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

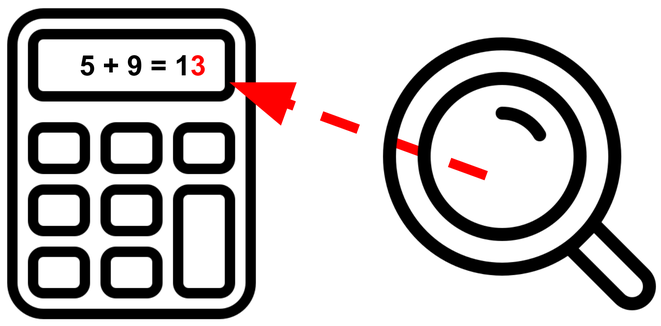
Leonardo Bertolazzi[1], Philipp Mondorf[2,3], Barbara Plank[2,3], Raffaella Bernardi[1]
[1]DISI and CIMeC, University of Trento, [2]MaiNLP, LMU, Munich, [3]MCML, Munich

ICT DAYS

CIMeC · MaiNLP · LMU

UNIVERSITÀ DI TRENTO
Dipartimento di Ingegneria e Scienza dell'Informazione

John has 5 eggs and he buys 9 more at the supermarket, how many eggs does he have in total?



Motivation

- Large language models are now capable of complex problem-solving, such as mathematics and competitive programming
- However, they exhibit limited backtracking abilities and often fail to detect errors in their own outputs [1]
- Despite this, there has been limited effort to reverse-engineer why models struggle with error detection

What mechanisms drive the detection of simple arithmetic errors in language models?

Data and Models

We consider simple aritmetic problem involving a single addition and three types of errors:

1) errors in the **arithmetic result**

Problem: Anne has 6 apples. She gets 7 more apples. How many apples does she have now? Reasoning: Anne has 6 + 7 = **17** apples. So, she has **13** apples in total. Answer: The above reasoning is → **incorrect**

2) errors in the **final answer**

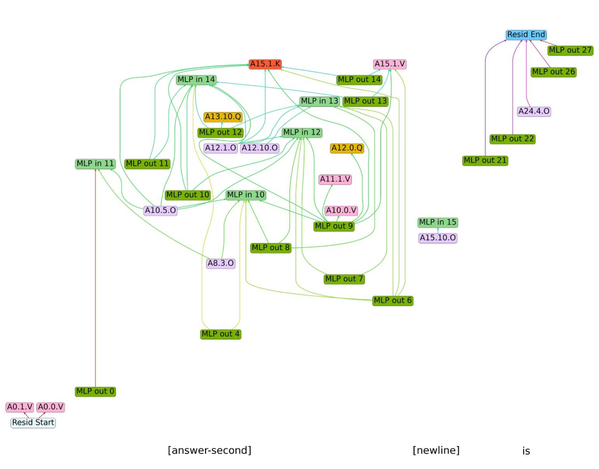
Problem: Anne has 6 apples. She gets 7 more apples. How many apples does she have now? Reasoning: Anne has 6 + 7 = **13** apples. So, she has **17** apples in total. Answer: The above reasoning is → **incorrect**

3) errors in **both** these positions

Problem: Anne has 6 apples. She gets 7 more apples. How many apples does she have now? Reasoning: Anne has 6 + 7 = **17** apples. So, she has **17** apples in total. Answer: The above reasoning is → **incorrect**

We evaluate 4 small Instruction-tuned LMs: **Qwen-2.5-**(Math)**-1.5B-Instruct**, Llama-3.2-3B-Instruct, Phi-3.5-Mini-Instruct

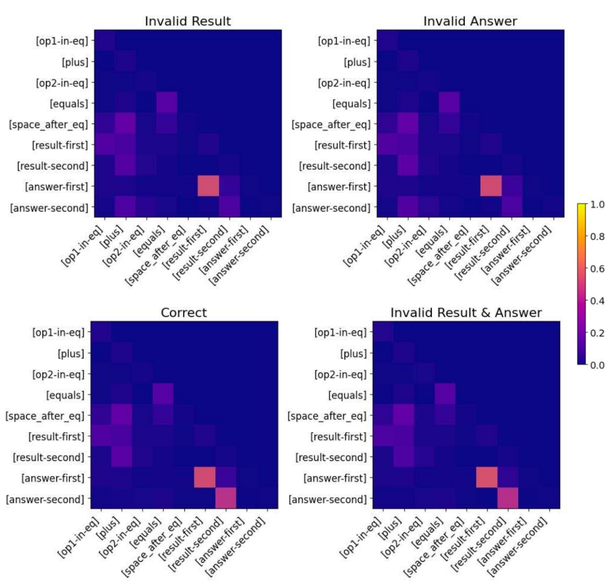
The Error Detection Mechanism



We identified subgraphs, or **circuits**, for detecting errors in the **arithmetic result** and **final answer** positions using **Edge Attribution Patching** (**EAP**) [2]

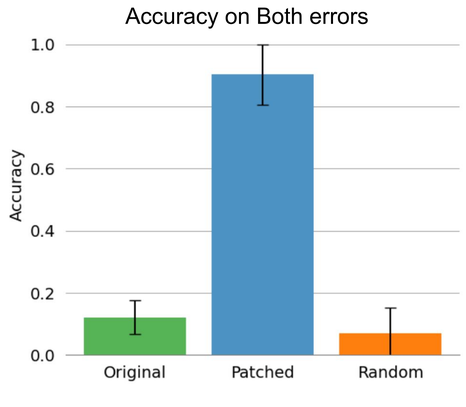
EAP approximates the effect of replacing activations from prompts **without an error** with those from prompts **containing an error** to identify which components are **causally** responsible for predicting that there is an error.

We identified within these circuits a set of **Consistency Heads**—attention heads that assess the **surface-level alignment** of numerical values in the result and final answer. These heads are typically found in the middle layers.



[op1-in-eq]	6
[plus]	+
[op2-in-eq]	7
[equals]	=
[space_after_eq]	

[result-first]	1
[result-second]	3
[answer-first]	1
[answer-second]	7



Consistency Heads play an important **causal role** in error detection, supported by two key results:

