,000 prompt pairs per template. Similarly, we find that all models (except for Phi-3-Mini-4k-Instruct) encounter greater difficulty in detecting ***consistent*** errors at both error positions compared to single-position errors. Complete results for all models and operations are reported in Table 7 in the Appendix.

Next, for models that struggle with consistent errors, we replicate the consistency head patching experiment presented in Section 4.2, this time for prompts involving other arithmetic operations. Focusing on Qwen-2.5-1.5B, we evaluate the model on subtraction and division prompts with ***consistent*** errors at both positions, where it incorrectly judged the reasoning as "valid," and patch the latent activations of the consistency heads with those obtained from prompts containing an error only at the arithmetic result. Notably, across all operations, we patch the same consistency heads initially identified with prompts involving additions (Table 4 in the Appendix). Figure 9 shows the change in error detection accuracy, indicating that the causal role of consistency heads in error detection extends beyond addition. Similar effects are observed for Qwen-2.5-Math-1.5B and Llama-3.2-3B (Figure 19 in the Appendix).

Finally, we find that some larger models, such as Llama-3.1-70B, similarly struggle with detecting consistent arithmetic errors, as shown in Table 8 in the Appendix. This may indicate that, to some extent, these models also rely on consistency heads—a direction that future work could explore.

[6] See Table 10 in the Appendix for the full set of templates and variables considered when generating data for other arithmetic operations.

5 Related Work

Self-correction in LLMs. Self-correction in LLMs refers to the ability of models to correct their own generated output (Kamoi et al., 2024b; Huang et al., 2024; Madaan et al., 2023). Recent studies (Tyen et al., 2024; Kamoi et al., 2024a) suggest that LLMs tend to struggle with *intrinsic* self-correction, especially with *detecting* errors in their own output (Huang et al., 2024; Tyen et al., 2024; Kamoi et al., 2024b). While most studies focus on improving the models' ability to self-correct (Kamoi et al., 2024b; Madaan et al., 2023; Chen et al., 2024; Zhao et al., 2023), we study error detection from a mechanistic point of view.

Arithmetic and Error Detection in LLMs. The underlying processes of arithmetic reasoning and error detection have been studied independently in LLMs so far. Several studies (Stolfo et al., 2023; Zhang et al., 2024; Nikankin et al., 2024) use causal mediation analysis (Pearl, 2001) to identify circuits that account for how LLMs process arithmetic operations. As of now, only few studies have analyzed self-correction in LLMs beyond the models' generated output (Li et al., 2024; Liu et al., 2024b).

6 Conclusion

This paper presents a mechanistic analysis of arithmetic error detection in LLMs. Our findings reveal that smaller-sized LLMs heavily rely on *consistency heads*—attention heads that evaluate surface-level alignment of numerical values in an arithmetic solution. Moreover, we highlight a structural dissociation between the models' arithmetic computation and validation processes. Finally, we show that bridging this gap can significantly improve the models' arithmetic error detection capacity.

7 Limitations

While our study provides new insights into the mechanisms underlying arithmetic error detection in LLMs, several limitations exist that can be addressed by future work.

Task Design. This study focuses on examining the error detection behavior of LLMs in the context of simple arithmetic tasks. Specifically, we analyze math word problems involving the addition of two single-digit numbers that yield a two-digit result, as described in Section 3. Future research could extend these findings to other arithmetic operations, such as subtraction, multiplication, and division, or explore their applicability to more complex mathematical problems. It would also be valuable to investigate how these insights generalize to other domains, such as logical or causal reasoning tasks.

Model Selection. As discussed in Sections 1 and 3, our analysis is limited to four smaller-sized LLMs. Although we observe consistent patterns across various model architectures, sizes, and fine-tuning procedures (particularly within the mathematical domain), future research could investigate how these findings extend to larger models with more advanced arithmetic capabilities. Our behavioral experiments with models such as Llama-3.1-70B-Instruct and Qwen-2.5-32B-Instruct (see Table 8 in the Appendix) show that even larger LLMs tend to struggle more with consistent errors at both error positions, compared to detecting an error present only at the position of the arithmetic result or the final answer. This may indicate that, similarly to smaller models, these models—at least to some extent—rely on *consistency heads* that are susceptible to the validation gap. We believe this is a promising direction for future work to explore.

Circuit Identification Method. As highlighted in Section 2, edge attribution patching (Syed et al., 2024) serves as a linear approximation of activation patching (Vig et al., 2020). It involves a trade-off between accuracy and computational efficiency. Notably, circuits identified using EAP are not guaranteed to be *complete* (Wang et al., 2023). Although the circuits identified in this study are highly sparse (comprising less than 0.1% of the total edges) and achieve near-perfect task faithfulness (see Section 4), future research should explore how these circuits compare to those identified through more exact methods.

Acknowledgments

We would like to thank ELLIS—the European Laboratory for Learning and Intelligent Systems—as the collaboration that led to this paper started at a workshop organized by Raquel Fernández and Sandro Pezzelle at MFO, the Oberwolfach Research Institute for Mathematics in the German Black Forest, on behalf of the ELLIS NLP program. We are also grateful to the members of the MaiNLP lab for their valuable feedback, with special thanks to Michael Hedderich, Robert Litschko, Diego Frassinelli, Rob van der Goot, Siyao Peng, Yang Janet Liu, Xinpeng Wang, Verena Blaschke, Raoyuan Zhao, Elena Senger, Felicia Körner, Soh-Eun Shim, Andreas Säuberli, Florian Eichin, Shijia Zhou, and Beiduo Chen. We further thank the anonymous reviewers for their insightful comments and suggestions. RB acknowledges the generous support of Amazon Alexa, whose donation helped make this work possible. Finally, we acknowledge the support for BP through the ERC Consolidator Grant DIALECT 101043235.

References

Marah Abdin, Jyoti Aneja, Hany Awadalla, Ahmed Awadallah, Ammar Ahmad Awan, Nguyen Bach, Amit Bahree, Arash Bakhtiari, Jianmin Bao, Harkirat Behl, and 1 others. 2024. Phi-3 technical report: A highly capable language model locally on your phone. *arXiv preprint arXiv:2404.14219*.

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, and 1 others. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Leonard Bereska and Stratis Gavves. 2024. Mechanistic interpretability for AI safety - a review. *Transactions on Machine Learning Research*. Survey Certification, Expert Certification.

Adithya Bhaskar, Alexander Wettig, Dan Friedman, and Danqi Chen. 2024. Finding transformer circuits with edge pruning. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. 2024. Teaching large language models to self-debug. In *The Twelfth International Conference on Learning Representations*.

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

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, and 1 others. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Nelson Elhage, Neel Nanda, Catherine Olsson, Tom Henighan, Nicholas Joseph, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, Nova DasSarma, Dawn Drain, Deep Ganguli, Zac Hatfield-Dodds, Danny Hernandez, Andy Jones, Jackson Kernion, Liane Lovitt, Kamal Ndousse, and 6 others. 2021. A mathematical framework for transformer circuits. *Transformer Circuits Thread*. Https://transformer-circuits.pub/2021/framework/index.html.

Javier Ferrando, Gabriele Sarti, Arianna Bisazza, and Marta R Costa-jussà. 2024. A primer on the inner workings of transformer-based language models. *arXiv preprint arXiv:2405.00208*.

Atticus Geiger, Hanson Lu, Thomas Icard, and Christopher Potts. 2021. Causal abstractions of neural networks. In *Advances in Neural Information Processing Systems*, volume 34, pages 9574–9586. Curran Associates, Inc.

Zhibin Gou, Zhihong Shao, Yeyun Gong, yelong shen, Yujiu Yang, Nan Duan, and Weizhu Chen. 2024. CRITIC: Large language models can self-correct with tool-interactive critiquing. In *The Twelfth International Conference on Learning Representations*.

Michael Hanna, Ollie Liu, and Alexandre Variengien. 2023. How does gpt-2 compute greater-than?: Interpreting mathematical abilities in a pre-trained language model. In *Advances in Neural Information Processing Systems*, volume 36, pages 76033–76060. Curran Associates, Inc.

Michael Hanna, Sandro Pezzelle, and Yonatan Belinkov. 2024. Have faith in faithfulness: Going beyond circuit overlap when finding model mechanisms. In *ICML 2024 Workshop on Mechanistic Interpretability*.

Ruixin Hong, Hongming Zhang, Xinyu Pang, Dong Yu, and Changshui Zhang. 2024. A closer look at the self-verification abilities of large language models in logical reasoning. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 900–925, Mexico City, Mexico. Association for Computational Linguistics.

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. 2024. Large language models cannot self-correct reasoning yet. In *The Twelfth International Conference on Learning Representations*.

Dongwei Jiang, Jingyu Zhang, Orion Weller, Nathaniel Weir, Benjamin Van Durme, and Daniel Khashabi. 2024. Self-[in]correct: Llms struggle with refining self-generated responses. *CoRR*, abs/2404.04298.

Ryo Kamoi, Sarkar Snigdha Sarathi Das, Renze Lou, Jihyun Janice Ahn, Yilun Zhao, Xiaoxin Lu, Nan Zhang, Yusen Zhang, Haoran Ranran Zhang, Sujeeth Reddy Vummanthala, Salika Dave, Shaobo Qin, Arman Cohan, Wenpeng Yin, and Rui Zhang. 2024a. Evaluating LLMs at detecting errors in LLM responses. In *First Conference on Language Modeling*.

Ryo Kamoi, Yusen Zhang, Nan Zhang, Jiawei Han, and Rui Zhang. 2024b. When can LLMs actually correct their own mistakes? a critical survey of self-correction of LLMs. *Transactions of the Association for Computational Linguistics*, 12:1417–1440.

Loka Li, Guangyi Chen, Yusheng Su, Zhenhao Chen, Yixuan Zhang, Eric Xing, and Kun Zhang. 2024. Confidence matters: Revisiting intrinsic self-correction capabilities of large language models. *CoRR*, abs/2402.12563.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, and 1 others. 2024a. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*.

Guangliang Liu, Haitao Mao, Jiliang Tang, and Kristen Johnson. 2024b. Intrinsic self-correction for enhanced morality: An analysis of internal mechanisms and the superficial hypothesis. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 16439–16455, Miami, Florida, USA. Association for Computational Linguistics.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. 2023. Self-refine: Iterative refinement with self-feedback. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Kevin Meng, David Bau, Alex Andonian, and Yonatan Belinkov. 2022. Locating and editing factual associations in gpt. In *Advances in Neural Information Processing Systems*, volume 35, pages 17359–17372. Curran Associates, Inc.

Joseph Miller, Bilal Chughtai, and William Saunders. 2024. Transformer circuit evaluation metrics are not robust. In *First Conference on Language Modeling*.

Philipp Mondorf, Sondre Wold, and Barbara Plank. 2025. Circuit compositions: Exploring modular structures in transformer-based language models. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 14934–14955, Vienna, Austria. Association for Computational Linguistics.

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

Neel Nanda. 2024. Attribution patching. Neel Nanda's Website. Accessed: 2025-02-04.

Neel Nanda and Joseph Bloom. 2022. Transformerlens. `https://github.com/TransformerLensOrg/TransformerLens`.

Yaniv Nikankin, Anja Reusch, Aaron Mueller, and Yonatan Belinkov. 2024. Arithmetic without algorithms: Language models solve math with a bag of heuristics. *Preprint*, arXiv:2410.21272.

Judea Pearl. 2001. Direct and indirect effects. In *Proceedings of the Seventeenth Conference on Uncertainty in Artificial Intelligence*, UAI'01, page 411–420, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Long Phan, Alice Gatti, Ziwen Han, Nathaniel Li, Josephina Hu, Hugh Zhang, Sean Shi, Michael Choi, Anish Agrawal, Arnav Chopra, and 1 others. 2025. Humanity's last exam. *arXiv preprint arXiv:2501.14249*.

Tomohiro Sawada, Daniel Paleka, Alexander Havrilla, Pranav Tadepalli, Paula Vidas, Alexander Kranias, John Nay, Kshitij Gupta, and Aran Komatsuzaki. 2023. ARB: Advanced reasoning benchmark for large language models. In *The 3rd Workshop on Mathematical Reasoning and AI at NeurIPS'23*.

Alessandro Stolfo, Yonatan Belinkov, and Mrinmaya Sachan. 2023. A mechanistic interpretation of arithmetic reasoning in language models using causal mediation analysis. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 7035–7052, Singapore. Association for Computational Linguistics.

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

Gladys Tyen, Hassan Mansoor, Victor Carbune, Peter Chen, and Tony Mak. 2024. LLMs cannot find reasoning errors, but can correct them given the error location. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 13894–13908, Bangkok, Thailand. Association for Computational Linguistics.

Jesse Vig, Sebastian Gehrmann, Yonatan Belinkov, Sharon Qian, Daniel Nevo, Yaron Singer, and Stuart Shieber. 2020. Investigating gender bias in language models using causal mediation analysis. In *Advances in Neural Information Processing Systems*, volume 33, pages 12388–12401. Curran Associates, Inc.

Kevin Ro Wang, Alexandre Variengien, Arthur Conmy, Buck Shlegeris, and Jacob Steinhardt. 2023. Interpretability in the wild: a circuit for indirect object identification in GPT-2 small. In *The Eleventh International Conference on Learning Representations*.

Sean Welleck, Ximing Lu, Peter West, Faeze Brahman, Tianxiao Shen, Daniel Khashabi, and Yejin Choi. 2023. Generating sequences by learning to self-correct. In *The Eleventh International Conference on Learning Representations*.

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jialong Tang, Jialin Wang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Ma, and 43 others. 2024a. Qwen2 technical report. *CoRR*, abs/2407.10671.

An Yang, Beichen Zhang, Binyuan Hui, Bofei Gao, Bowen Yu, Chengpeng Li, Dayiheng Liu, Jianhong Tu, Jingren Zhou, Junyang Lin, and 1 others. 2024b. Qwen2.5-math technical report: Toward mathematical expert model via self-improvement. *arXiv preprint arXiv:2409.12122*.

Seonghyeon Ye, Yongrae Jo, Doyoung Kim, Sungdong Kim, Hyeonbin Hwang, and Minjoon Seo. 2023. Selfee: Iterative self-revising llm empowered by self-feedback generation. Blog post.

Fred Zhang and Neel Nanda. 2024. Towards best practices of activation patching in language models: Metrics and methods. In *The Twelfth International Conference on Learning Representations*.

Wei Zhang, Chaoqun Wan, Yonggang Zhang, Yiu-ming Cheung, Xinmei Tian, Xu Shen, and Jieping Ye. 2024. Interpreting and improving large language models in arithmetic calculation. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org.

Ruochen Zhao, Xingxuan Li, Shafiq Joty, Chengwei Qin, and Lidong Bing. 2023. Verify-and-edit: A knowledge-enhanced chain-of-thought framework. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5823–5840, Toronto, Canada. Association for Computational Linguistics.

A Reproducibility Statement

To ensure the reproducibility of our experiments, we make all code publicly available at: https://github.com/mainlp/validation-gap. Details of our circuit identification process are described in Section 3, Appendix B.3 and D.3. Documentation of our computational budget and the software we

use can be found in Appendix F. Furthermore, a detailed account of the data used in this work is provided in Section 3 and Appendix C.

B Circuit Discovery Details

B.1 Edge Attribution Patching

Attribution patching (Nanda, 2024), and specifically *edge attribution patching* (EAP) (Syed et al., 2024), is a computationally efficient linear approximation of activation patching to estimate the effect of interventions on latent activations. Following the activation patching terminology proposed by Zhang and Nanda (2024), consider a clean prompt X_{clean} and a corrupted prompt X_{corr}. EAP approximates the change in a predefined metric $\mathcal{P}$ on the model's output when a specific activation $\boldsymbol{z}$ is patched from its corrupted value $\boldsymbol{z}(X_{corr})$ to its clean value $\boldsymbol{z}(X_{clean})$. This approximation is formulated using a first-order Taylor expansion around the corrupted input X_{corr}. Specifically, EAP approximates $f_{\boldsymbol{z}}(X_{corr}; \boldsymbol{z}(X_{clean})) - f_{\boldsymbol{z}}(X_{corr}; \boldsymbol{z}(X_{corr}))$ as:

$$f_{\boldsymbol{z}}(X_{corr}; \boldsymbol{z}(X_{clean})) - f_{\boldsymbol{z}}(X_{corr}; \boldsymbol{z}(X_{corr})) \approx (\boldsymbol{z}(X_{clean}) - \boldsymbol{z}(X_{corr})) \cdot \frac{\partial f_{\boldsymbol{z}}}{\partial \boldsymbol{z}}\Big|_{\boldsymbol{z}=\boldsymbol{z}(X_{corr})}$$

where $f_{\boldsymbol{z}}(X, \boldsymbol{z}) = \mathcal{P}(M_{\boldsymbol{z}}(X; \boldsymbol{z}))$ represents the metric $\mathcal{P}$ applied to the patched model $M_{\boldsymbol{z}}$. Here, $M_{\boldsymbol{z}}(X; \boldsymbol{z})$ denotes the model M where the activation $\boldsymbol{z}$ is replaced with the value $\boldsymbol{z}$ for input X.

To compute the gradient $\frac{\partial f_{\boldsymbol{z}}}{\partial \boldsymbol{z}}\Big|_{\boldsymbol{z}=\boldsymbol{z}(X_{corr})}$, a backward pass is performed on the corrupted input X_{corr} with respect to the activation $\boldsymbol{z}$. The absolute value of the resulting score, often referred to as the *absolute attribution score* $|\nabla_{\boldsymbol{z}}\mathcal{P}| = |(\boldsymbol{z}(X_{clean}) - \boldsymbol{z}(X_{corr})) \cdot \frac{\partial f_{\boldsymbol{z}}}{\partial \boldsymbol{z}}\Big|_{\boldsymbol{z}=\boldsymbol{z}(X_{corr})}|$, quantifies the estimated influence of patching activation $\boldsymbol{z}$. This facilitates efficient circuit identification in EAP by ranking edges according to these scores.

B.2 Faithfulness Metric

Let $(X_{\text{clean}}, X_{\text{corr}})_i$ represent a pair of *clean* and *corrupt* prompts within a dataset of size N. For each input, let $\mathcal{C}(X_{i,clean})$ and $\mathcal{C}(X_{i,corr})$ denote the logits of the clean and corrupt answer tokens in the circuit's output, and let $M(X_{i,clean})$ and $M(X_{i,corr})$ be the corresponding logits for the full model. In this study, we define the faithfulness as the *logit difference recovered*:

$$\frac{1}{N}\sum_{i=1}^{N}\left(\frac{\mathcal{C}(X_{i,clean}) - \mathcal{C}(X_{i,corr})}{M(X_{i,clean}) - M(X_{i,corr})} \times 100\right) \tag{2}$$

A faithfulness score of 100% indicates that the circuit preserves the same logit difference as the full model, thus effectively recovering the model's task behavior.

B.3 Circuit Identification Process

To identify a minimal set of edges whose circuit achieves a faithfulness score between 99% and 101%, we employ an iterative search process. Starting from the sorted *absolute attribution scores*, denoted by $|\nabla_{\boldsymbol{z}}\mathcal{P}|$, we initially select the top-k edges and evaluate the corresponding circuit. In each iteration, we then add the next n edges from this sorted list and re-evaluate the faithfulness of the resulting circuit. The search stops as soon as a circuit with a faithfulness score (see Equation 2) within the desired interval (99% to 101%) is found. In our experiments, we set $k = 100$ and $n = 20$.

C Dataset

C.1 Templates

We generate our dataset of *clean* and *corrupt* prompts based on the templates in Table 9. These templates have a set of variables, namely `[instruction]`, `[person]`, `[object]`, `[verb]`, `[pronoun]`, `[num1]`, `[num2]`, `[num3]`, each of which can be assigned different values. Table 11 lists all possible values. For the numerical variables, we use single-digit numerical values (`[num1]` and `[num2]`) that add to a two-digit arithmetic result (`[num3]`). To ensure that each variable assignment occupies the same position in the token sequence within a template, we retain only variables that are tokenized as a single token for each model. For the `[instruction]` variable, we include only instructions that share the same number of tokens. Finally, for the `[correct_pair]` variable, we select assignments where labels are tokenized as a single token across models.

For other operations (subtraction, multiplication, and division) we follow the same procedure and construct eight templates for each operation (see Table 10). These templates use the same variable structure as addition, with subtraction including also a `[verb]` variable. For the numerical variables, we restrict number pairs to ensure two-digit

Model	Parameters	Layers	Hidden Dim	Num Heads	License
Qwen2.5-1.5B-Instruct	1.54B	28	1536	12	Apache
Qwen2.5-Math-1.5B-Instruct	1.54B	28	1536	12	Apache
Llama-3.2-3B-Instruct	3.21B	28	3072	24	Llama 3.2
Phi-3-Mini-4k-Instruct	3.82B	32	3072	32	MIT

Table 3: Properties of the models studied in this work. provide details on the number of parameters, layers, the hidden dimension size of the residual stream, and the number of attention heads for each model. Model weights were obtained from their respective Hugging Face repositories, accessible via the model names listed in the tables. Additionally, we specify the licenses under which the models are distributed.

results: subtraction uses 2-digit minus 1-digit inputs yielding a 2-digit result, multiplication uses 2-digit times 1-digit inputs yielding a 2-digit result, and division uses 2-digit dividends and 1-digit divisors producing 2-digit quotients.

C.2 Aligning Token Positions Across Templates

Since we employ token-specific EAP to identify relevant edges at specific token positions, the same element (e.g., the arithmetic result or the final numeric answer token) might appear at different token positions depending on the specific template (see Table 9). This variation in token positions makes it difficult to determine whether edges from two different template-specific circuits appear at semantically similar tokens (e.g., the arithmetic result in template 1 at token position 13 and the arithmetic result in template 2 at token position 16). To address this challenge, we assign shared abstract labels to corresponding elements across templates. Examples of such labels include `[op1-in-eq]`, `[op2-in-eq]`, `[equals]`, `[result-first]`, `[result-second]`, `[answer-first]`, and `[answer-second]`, which represent the two operands of the addition, the equal sign, and the digits of the arithmetic result and the numeric answer, respectively. Mapping tokens to a shared set of labels enables us to compute the *soft intersection circuits* between templates —allowing for the comparison of circuits associated with semantically equivalent elements without being confounded by their varying positions within the sequence.

D Experiment Details

D.1 Models

Details of the models used in this study are presented in Table 3. Specifically, we include information on the number of parameters, the number of layers, the size of the hidden model dimension, the number of attention heads per attention block, and the respective model licenses. All models are instruction-tuned and expect a series of special tokens (e.g., to indicate the beginning of the user prompt or the end of a turn). Therefore, we wrap all prompts in the respective chat templates of the models[7]. When applicable, we use the models' default system prompts.

D.2 Edge Overlap

To quantify the proportion of shared edges between two circuits, $\mathcal{C}_1$ and $\mathcal{C}_2$, we compute both the Intersection over Union (IoU) and the Intersection over Minimum (IoM).

As discussed in Section 3, we employ token-level EAP to identify relevant edges for each token position t in the prompt. Therefore, the set of edges $e_{ij}^{(t)} \in \mathcal{E} = \mathcal{C}$ is determined by the specific token position t. When computing the IoU and IoM between two circuits, the intersection and union of edge sets are computed separately for each token position. Specifically, we define the two metrics as:

$$\mathrm{IoU}(\mathcal{C}_{\text{result}}, \mathcal{C}_{\text{answer}}) = \frac{|\mathcal{C}_{\text{result}} \cap \mathcal{C}_{\text{answer}}|}{|\mathcal{C}_{\text{result}} \cup \mathcal{C}_{\text{answer}}|}, \quad (3)$$

$$\mathrm{IoM}(\mathcal{C}_{\text{result}}, \mathcal{C}_{\text{answer}}) = \frac{|\mathcal{C}_{\text{result}} \cap \mathcal{C}_{\text{answer}}|}{\min(|\mathcal{C}_{\text{result}}|, |\mathcal{C}_{\text{answer}}|)} \quad (4)$$

This provides an efficient way of measuring the degree of edge overlap at each token position in the analyzed circuits.

D.3 Circuits for Arithmetic Computation

To gain a deeper understanding of the relationship between models' mechanisms for arithmetic computation and validation, we identify an additional

[7] huggingface.co/docs/transformers/chat_templating.

	Model	Attention Heads
Consistency	Qwen-2.5-1.5B-Instruct	L12H0, L12H2, L12H10, L13H0, L13H1, L13H10
	Qwen-2.5-Math-1.5B-Instruct	L13H0, L13H1
	Llama-3.2-3B-Instruct	L4H14, L5H3, L7H11, L8H1, L10H5, L10H18
	Phi-3-Mini-4k-Instruct	L10H1, L10H5, L10H14, L14H19, L16H18
Random	Qwen-2.5-1.5B-Instruct	L1H0, L4H9, L9H6, L10H5, L11H8, L27H9
	Qwen-2.5-Math-1.5B-Instruct	L4H9, L27H5
	Llama-3.2-3B-Instruct	L1H1, L4H19, L9H16, L10H15, L11H23, L27H12
	Phi-3-Mini-4k-Instruct	L0H25, L3H18, L8H25, L20H19, L23H28

Table 4: Attention heads used in the patching experiment. *Consistency heads* refer to attention heads that demonstrate a behavior consistent with the pattern shown in Figure 5, assessing numerical alignment between the digits of the arithmetic result and the final numeric answer. In contrast, *random heads* are arbitrarily selected attention heads not classified as consistency heads, serving as a control group for comparison in the experiment.

set of circuits responsible for correctly computing the arithmetic result at the position of the equation (e.g., "5 + 8 = 13"). This process involves generating a new set of (*clean*, *corrupt*) prompt pairs for each template $\mathcal{T}_i \in \{\mathcal{T}_1, \ldots, \mathcal{T}_8\}$.

We construct these datasets based on the data samples used for identifying circuits for arithmetic error detection (as described in Section 3). Specifically, we first truncate both *clean* and *corrupt* prompts at the position of the equation sign (e.g., "'...5 + 8 ="). Next, we modify the *corrupt* prompt by replacing the numbers with a different set of numbers that produces an alternative result (e.g., "...3 + 9 ="). This process results in two prompts—neither containing any errors yet—where the next token is expected to be the correct outcome of the arithmetic equation (e.g., "13" for the *clean* prompt and "12" for the *corrupt* prompt). The corresponding labels represent the correct results for each prompt.

Using the new sets of *clean* and *corrupt* prompt pairs, we aim to identify the model components involved in computing the correct arithmetic result. We follow the same steps described in Section 3 and Appendix B.3 to identify circuits for each template. Finally, we compute the corresponding *soft intersection circuits*, which are responsible for generating the correct arithmetic result across templates.

D.4 Patching Consistency Heads

To evaluate the influence of individual consistency heads on the models' error detection behavior, we conduct two complementary patching interventions. In the first intervention, we run models on prompts X_{both}, which contain a consistent error at *both* the arithmetic result and the final numeric answer (for which models tend to incorrectly predict "valid"), and patch the latent activations of the consistency heads listed in Table 4 with activations from prompts X_{result}, where the error appears only at the arithmetic result (for which models typically predict "invalid"). This intervention is expected to increase the rate of "invalid" predictions. In the second, reverse intervention, we run models on prompts X_{result} and patch the same heads with activations from X_{both}. This setup is expected to decrease the rate of "invalid" predictions.

Specifically, for an attention head h, let $A^h(X)$ denote its attention matrix for a given prompt X. The patching operation we perform is defined as $A^h(X_{target}) = \alpha \cdot A^h(X_{source})$, where α is a scaling factor that controls the influence of the patched activation. For the first intervention, we set α to 3.1 for Qwen-2.5-1.5B-Instruct and Llama-3.2-3B-Instruct, and to 3.3 and 3.4 for Qwen-2.5-Math-1.5B-Instruct and Phi-3-Mini-4k-Instruct, respectively. For the second intervention, we set $\alpha = 1.0$ for all models. We perform the patching over $1,000$ prompt pairs. As a control setup for this experiment, we compare the result to patching randomly selected attention heads that are not labeled as consistency heads (see the full list in Table 4).

D.5 Linear Probes

We train linear probes on the hidden states of the models' residual stream to test whether a specific layer linearly encodes information about the correct result of the arithmetic equation within the prompts described in Section 3. The probes are trained

separately for each layer and a set of selected token positions. For each layer and token position, we use a training set of 500 hidden states per template (i.e., 4,000 samples in total per layer and token position) and a test set of 100 samples per template (i.e., 800 samples in total). The hidden states are collected from prompts where both the arithmetic result and the numeric answer contain consistent errors. All probes are trained for one epoch using the Adam optimizer with a learning rate of 0.001.

E Additional Results

In this section, we present the results of additional experiments we conducted.

E.1 Error Detection Circuits

As described in Section 4.1, we identify a circuit with faithfulness score between 99% and 101% for each template $\mathcal{T}_i \in \{\mathcal{T}_1, \dots \mathcal{T}_8\}$. Table 12 provides a comparison between the size of the identified circuits, their exact faithfulness scores, and the total number of edges in the full computational graph for all models and templates.

Once a circuit is identified for each template, we compute the *soft intersection circuit* $\mathcal{C}^{(\tau)}$ to derive a final circuit that generalizes across templates, as described in Section 3. For each model, we analyze the faithfulness scores and edge counts for different threshold values τ in the soft intersection circuit $\mathcal{C}^{(\tau)}$.

Figure 10 illustrates the faithfulness scores and number of edges for the *soft intersection circuit* $\mathcal{C}^{(\tau)}$ of Qwen-2.5-1.5B-Instruct across various threshold values τ. Specifically, Figure 10a displays values for the circuit responsible for detecting errors at the position of the arithmetic results, while Figure 10b shows values for an error at the final numeric answer. The red circles indicate the circuits that offer the best balance between faithfulness and size. For detecting arithmetic errors at the position of the arithmetic result, the optimal threshold is $\tau = \frac{5}{8}$, while for detecting errors at the final numeric answer, the strict intersection at $\tau = \frac{8}{8}$ provides the best trade-off. The corresponding circuits are visualized in Figures 26 and 30, respectively.

Similar results for Qwen-2.5-Math-1.5B-Instruct are shown in Figure 11, where the optimal circuits are $\mathcal{C}^{(7/8)}_{\text{result}}$ and $\mathcal{C}^{(8/8)}_{\text{answer}}$, depicted in Figures 27 and 31. For Llama-3.2-3B-Instruct, the corresponding results are displayed in Figure 12, with optimal circuits $\mathcal{C}^{(7/8)}_{\text{result}}$ and $\mathcal{C}^{(8/8)}_{\text{answer}}$, visualized in Figures 28 and 32. Finally, Figure 13 provides the results for Phi-3-Mini-4k-Instruct, where the best soft intersection circuits are $\mathcal{C}^{(7/8)}_{\text{result}}$ and $\mathcal{C}^{(6/8)}_{\text{answer}}$, shown in Figures 29 and 33.

Across all models and error types, a consistent structural pattern emerges. The most relevant edges are concentrated in the middle layers at the position of the final numeric answer. Additionally, a smaller subset of edges appears in the higher layers at the final token position of the prompt, primarily connecting MLP layers with the final residual output. Phi-3-Mini-4k-Instruct exhibits a slight variation, displaying a larger set of edges in the higher layers at the final token position. While many of these edges involve MLP components, others include attention head output matrices. Both types contribute to the residual stream forming the model's final output. Nonetheless, this model also exhibits a concentration of edges in the middle layers at the numeric answer position, consistent with the overall pattern observed in other models.

Model	IoU	IoM
Qwen-2.5-1.5B-Instruct	0.20	0.92
Qwen-2.5-Math-1.5B-Instruct	0.30	0.91
Llama-3.2-3B-Instruct	0.59	0.75
Phi-3-Mini-4k-Instruct	0.80	0.91

Table 5: The edge overlap between the two error detection circuits in terms of their Intersection over Union (IoU) and Intersection over Minimum (IoM). The metrics quantify the proportion of edges shared by both circuits while considering the token position in which an edge appears.

E.2 Edge Overlap of Error Detection Circuits

Table 5 shows the the Intersection over Union (IoU) and Intersection over Minimum (IoM) between the error detection circuits $\mathcal{C}_{\text{result}}$ and $\mathcal{C}_{\text{answer}}$ for each model. Notably, the IoM between the two circuits remains consistently ≥ 0.75 across all models. Meanwhile, IoU values exhibit greater variability, ranging from 0.20 for Qwen-2.5-1.5B-Instruct to 0.80 for Phi-3-Mini-4k-Instruct. This variability is primarily attributable to the size differences between circuits. Specifically, the circuits responsible for detecting errors at the position of the final numeric answer are generally smaller, with particularly pronounced size reductions for the Qwen family of models (refer to Table 12).

	Circuit	Qwen-2.5-1.5B	Qwen-2.5-Math-1.5B	Llama-3.2-3B	Phi-3-Mini-3.8B
Result	$\cap$	78.60 ± 7.46	47.10 ± 7.64	74.26 ± 5.67	73.02 ± 7.02
	$\cup$	100.3 ± 0.27	99.28 ± 0.57	97.71 ± 1.50	99.03 ± 0.34
Answer	$\cap$	82.59 ± 5.08	46.89 ± 7.87	74.07 ± 5.84	71.96 ± 6.86
	$\cup$	100.3 ± 0.27	98.80 ± 0.92	97.23 ± 0.93	99.18 ± 0.44
# Edges	$\cap$	49	20	83	235
	$\cup$	249	66	141	340

Table 6: Faithfulness scores for the intersection ($\cap$) and union ($\cup$) between the final *soft intersection circuits* $\mathcal{C}_{\text{result}}^{(\tau)}$ and $\mathcal{C}_{\text{answer}}^{(\tau)}$ computed for each model. For Qwen-2.5-1.5B-Instruct, the intersection and union between $\mathcal{C}_{\text{result}}^{(5/8)}$ and $\mathcal{C}_{\text{answer}}^{(8/8)}$ are calculated. For Qwen-2.5-Math-1.5B-Instruct, they are computed for $\mathcal{C}_{\text{result}}^{(7/8)}$ and $\mathcal{C}_{\text{answer}}^{(8/8)}$. For Llama-3.2-3B-Instruct, the intersection and union between $\mathcal{C}_{\text{result}}^{(7/8)}$ and $\mathcal{C}_{\text{answer}}^{(8/8)}$ are shown. For Phi-3-Mini-4k-Instruct, they are shown for $\mathcal{C}_{\text{result}}^{(7/8)}$ and $\mathcal{C}_{\text{answer}}^{(6/8)}$. The last two rows show the number of edges of the resulting circuits.

Table 6 reports the faithfulness results obtained from the intersection $\mathcal{C}_{\text{result}} \cap \mathcal{C}_{\text{answer}}$ and union $\mathcal{C}_{\text{result}} \cup \mathcal{C}_{\text{answer}}$ of the error detection circuits across models. The union circuits exhibit near-perfect faithfulness for both error types across all models, achieving faithfulness scores $\geq 97.00\%$. In contrast, the faithfulness of the intersection circuits is generally lower, although it remains above 70.00% for models such as Qwen-2.5-1.5B-Instruct, Llama-3.2-3B-Instruct, and Phi-3-Mini-4k-Instruct. The lowest faithfulness is observed for the intersection circuit of Qwen-2.5-Math-1.5B-Instruct, likely due to its extreme sparsity, containing only 20 edges in total.

E.3 Detection of Consistent Errors

As outlined in Section 4.2, we expect models to struggle with differentiating between error-free samples and those containing a consistent error in both the arithmetic result and the final numeric answer. We evaluate models on a dataset of 1,000 (*clean*, *corrupt*) prompt pairs for each template $\mathcal{T}_i \in \{\mathcal{T}_1, \ldots, \mathcal{T}_8\}$, where the *clean* prompts contain a consistent error at both specified positions. As mentioned in Section 4, a prompt pair is considered correctly classified if the model predicts the *clean* prompt as erroneous ($y_{clean} \in \{\text{invalid}, \text{incorrect}, \text{wrong}\}$) and the *corrupt* prompt as error-free ($y_{corrupt} \in \{\text{valid}, \text{correct}, \text{right}\}$). The respective accuracy of all models is summarized in Table 2. Overall, the results indicate that the models perform poorly on this task. For instance, Qwen-2.5-Math-1.5B-Instruct achieves an average accuracy of only $3.37\% \pm 2.06\%$. Among the evaluated models, Phi-3-Mini-4k-Instruct demonstrates the best performance, with an average accuracy of $40.98\% \pm 18.41\%$.

To study how these findings transfer to larger models, we additionally evaluate the performance of LLaMA-3.1-70B-Instruct and Qwen-2.5-32B-Instruct on different types of errors. As shown in Table 8, we observe that, similar to the smaller models, both larger LLMs struggle more with detecting a consistent error at both error positions (result & answer) than with detecting an error present at only the arithmetic result or the final numeric answer. In particular, the average accuracy of LLaMA-3.1-70B-Instruct for consistent errors is $44.67\% \pm 10.50\%$, a drop of more than half compared to its $100.0\% \pm 0.00\%$ accuracy on detecting errors present only at the position of the arithmetic result or the final numeric answer.

E.4 Consistency Heads

As mentioned in Section 4.2, we find that consistency heads play an important role in the model's arithmetic error detection process. These heads exhibit high average attention scores when the digits of the arithmetic result either align or misalign with the final numeric answer. Figures 14a and 14b illustrate examples of attention scores for two of these heads in Qwen-2.5-1.5B-Instruct. Similar patterns are presented for Qwen-2.5-Math-1.5B-Instruct in Figures 15a and 15b, for Llama-3.2-3B-Instruct in Figures 16a and 16b, and for Phi-3-Mini-4k-Instruct in Figures 17a and 17b. A comprehensive list of all identified consistency heads for each model is provided in Table 4.

Operation	Error Type	Qwen2.5-1.5B	Llama-3.2-3B	Phi-3-Mini-3.8B	Qwen2.5-Math-1.5B
Subtraction	*Arithmetic Result*	71.51 ± 11.50	99.66 ± 0.57	97.56 ± 5.74	93.54 ± 11.09
	Numeric Answer	70.05 ± 11.57	99.60 ± 0.65	97.60 ± 5.68	93.61 ± 10.77
	Result & Answer	8.68 ± 4.98	14.08 ± 2.01	99.64 ± 0.78	14.33 ± 1.60
Multiplication	*Arithmetic Result*	55.30 ± 19.46	78.53 ± 31.47	99.54 ± 0.73	94.00 ± 13.68
	Numeric Answer	55.22 ± 18.50	78.89 ± 31.11	99.54 ± 0.62	95.03 ± 13.12
	Result & Answer	62.88 ± 17.26	42.73 ± 30.73	99.65 ± 0.45	80.96 ± 9.25
Division	*Arithmetic Result*	59.86 ± 19.08	86.44 ± 13.04	94.52 ± 12.35	99.30 ± 0.92
	Numeric Answer	60.06 ± 18.07	86.76 ± 12.97	94.18 ± 12.72	99.53 ± 0.69
	Result & Answer	50.90 ± 15.84	11.79 ± 7.99	99.14 ± 1.50	57.95 ± 13.89

Table 7: Accuracy of models in correctly classifying the solutions' validity of (*clean*, *corrupt*) prompt pairs across different arithmetic operations and error types. Values represent the mean accuracy across all templates, reported with their corresponding standard deviation. Cells highlighted in red indicate cases where the accuracy for *Result & Answer* is notably lower than for *Numeric Answer* and *Arithmetic Result*.

E.5 Consistency Heads Patching

Figure 18 shows the models' accuracy in detecting consistent errors at both the position of the arithmetic result and the final numeric answer, before and after patching a small subset of *consistency* heads, as outlined in Section 4.3. For the exact list of patched heads, please refer to Table 4. Consistent with the results reported in Section 4.3 for Qwen-2.5-1.5B-Instruct, we observe a significant improvement in accuracy for Qwen-2.5-Math-1.5B-Instruct, Llama-3.2-3B-Instruct, and Phi-3-Mini-4k-Instruct.

E.6 Computation Circuits

As outlined in Section D.3, we identify the circuits responsible for predicting the correct arithmetic result of the equations in the prompts described in Section 3. Table 12 presents the size and faithfulness of the respective circuits for all models and templates. Results for the faithfulness scores and sizes of the *soft intersection circuits* $\mathcal{C}^{(\tau)}$ for different threshold values τ for all models are shown in Figures 10c, 11c, 12c, and 13c, respectively. The final circuits are visualized in Figures 22, 23, 24, and 25.

Error Type	Qwen-2.5-32B	Llama-3.1-70B
Arithmetic Result	98.93 ± 1.57	100.00 ± 0.00
Numeric Answer	99.51 ± 0.09	100.00 ± 0.00
Result & Answer	80.13 ± 9.65	44.76 ± 10.50

Table 8: Accuracy of bigger models in correctly classifying the solutions' validity of (*clean*, *corrupt*) prompt pairs with different error types. Values represent the mean accuracy across all templates, reported with their corresponding standard deviation.

E.7 Accuracy of Linear Probes

The results of the probing experiment, detailed in Section 4.3 and Appendix D.5, are presented for Qwen-2.5-Math-1.5B-Instruct, Llama-3.2-3B-Instruct, and Phi-3-Mini-4k-Instruct in Figure 20. The findings indicate that for all models, near-perfect accuracy is achieved by probes trained on the hidden representations of the residual stream in the upper layers. Notably, Phi-3-Mini-4k-Instruct is the only model that demonstrates significant probe accuracy in the middle layers.

E.8 Residual Stream Patching

To bridge the gap between the models' circuits responsible for arithmetic computation and validation, we add the hidden representation from higher layers—where the correct arithmetic result is linearly encoded (see Figure 20—to lower layers. Specifically, for Qwen-2.5-Math-1.5B-Instruct, we intervene on layer 1 using the hidden representation from layer 22. For Llama-3.2-3B-Instruct, the intervention is performed on layer 2 using the hidden representation from layer 16, while for Phi-3-Mini-4k-Instruct, layer 1 is modified using the hidden representation from layer 24. The results of these interventions are depicted in Figure 21.

Qwen-2.5-1.5B-Instruct, Qwen-2.5-Math-1.5B-Instruct, and Phi-3-Mini-4k-Instruct exhibit significant improvements in accuracy following these interventions. In contrast, Llama-3.2-3B-Instruct demonstrates a more modest performance gain. We attribute this difference to the simplicity of our approach and consider this a promising direction for further investigation.

F Implementation Details

For the majority of our circuit identification experiments, we used the `AutoCircuit` library developed by Miller et al. (2024). For the remaining experiments, we relied on the `TransformerLens` library by Nanda and Bloom (2022). All models were loaded with bfloat16 precision. The experiments were conducted on a single A100 GPU with 80GB of memory, consuming approximately 350 GPU hours in total. Additionally, GitHub Copilot was used as an assistant tool for parts of the project's source code development, and ChatGPT was used to correct minor grammatical errors.

Templates 1-8
[instruction] Problem: [person] has [num1] [object]. [pronoun] [verb] [num2] more [object]. How many [object] does [pronoun] have now? Reasoning: [person] has [num1] + [num2] = [num3] [object]. So, [pronoun] has [num3] [object] in total. Answer: The above reasoning is
[instruction] Problem: [person] starts with [num1] [object]. After [pronoun] [verb] [num2] more, how many [object] does [pronoun] have in total? Reasoning: To solve this, we add [num1] and [num2]: [num1] + [num2] = [num3]. Therefore, [person] now has [num3] [object]. Answer: The above reasoning is
[instruction] Problem: Initially, [person] possesses [num1] [object]. [pronoun] then [verb] [num2] additional [object]. What's the new total amount of [object] that [pronoun] has? Reasoning: We calculate: [num1] (original) + [num2] (added) = [num3] (total). So, [person] now has [num3] [object]. Answer: The above reasoning is
[instruction] Problem: [person]'s collection of [object] grows from [num1] to an unknown amount after [pronoun] [verb] [num2] more. Reasoning: To find the new total, we add: [num1] + [num2] = [num3] (final amount). Thus, [person] ends up with [num3] [object]. Answer: The above reasoning is
[instruction] Problem: [person] originally owns [num1] [object]. After [pronoun] [verb] [num2] additional [object], how many does [pronoun] have altogether? Reasoning: a simple addition gives us [num1] + [num2] = [num3]. Therefore, [person] has [num3] [object] now. Answer: The above reasoning is
[instruction] Problem: [person] possesses [num1] [object] at first. If [pronoun] [verb] [num2] more [object], what is the total count? Reasoning: Adding them gives: [num1] + [num2] = [num3]. Consequently, [person] has a total of [num3] [object]. Answer: The above reasoning is
[instruction] Problem: [num1] [object] belong to [person]. [pronoun] [verb] [num2] additional ones. What's the total? Reasoning: By addition, we get [num1] + [num2] = [num3]. Thus, [person] has [num3] [object] in total. Answer: The above reasoning is
[instruction] Problem: [person] begins with [num1] [object] and then [verb] [num2] more. How many [object] does [pronoun] have now? Reasoning: Let's add them up: [num1] + [num2] = [num3]. Therefore, [person] has a total of [num3] [object]. Answer: The above reasoning is

Table 9: The 8 problem templates including [instruction], [person], [object], [pronoun], [num1], [num2], [num3] as variable components. While samples within a template contain the same number of tokens, samples across templates vary in length due to differences in their non-variable parts.

Subtraction Templates 1-8	Multiplication Templates 1-8	Division Templates 1-8
[instruction] Problem: [person] has [num1] [object]. [pronoun] [verb] [num2] [object]. How many [object] does [pronoun] have now? Reasoning: [person] has [num1] - [num2] = [num3] [object]. So, [pronoun] has [num3] [object] remaining. Answer: The above reasoning is	[instruction] Problem: [person] has [num1] [object] per day. After [num2] days, how many [object] does [pronoun] have in total? Reasoning: [person] has [num1] × [num2] = [num3] [object]. So, [pronoun] has [num3] [object] in total. Answer: The above reasoning is	[instruction] Problem: [person] has [num1] [object]. [pronoun] wants to organize them into equal groups of [num2] [object] each. How many groups can [pronoun] make? Reasoning: [person] can make [num1] ÷ [num2] = [num3] groups. So, [pronoun] can make [num3] groups. Answer: The above reasoning is
[instruction] Problem: [person] starts with [num1] [object]. After [pronoun] [verb] [num2], how many [object] does [pronoun] have left? Reasoning: To solve this, we subtract [num2] from [num1]: [num1] - [num2] = [num3]. Therefore, [person] now has [num3] [object]. Answer: The above reasoning is	[instruction] Problem: [person] buys [num1] [object] each time [pronoun] goes shopping. If [pronoun] goes shopping [num2] times, how many [object] does [pronoun] buy in total? Reasoning: To solve this, we multiply [num1] and [num2]: [num1] × [num2] = [num3]. Therefore, [person] buys [num3] [object] in total. Answer: The above reasoning is	[instruction] Problem: [person] starts with [num1] [object]. If [pronoun] puts [num2] [object] in each container, how many containers can [pronoun] fill? Reasoning: To solve this, we divide [num1] by [num2]: [num1] ÷ [num2] = [num3]. Therefore, [person] can fill [num3] containers. Answer: The above reasoning is
[instruction] Problem: Initially, [person] possesses [num1] [object]. [pronoun] then [verb] [num2] [object]. What's the remaining amount of [object] that [pronoun] has? Reasoning: We calculate: [num1] (original) - [num2] (removed) = [num3] (remaining). So, [person] now has [num3] [object]. Answer: The above reasoning is	[instruction] Problem: [person] collects [num1] [object] each week. After [num2] weeks, how many [object] has [pronoun] collected altogether? Reasoning: We calculate: [num1] (per week) × [num2] (weeks) = [num3] (total). So, [pronoun] has collected [num3] [object] altogether. Answer: The above reasoning is	[instruction] Problem: [person] has [num1] [object] to share equally. If each person gets [num2] [object], how many people can receive [object]? Reasoning: We calculate: [num1] (total) ÷ [num2] (per person) = [num3] (people). So, [num3] people can receive [object]. Answer: The above reasoning is
[instruction] Problem: [person]'s collection of [object] decreases from [num1] to an unknown amount after [pronoun] [verb] [num2] [object]. Reasoning: To find the remaining total, we subtract: [num1] - [num2] = [num3] (final amount). Thus, [person] ends up with [num3] [object]. Answer: The above reasoning is	[instruction] Problem: Initially, [person] receives [num1] [object] each month. After [num2] months, what's the total amount of [object] that [pronoun] has received? Reasoning: We calculate: [num1] (per month) × [num2] (months) = [num3] (total). So, [person] has received [num3] [object]. Answer: The above reasoning is	[instruction] Problem: Initially, [person] possesses [num1] [object]. [pronoun] wants to arrange them in rows with [num2] [object] per row. What's the total number of rows that can be formed? Reasoning: We calculate: [num1] (total) ÷ [num2] (per row) = [num3] (rows). So, [person] can form [num3] rows. Answer: The above reasoning is
[instruction] Problem: [person] originally owns [num1] [object]. After [pronoun] [verb] [num2] [object], how many does [pronoun] have left? Reasoning: A simple subtraction gives us [num1] - [num2] = [num3]. Therefore, [person] has [num3] [object] remaining. Answer: The above reasoning is	[instruction] Problem: [person] originally gets [num1] [object] per visit. After [num2] visits, how many [object] has [pronoun] gotten altogether? Reasoning: A simple multiplication gives us [num1] × [num2] = [num3]. Therefore, [person] has gotten [num3] [object] in total. Answer: The above reasoning is	[instruction] Problem: [person] originally owns [num1] [object]. If [pronoun] distributes [num2] [object] to each recipient, how many recipients can get [object]? Reasoning: A simple division gives us [num1] ÷ [num2] = [num3]. Therefore, [num3] recipients can get [object]. Answer: The above reasoning is
[instruction] Problem: [person] possesses [num1] [object] at first. If [pronoun] [verb] [num2] [object], what is the remaining count? Reasoning: Subtracting them gives: [num1] - [num2] = [num3]. Consequently, [person] has [num3] [object] left. Answer: The above reasoning is	[instruction] Problem: [person] earns [num1] [object] per task at first. If [pronoun] completes [num2] tasks, what is the total count of [object]? Reasoning: Multiplying them gives: [num1] × [num2] = [num3]. Consequently, [pronoun] earns [num3] [object] in total. Answer: The above reasoning is	[instruction] Problem: [person] possesses [num1] [object] at first. If [pronoun] arranges [num2] [object] in each pile, what is the total number of piles? Reasoning: Dividing them gives: [num1] ÷ [num2] = [num3]. Consequently, [person] can make [num3] piles. Answer: The above reasoning is
[instruction] Problem: [num1] [object] belong to [person]. [pronoun] [verb] [num2] of them. What's the remainder? Reasoning: By subtraction, we get [num1] - [num2] = [num3]. Thus, [person] has [num3] [object] remaining. Answer: The above reasoning is	[instruction] Problem: [person] finds [num1] [object] each time [pronoun] searches. After [num2] searches, what's the total? Reasoning: By multiplication, we get [num1] × [num2] = [num3]. Thus, [person] finds [num3] [object] in total. Answer: The above reasoning is	[instruction] Problem: [num1] [object] belong to [person]. [pronoun] places [num2] [object] in each box. What's the total number of boxes needed? Reasoning: By division, we get [num1] ÷ [num2] = [num3]. Thus, [person] needs [num3] boxes. Answer: The above reasoning is
[instruction] Problem: [person] begins with [num1] [object] and then [verb] [num2] of them. How many [object] does [pronoun] have left? Reasoning: Let's subtract them: [num1] - [num2] = [num3]. Therefore, [person] has [num3] [object] remaining. Answer: The above reasoning is	[instruction] Problem: [person] produces [num1] [object] per session and then has [num2] sessions. How many [object] are there in total? Reasoning: Let's multiply them: [num1] × [num2] = [num3]. Therefore, there are [num3] [object] altogether. Answer: The above reasoning is	[instruction] Problem: [person] begins with [num1] [object] and then organizes [num2] [object] per shelf. How many shelves does [pronoun] need? Reasoning: Let's divide them: [num1] ÷ [num2] = [num3]. Therefore, [person] needs [num3] shelves. Answer: The above reasoning is

Table 10: Templates used for other arithmetic operations. We created 8 templates for each operation, including [instruction], [person], [object], [pronoun], [num1], [num2], [num3] as variable components. The subtraction template also has a [verb] variable, for which we use the following verbs: "lost", "sold", "gave away", "donated", "threw away",

Variable	Assignments
[person]	Aaron, Adam, Alan, Alex, Alice, Amy, Anderson, Andre, Andrew, Andy, Anna, Anthony, Arthur, Austin, Blake, Brandon, Brian, Carter, Charles, Charlie, Christian, Christopher, Clark, Cole, Collins, Connor, Crew, Crystal, Daniel, David, Dean, Edward, Elizabeth, Emily, Eric, Eva, Ford, Frank, George, Georgia, Graham, Grant, Henry, Ian, Jack, Jacob, Jake, James, Jamie, Jane, Jason, Jay, Jennifer, Jeremy, Jessica, John, Jonathan, Jordan, Joseph, Joshua, Justin, Kate, Kelly, Kevin, Kyle, Laura, Leon, Lewis, Lisa, Louis, Luke, Madison, Marco, Marcus, Maria, Mark, Martin, Mary, Matthew, Max, Michael, Michelle, Morgan, Patrick, Paul, Peter, Prince, Rachel, Richard, River, Robert, Roman, Rose, Ruby, Russell, Ryan, Sarah, Scott, Sean, Simon, Stephen, Steven, Sullivan, Taylor, Thomas, Tyler, Victoria, Warren, William
[object]	apples, bananas, oranges, grapes, pears, mangoes, strawberries, blueberries, cherries, pineapples, lemons, watermelons, kiwis, peaches, plums, books, pens, notebooks, flowers, candies, gifts, toys, bottles, tickets, clothes, shoes, hats, gloves, keys, wallets, phones, laptops, tablets, cameras, headphones, glasses, watches, rings, necklaces, bracelets, purses, backpacks, umbrellas, mugs, plates, bowls, forks, spoons, knives, chairs, tables, lamps, blankets, pillows, towels, socks, scarves, jackets, belts, bookshelves, paintings, mirrors, candles, frames
[verb]	won, bought, received, gained, obtained, earned, acquired, collected, accumulated, gathered, got
[correct_pair]	"valid or invalid", "correct or incorrect", "right or wrong"

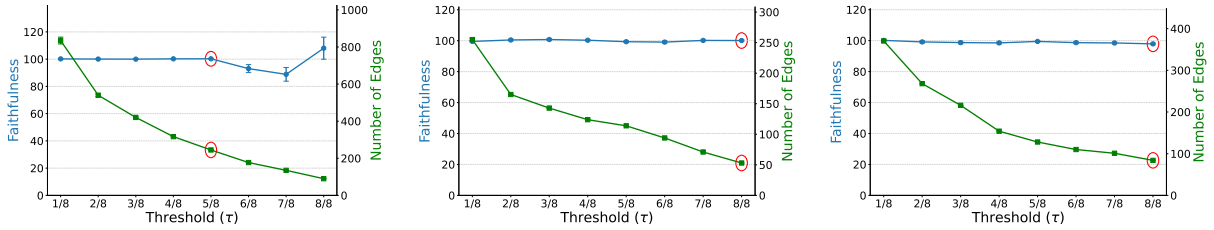
[instruction]
Does the following reasoning chain contain any mistakes? Determine whether it is [correct_pair].
Does the reasoning chain provided have any errors? Decide whether it is [correct_pair].
Does the given reasoning chain contain any flaws? Evaluate whether it is [correct_pair].
Does the reasoning chain shown have any errors? Verify whether it is [correct_pair].
Does the reasoning chain below have any mistakes? Check if it is [correct_pair].
Does the following reasoning chain have any errors? Specify whether it is [correct_pair].
Does the provided reasoning chain contain any flaws? Assess if it is [correct_pair].
Does the reasoning chain presented have any issues? Judge whether it is [correct_pair].
Does the reasoning chain contain any mistakes? Examine if it is [correct_pair].
Does the reasoning chain have any errors? Inspect it and determine if it is [correct_pair].
Does the reasoning chain have any flaws? Review it and confirm if it is [correct_pair].
Does the given reasoning chain contain any issues? Analyze it and decide if it is [correct_pair].

Table 11: The full set of variables and possible values that are used to create the data. [instruction] is the only variable that contains [correct_pair] as another variable part.

	Template	Task	Total Edges	Num Edges	Faithfulness
Qwen-2.5-1.5B-Instruct	1	Inv. Result	84715	281	100.00
		Inv. Answer	84715	101	100.57
		Computation	84715	140	99.04
	2	Inv. Result	84715	440	100.40
		Inv. Answer	84715	115	100.00
		Computation	84715	100	99.21
	3	Inv. Result	84715	522	100.00
		Inv. Answer	84715	120	100.00
		Computation	84715	300	99.13
	4	Inv. Result	84715	261	100.49
		Inv. Answer	84715	208	100.57
		Computation	84715	104	99.20
	5	Inv. Result	84715	721	100.00
		Inv. Answer	84715	120	99.29
		Computation	84715	280	100.00
	6	Inv. Result	84715	142	99.47
		Inv. Answer	84715	163	100.41
		Computation	84715	120	99.15
	7	Inv. Result	84715	200	99.42
		Inv. Answer	84715	104	100.67
		Computation	84715	260	99.60
	8	Inv. Result	84715	201	99.52
		Inv. Answer	84715	112	100.60
		Computation	84715	140	99.15
Qwen-2.5-Math-1.5B-Instruct	1	Inv. Result	84715	141	99.46
		Inv. Answer	84715	200	99.45
		Computation	84715	101	99.27
	2	Inv. Result	84715	241	100.00
		Inv. Answer	84715	101	100.00
		Computation	84715	100	99.27
	3	Inv. Result	84715	340	100.00
		Inv. Answer	84715	100	100.00
		Computation	84715	101	99.56
	4	Inv. Result	84715	320	99.01
		Inv. Answer	84715	100	100.00
		Computation	84715	120	99.26
	5	Inv. Result	84715	318	99.42
		Inv. Answer	84715	102	99.40
		Computation	84715	122	99.61
	6	Inv. Result	84715	102	99.47
		Inv. Answer	84715	187	99.45
		Computation	84715	100	99.53
	7	Inv. Result	84715	321	99.53
		Inv. Answer	84715	323	99.53
		Computation	84715	100	99.59
	8	Inv. Result	84715	321	99.42
		Inv. Answer	84715	100	100.00
		Computation	84715	100	99.55

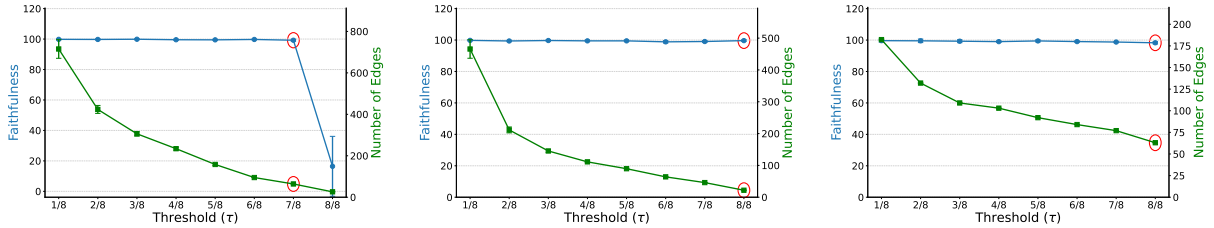
	Template	Task	Total Edges	Num Edges	Faithfulness
Llama-3.2-3B-Instruct	1	Inv. Result	389971	190	99.28
		Inv. Answer	389971	294	100.00
		Computation	389971	160	99.10
	2	Inv. Result	389971	380	99.50
		Inv. Answer	389971	282	100.00
		Computation	389971	100	99.14
	3	Inv. Result	389971	187	99.10
		Inv. Answer	389971	280	99.52
		Computation	389971	100	100.00
	4	Inv. Result	389971	180	99.32
		Inv. Answer	389971	382	99.25
		Computation	389971	118	99.59
	5	Inv. Result	389971	220	99.58
		Inv. Answer	389971	241	100.00
		Computation	389971	160	99.22
	6	Inv. Result	389971	443	99.28
		Inv. Answer	389971	240	100.00
		Computation	389971	100	100.00
	7	Inv. Result	389971	180	99.04
		Inv. Answer	389971	561	100.52
		Computation	389971	180	99.09
	8	Inv. Result	389971	221	99.21
		Inv. Answer	389971	200	99.19
		Computation	389971	119	99.53
Phi-3-Mini-4k-Instruct	1	Inv Result	1592881	285	99.25
		Inv. Answer	1592881	581	99.24
		Computation	1592881	161	99.46
	2	Inv. Result	1592881	500	99.25
		Inv. Answer	1592881	480	99.25
		Computation	1592881	122	99.08
	3	Inv. Result	1592881	855	99.24
		Inv. Answer	1592881	683	99.25
		Computation	1592881	240	99.47
	4	Inv. Result	1592881	371	99.15
		Inv. Answer	1592881	504	99.13
		Computation	1592881	144	99.08
	5	Inv. Result	1592881	504	99.21
		Inv. Answer	1592881	500	99.20
		Computation	1592881	145	99.49
	6	Inv. Result	1592881	569	99.22
		Inv. Answer	1592881	422	99.22
		Computation	1592881	140	99.42
	7	Inv. Result	1592881	886	99.20
		Inv. Answer	1592881	605	99.21
		Computation	1592881	120	99.03
	8	Inv. Result	1592881	625	99.22
		Inv. Answer	1592881	481	99.23
		Computation	1592881	146	99.50

Table 12: The faithfulness score and the number of edges of the circuit identified for each model, template, and task. To better compare circuit sizes, we also present the total number of edges ***per token position*** for each model. Left: Qwen models (Qwen-2.5 and Qwen-2.5-Math) from templates 1–8; Right: Llama-3.2 and Phi-3-Mini models from templates 1–8.



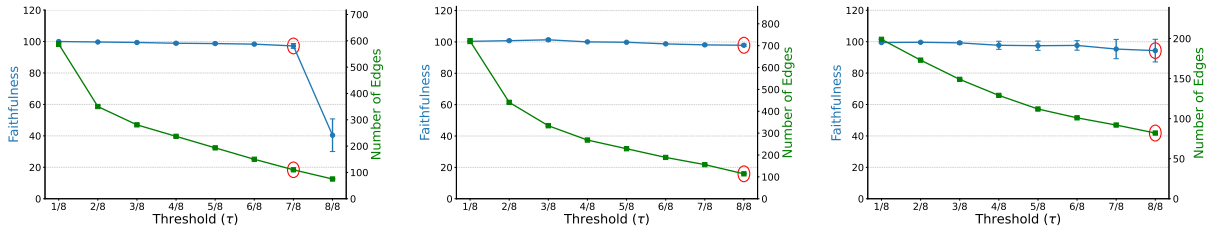
(a) Invalid Result (b) Invalid Answer (c) Computation

Figure 10: The number of edges and average faithfulness scores of the *soft intersection circuit* for different threshold values, τ. Red circles indicate the soft intersection circuit that best trade offs size with faithfulness. Results are shown for Qwen-2.5-1.5B-Instruct.



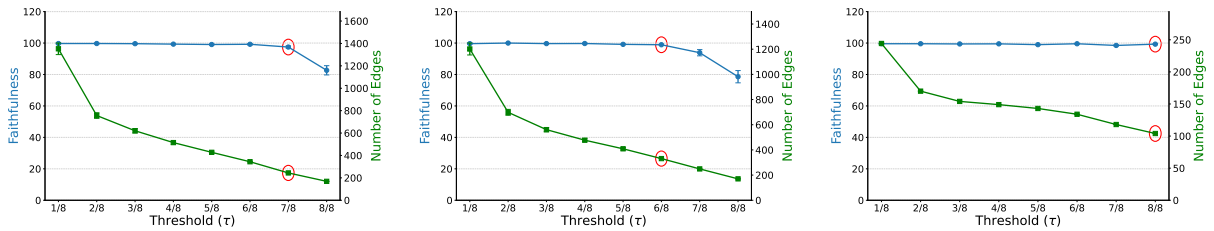
(a) Invalid Result (b) Invalid Answer (c) Computation

Figure 11: The number of edges and average faithfulness scores of the *soft intersection circuit* for different threshold values, τ. Red circles indicate the soft intersection circuit that best trade offs size with faithfulness. Results are shown for Qwen-2.5-Math-1.5B-Instruct.



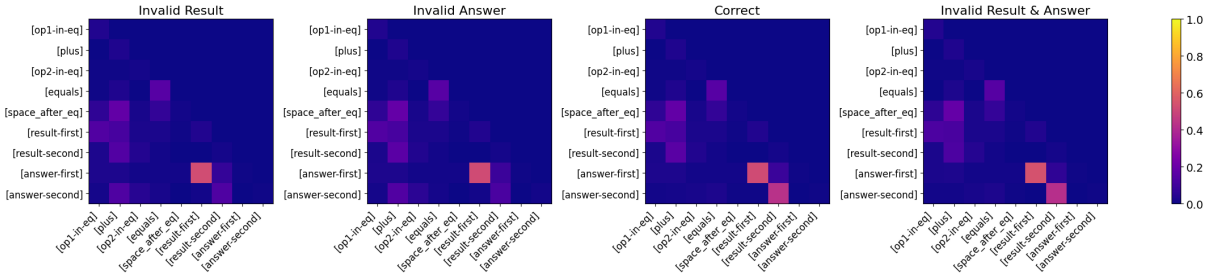
(a) Invalid Result (b) Invalid Answer (c) Computation

Figure 12: The number of edges and average faithfulness scores of the *soft intersection circuit* for different threshold values, τ. Red circles indicate the soft intersection circuit that best trade offs size with faithfulness. Results are shown for Llama-3.2-3B-Instruct.

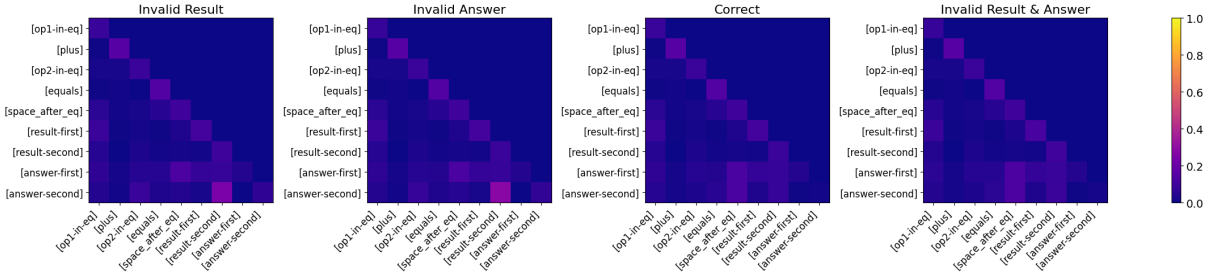


(a) Invalid Result (b) Invalid Answer (c) Computation

Figure 13: The number of edges and average faithfulness scores of the *soft intersection circuit* for different threshold values, τ. Red circles indicate the soft intersection circuit that best trade offs size with faithfulness. Results are shown for Phi-3-Mini-4k-Instruct.

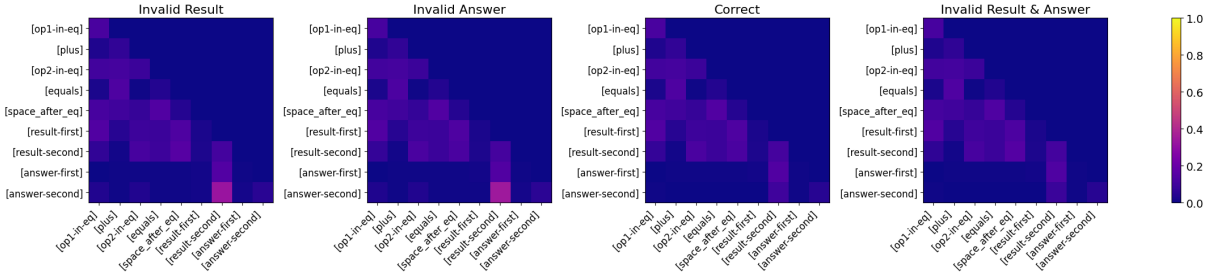


(a) consistency head L12H2.

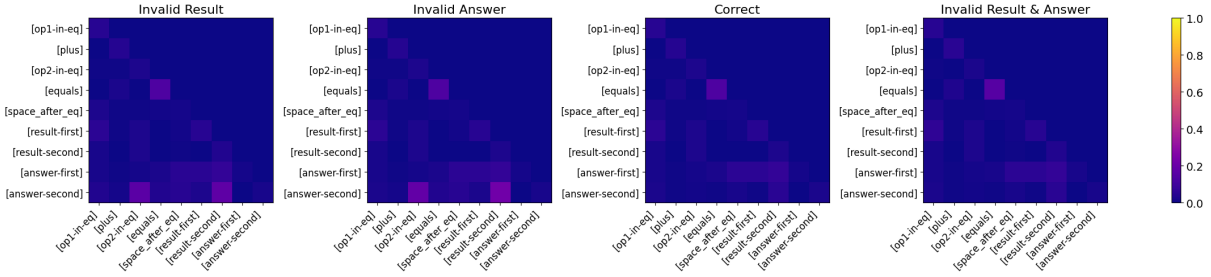


(b) (in)consistency head L13H1.

Figure 14: Attention patterns of two *consistency heads* in Qwen-2.5-1.5B-Instruct. Reported scores are averaged over 5,000 prompts where (*left*) an error is present at the position of the arithmetic result, (*second to left*) an error is present at the position of the final numeric answer, (*second to right*) no error is present, and (*right*) a consistent error is present at both considered positions.

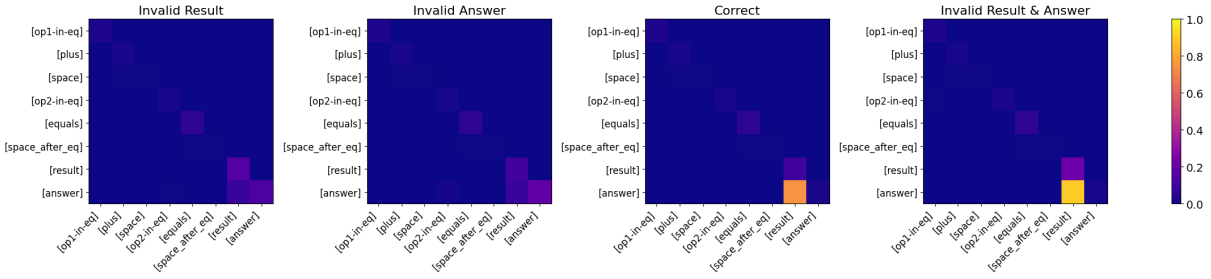


(a) (in)consistency head L13H0.

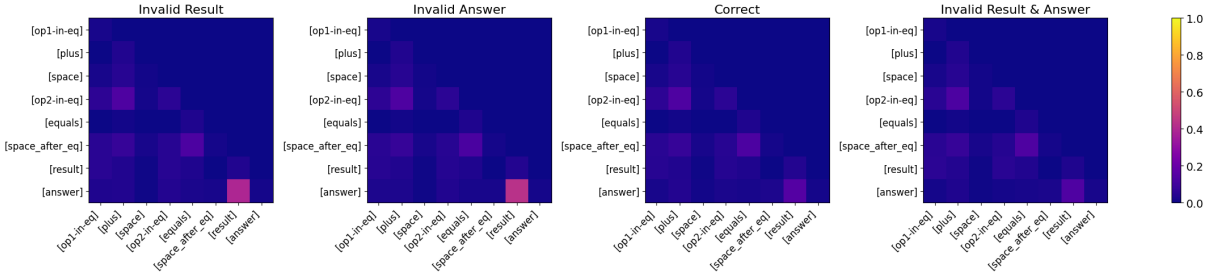


(b) (in)consistency head L13H1.

Figure 15: Attention patterns of two *consistency heads* in Qwen-2.5-Math-1.5B-Instruct. Reported scores are averaged over 5,000 prompts where (*left*) an error is present at the position of the arithmetic result, (*second to left*) an error is present at the position of the final numeric answer, (*second to right*) no error is present, and (*right*) a consistent error is present at both considered positions.

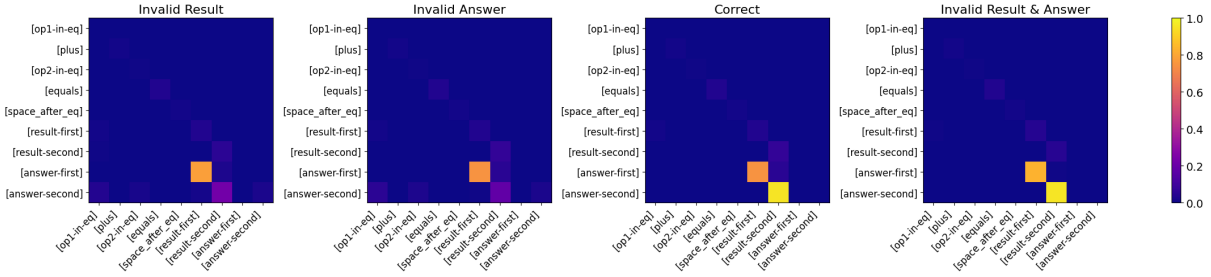


(a) consistency head L4H14.

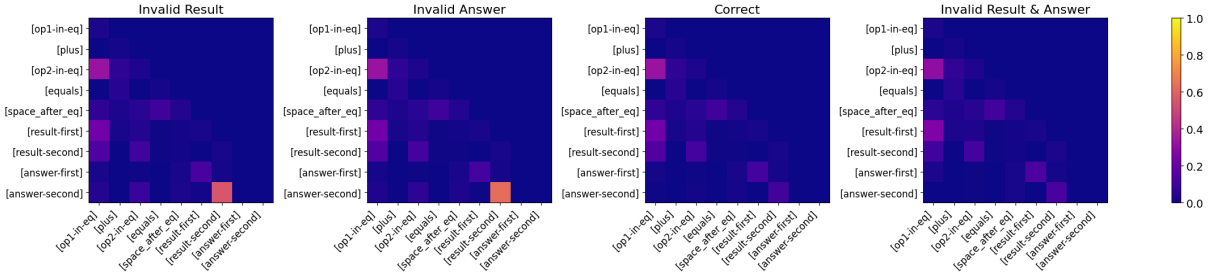


(b) (in)consistency head L8H1.

Figure 16: Attention patterns of two *consistency heads* in Llama-3.2-3B-Instruct. Reported scores are averaged over 5,000 prompts where (*left*) an error is present at the position of the arithmetic result, (*second to left*) an error is present at the position of the final numeric answer, (*second to right*) no error is present, and (*right*) a consistent error is present at both considered positions. Note that Llama-3.2 does not tokenize numbers digit-by-digit.

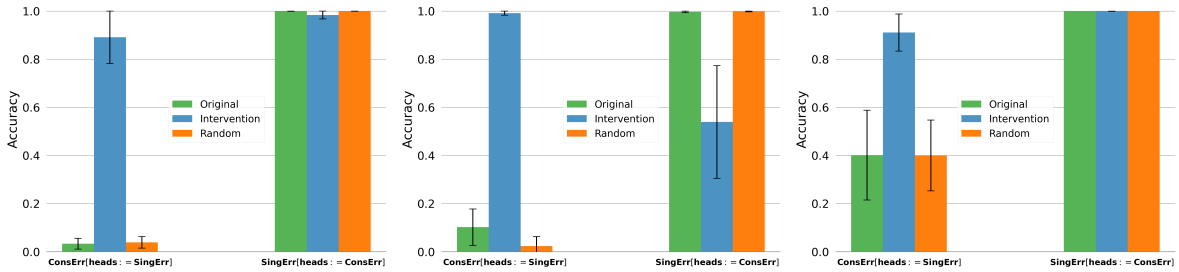


(a) consistency head L10H1.



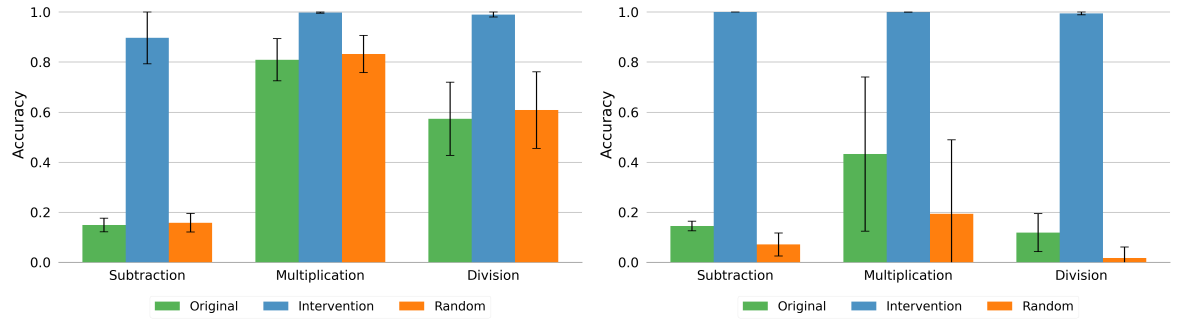
(b) (in)consistency head L10H14.

Figure 17: Attention patterns of two *consistency heads* in Phi-3-Mini-4k-Instruct. Reported scores are averaged over 5,000 prompts where (*left*) an error is present at the position of the arithmetic result, (*second to left*) an error is present at the position of the final numeric answer, (*second to right*) no error is present, and (*right*) a consistent error is present at both considered positions.



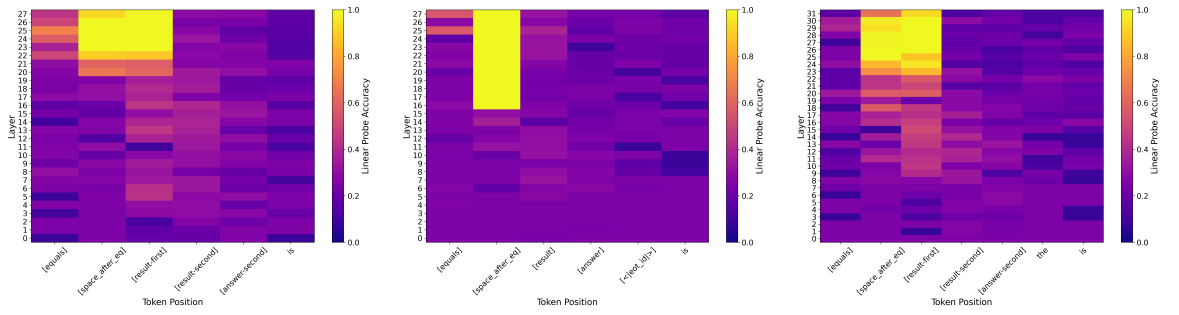
(a) Qwen-2.5-Math-1.5B-Instruct (b) Llama-3.2-3B-Instruct (c) Phi-3-Mini-4k-Instruct

Figure 18: Accuracy of models on (*clean*, *corrupt*) prompt pairs, where the *clean* prompt contains a consistent error at both error positions. The blue intervention bar represents the result after patching a set of *consistency heads* (for a list of heads, please refer to Table 4 in the Appendix). In contrast, the orange bar indicates the accuracy after patching a set of randomly chosen attention heads that are not labeled as consistency heads.



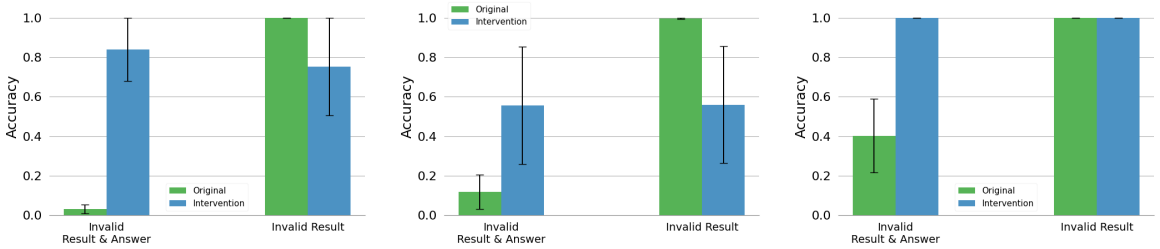
(a) Qwen-2.5-Math-1.5B-Instruct (b) Llama-3.2-3B-Instruct

Figure 19: Accuracy of models on (*clean*, *corrupt*) prompt pairs for other arithmetic operations, where the *clean* prompt contains a consistent error at both error positions. The blue intervention bar represents the result after patching a set of *consistency heads* (for a list of heads, please refer to Table 4 in the Appendix). In contrast, the red bar indicates the accuracy after patching a set of randomly chosen attention heads that are not labeled as consistency heads.



(a) Qwen-2.5-Math-1.5B-Instruct (b) Llama-3.2-3B-Instruct (c) Phi-3-Mini-4k-Instruct

Figure 20: The linear probe's accuracy across all layers of Qwen-2.5-Math-1.5B-Instruct (Figure 20a), Llama-3.2-3B-Instruct (Figure 20b), and Phi-3-Mini-4k-Instruct (Figure 20c) at selected token positions. Only in higher layers, the probe is able to achieve predict the correct arithmetic result perfectly. For Qwen-2.5-Math-1.5B-Instruct and Phi-3-Mini-4k-Instruct, we observe moderate accuracies also in middle layers.



(a) Qwen-2.5-Math-1.5B-Instruct (b) Llama-3.2-3B-Instruct (c) Phi-3-Mini-4k-Instruct

Figure 21: Accuracy of models on (*clean*, *corrupt*) prompt pairs where (*left*) the *clean* prompt contains a consistent error at both error positions (invalid result & answer), and (*right*) an error is present only at the position of the arithmetic result (invalid result). The blue intervention bar denotes the result after adding the hidden representation of the residual stream in layer higher layers (at [result-first]) to the residual stream in lower layers (at [result-second]). For Qwen-2.5-Math-1.5B-Instruct (Figure 21a), the residual steams' hidden representation from layer 22 is added to the one in layer 1. For Llama-3.2-3B-Instruct (Figure 21b), we add the representation from layer 16 to layer 2, and for Phi-3-Mini-4k-Instruct (Figure 21c), the representation from layer 24 is added to layer 1.

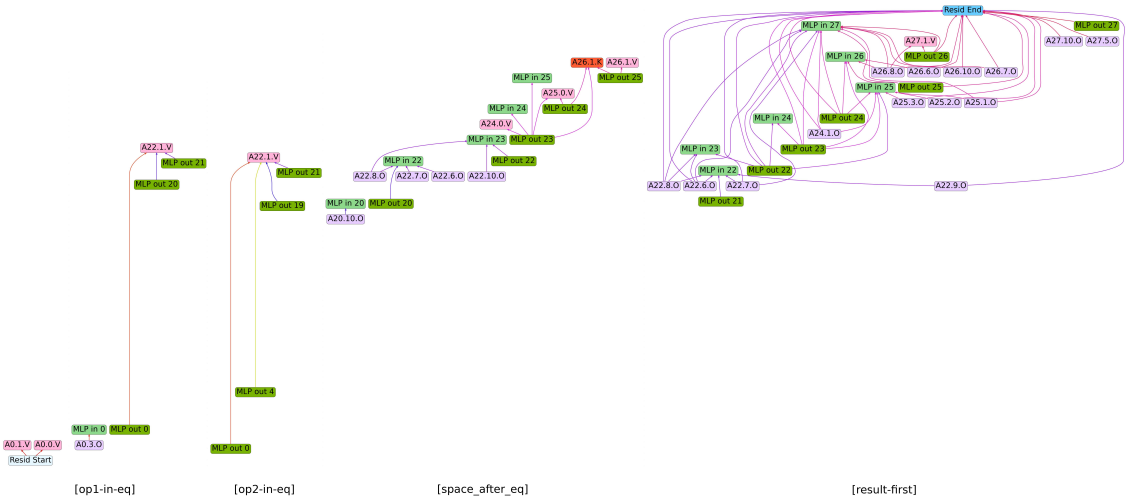


Figure 22: The computation circuit $\mathcal{C}^{(8/8)}_{\text{computation}}$ of Qwen-2.5-1.5B-Instruct obtained after taking the soft intersection between all template circuits with a threshold value of $\tau = \frac{8}{8}$.

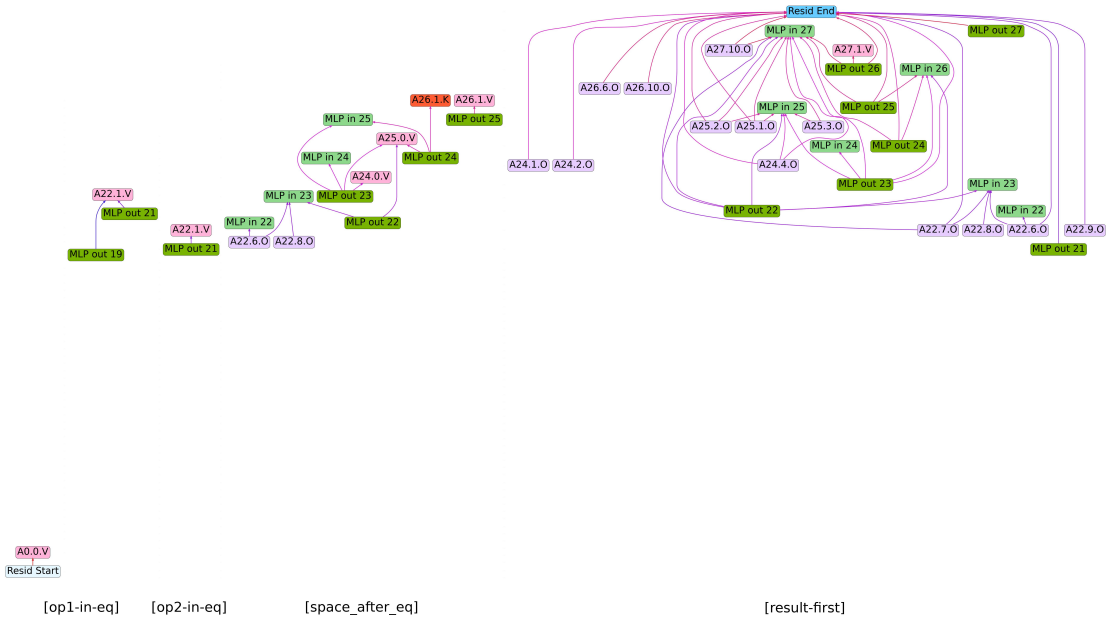


Figure 23: The computation circuit $\mathcal{C}^{(8/8)}_{\text{computation}}$ of Qwen-2.5-Math-1.5B-Instruct obtained after taking the soft intersection between all template circuits with a threshold value of $\tau = \frac{8}{8}$.

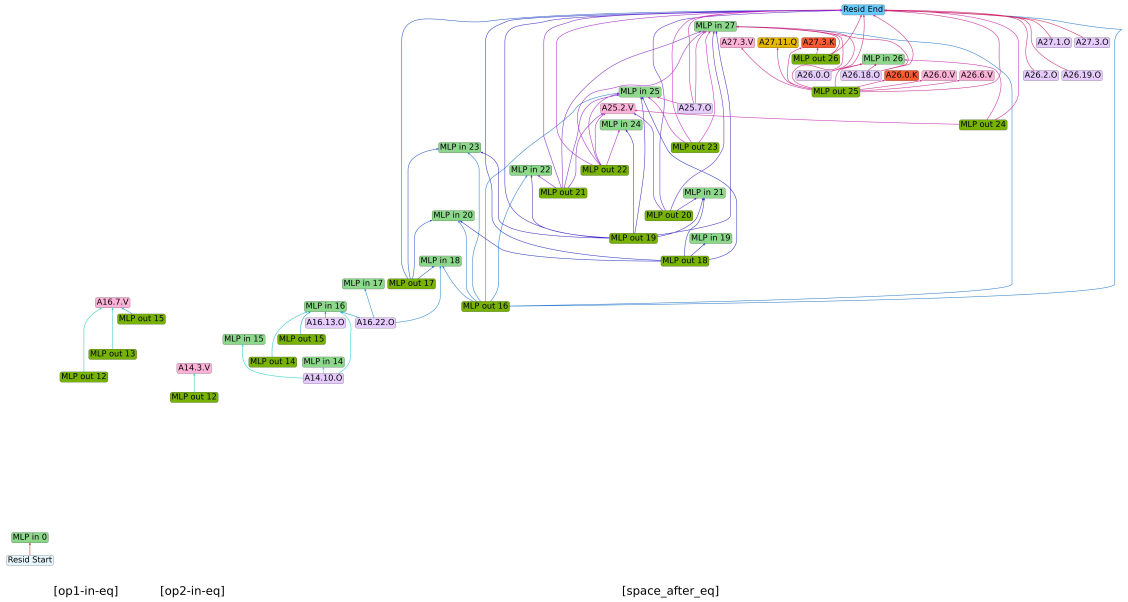


Figure 24: The computation circuit $\mathcal{C}^{(8/8)}_{\text{computation}}$ of Llama-3.2-3B-Instruct obtained after taking the soft intersection between all template circuits with a threshold value of $\tau = \frac{8}{8}$.

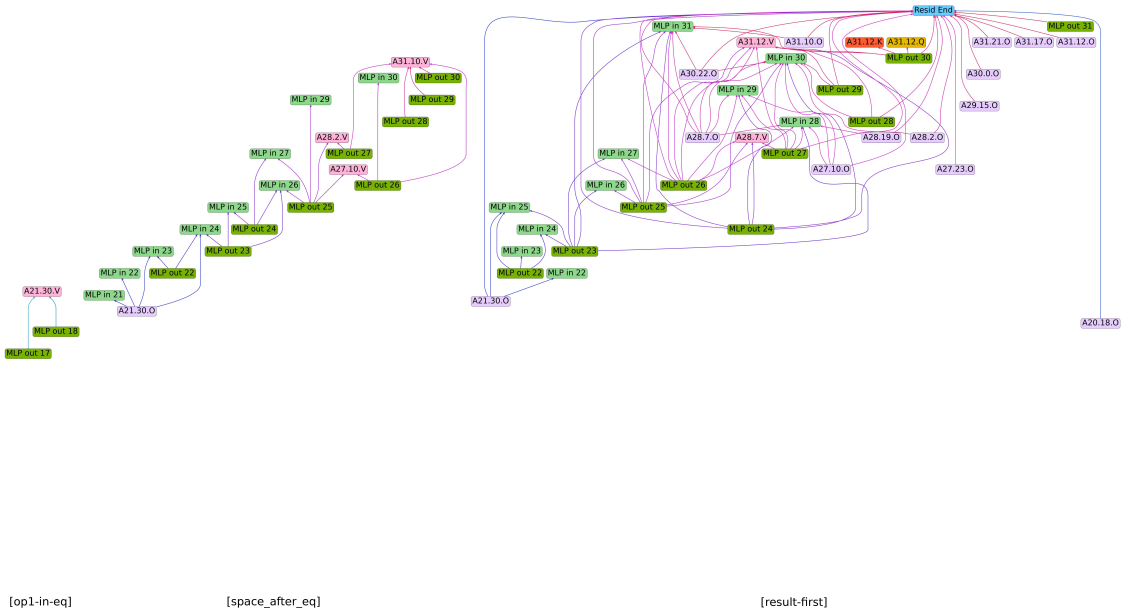


Figure 25: The computation circuit $\mathcal{C}^{(8/8)}_{\text{computation}}$ of Phi-3-Mini-4k-Instruct obtained after taking the soft intersection between all template circuits with a threshold value of $\tau = \frac{8}{8}$.

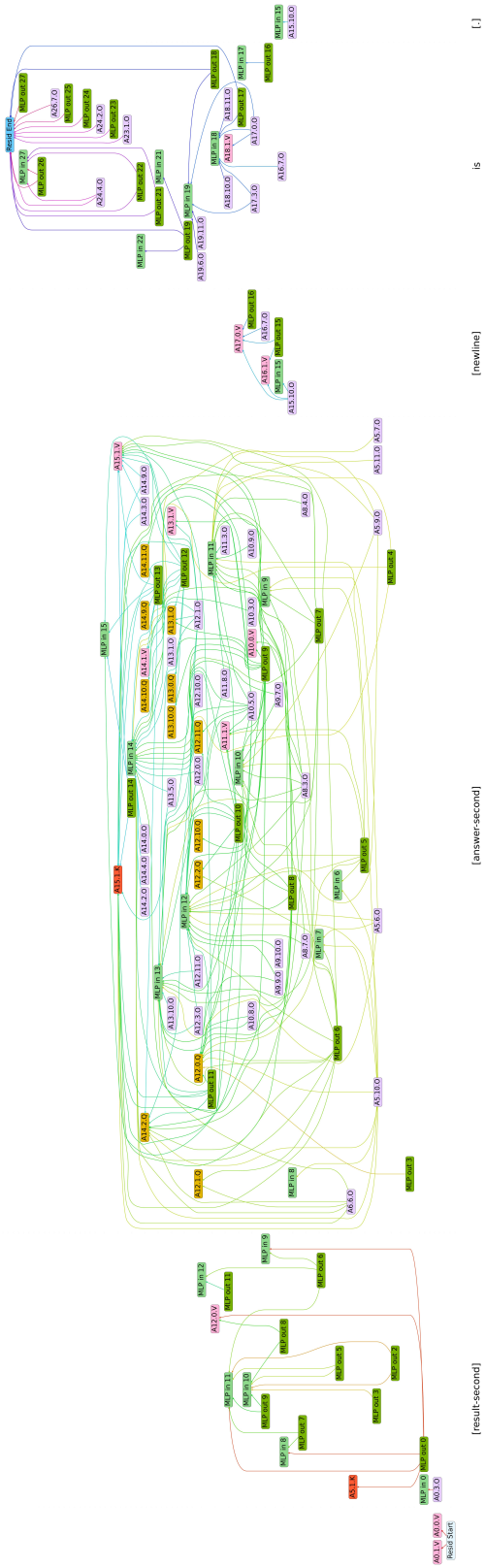


Figure 26: The arithmetic result error identification circuit $\mathcal{C}_{\text{result}}^{(5/8)}$ of Qwen-2.5-1.5B-Instruct obtained after taking the soft intersection between all template circuits with a threshold value of $\tau = \frac{5}{8}$.

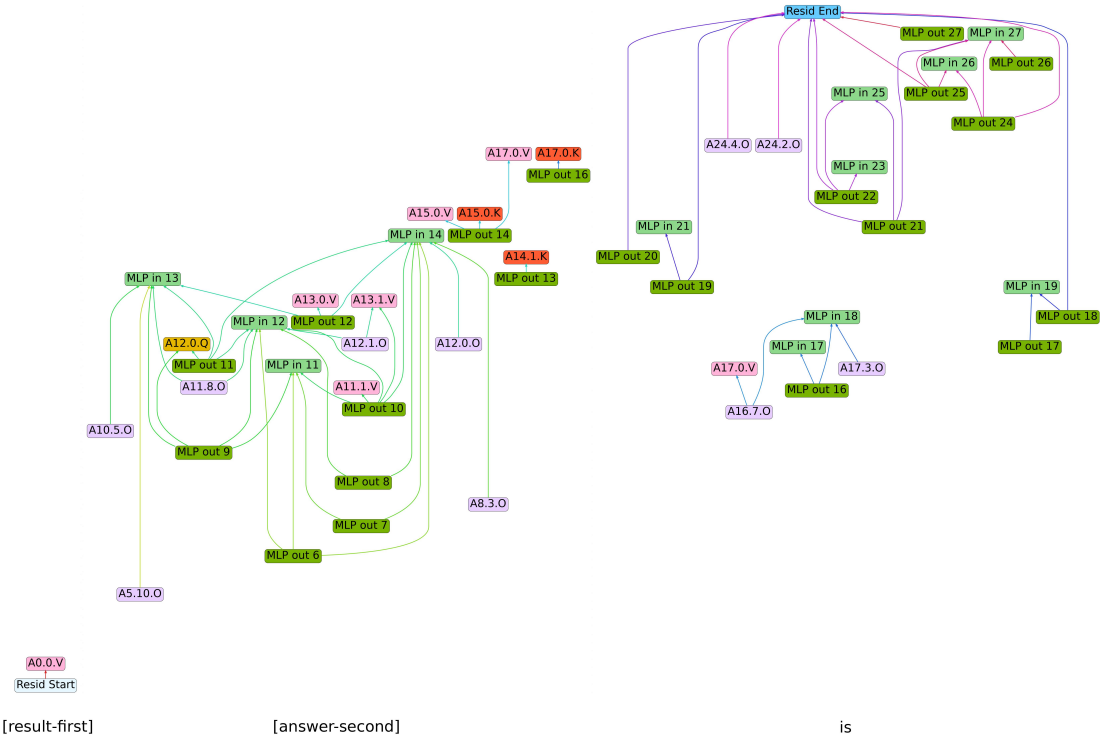


Figure 27: The arithmetic result error identification circuit $\mathcal{C}_{\text{result}}^{(7/8)}$ of Qwen-2.5-Math-1.5B-Instruct obtained after taking the soft intersection between all template circuits with a threshold value of $\tau = \frac{7}{8}$.

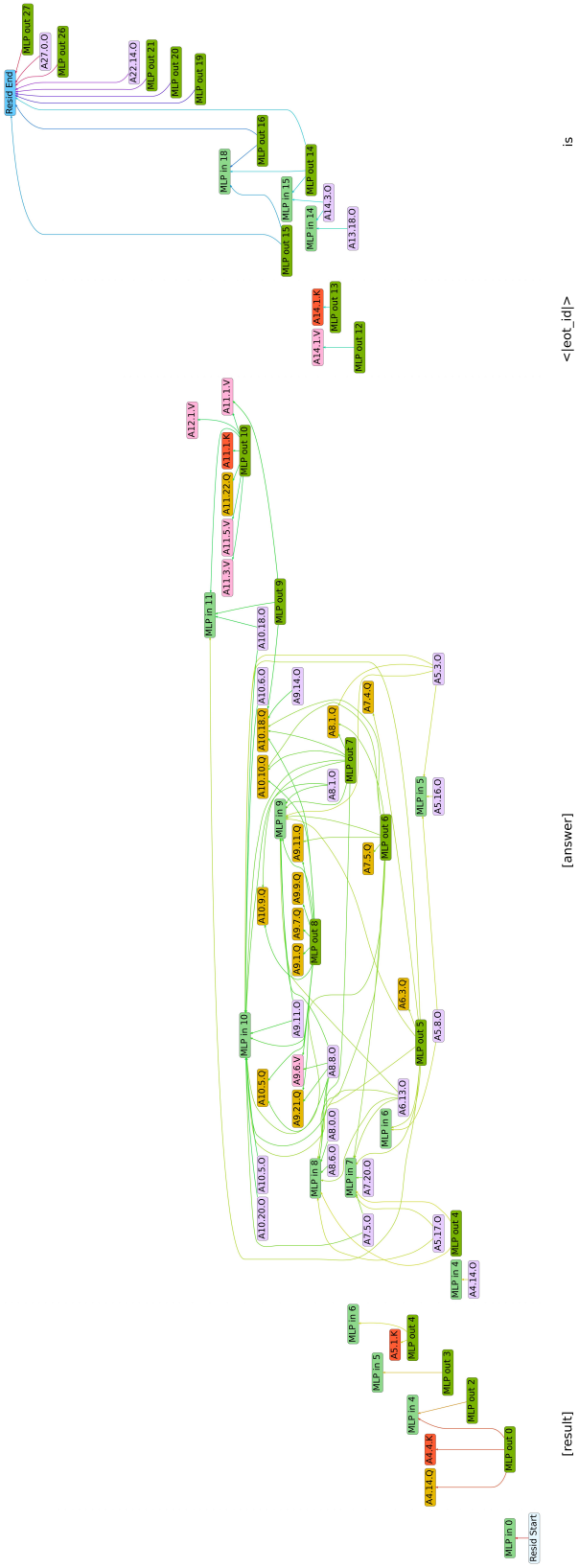


Figure 28: The arithmetic result error identification circuit $\mathcal{C}_{\text{result}}^{(7/8)}$ of Llama-3.2-3B-Instruct obtained after taking the soft intersection between all template circuits with a threshold value of $\tau = \frac{7}{8}$.

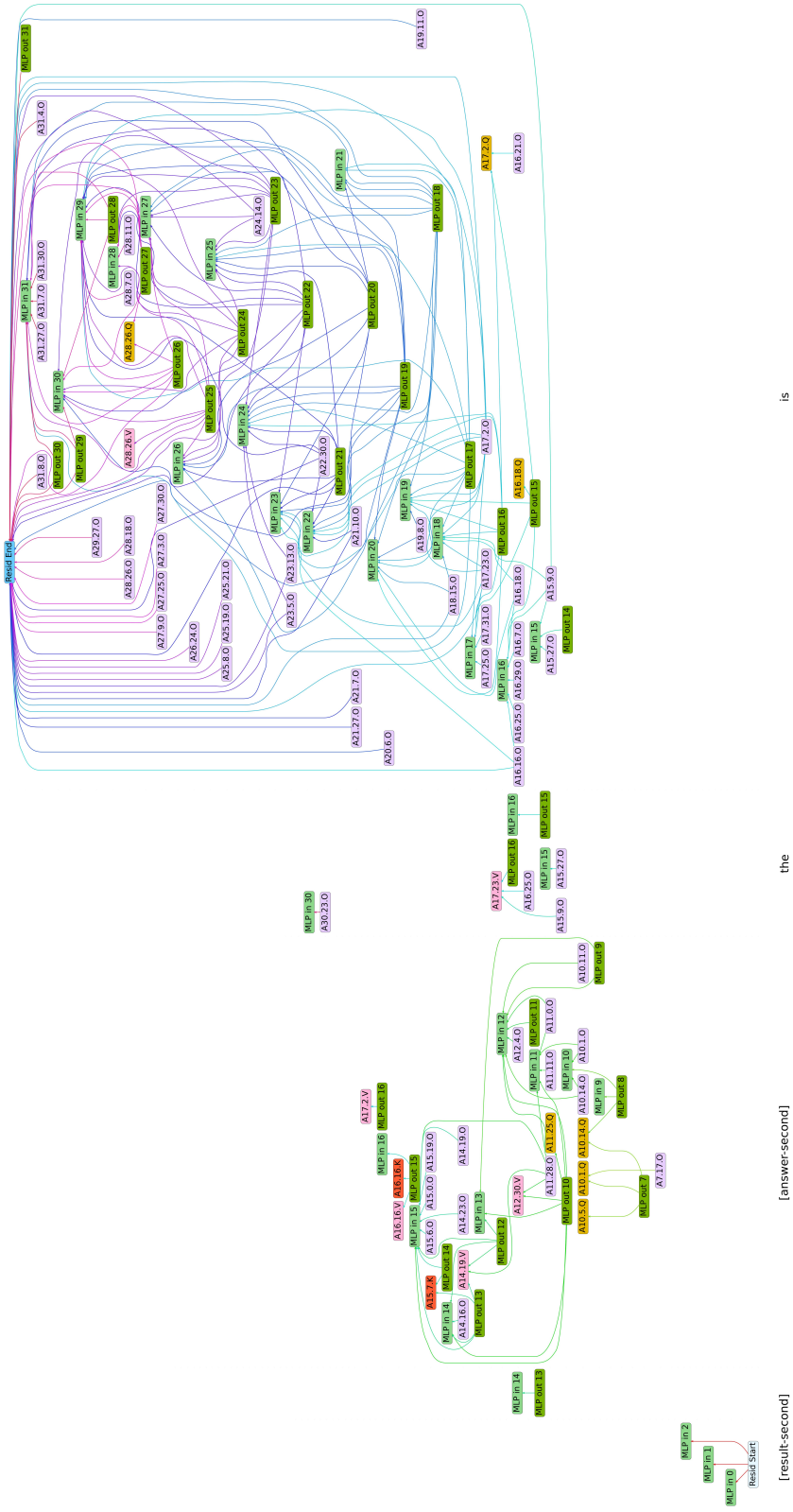


Figure 29: The arithmetic result error identification circuit $\mathcal{C}_{\text{result}}^{(7/8)}$ of Phi-3-Mini-4k-Instruct obtained after taking the soft intersection between all template circuits with a threshold value of $\tau = \frac{7}{8}$.

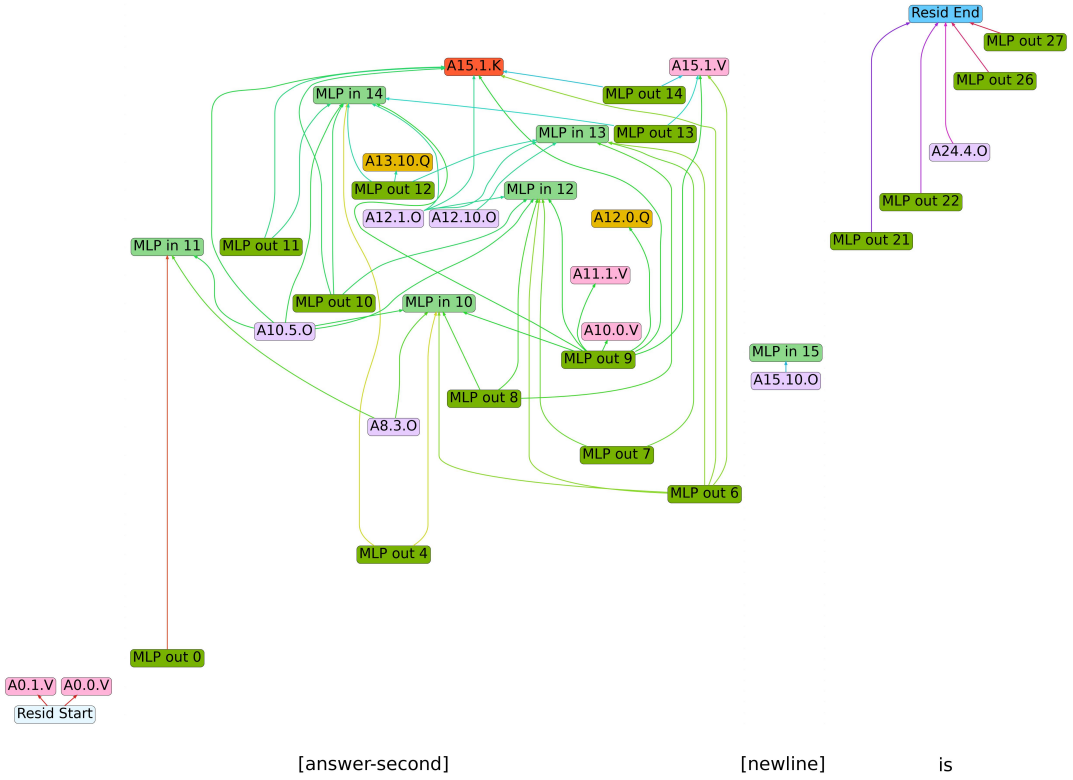


Figure 30: The numeric answer error identification circuit $\mathcal{C}_{\text{answer}}^{(8/8)}$ of Qwen-2.5-1.5B-Instruct obtained after taking the soft intersection between all template circuits with with a threshold value of $\tau = \frac{8}{8}$.

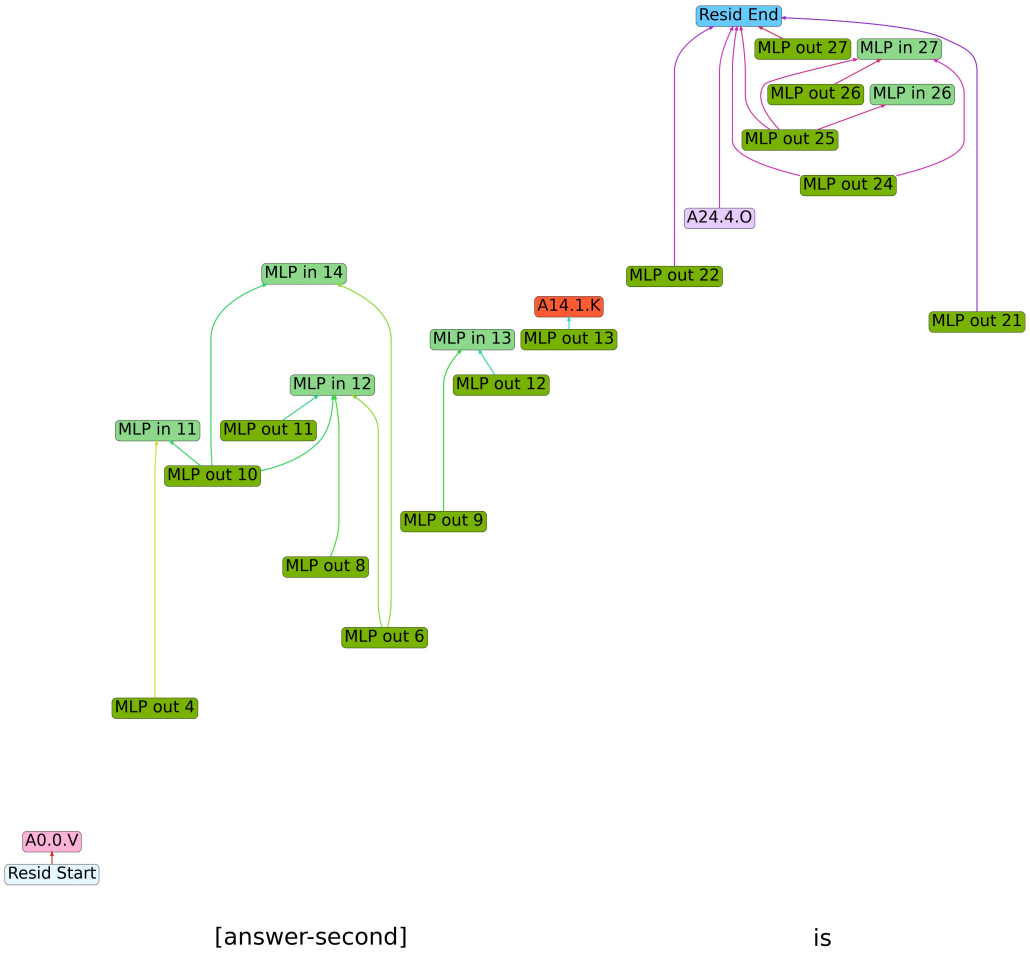


Figure 31: The numeric answer error identification circuit $\mathcal{C}_{\text{answer}}^{(8/8)}$ of Qwen-2.5-Math-1.5B-Instruct obtained after taking the soft intersection between all template circuits with with a threshold value of $\tau = \frac{8}{8}$.

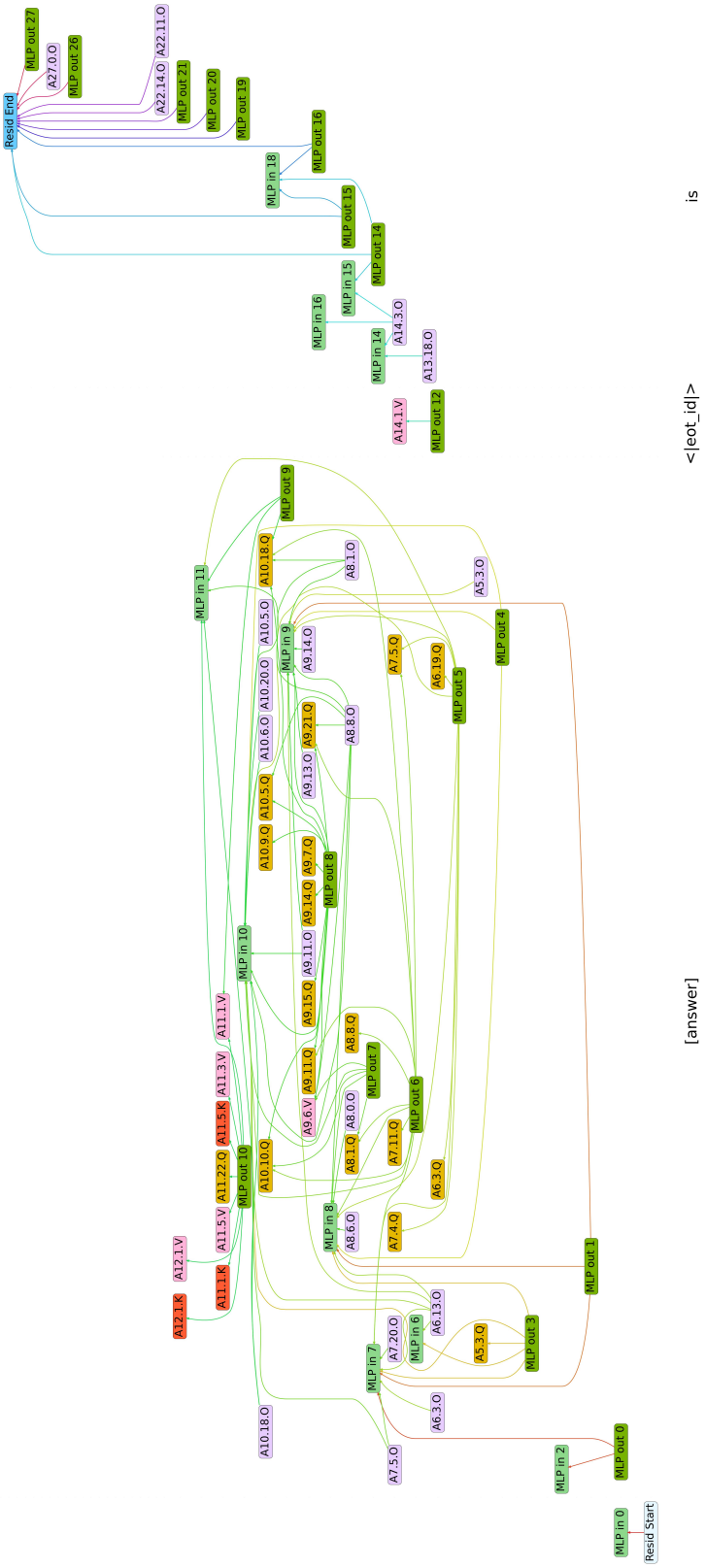


Figure 32: The numeric answer error identification circuit $\mathcal{C}^{(8/8)}_{\text{answer}}$ of Llama-3.2-3B-Instruct obtained after taking the soft intersection between all template circuits with with a threshold value of $\tau = \frac{8}{\infty}$.

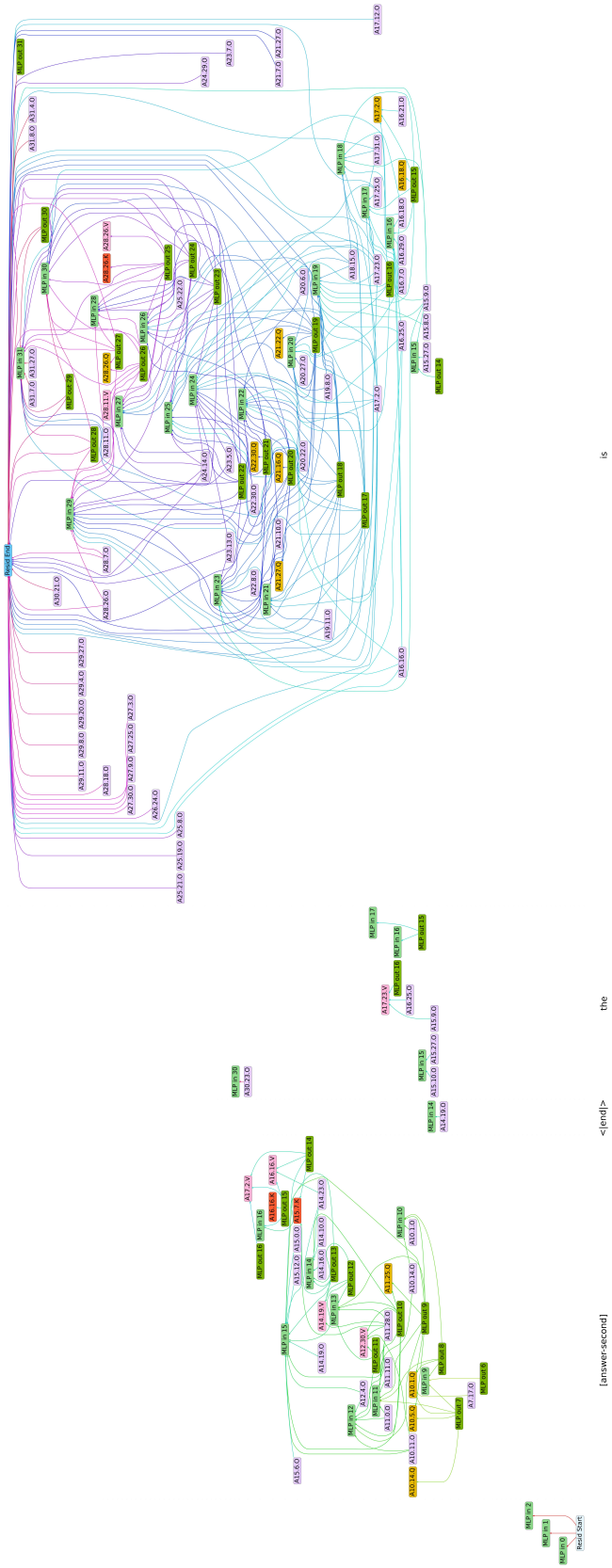


Figure 33: The numeric answer error identification circuit $\mathcal{C}_{\text{answer}}^{(6/8)}$ of Phi-3-Mini-4k-Instruct obtained after taking the soft intersection between all template circuits with with a threshold value of $\tau = \frac{6}{8}$.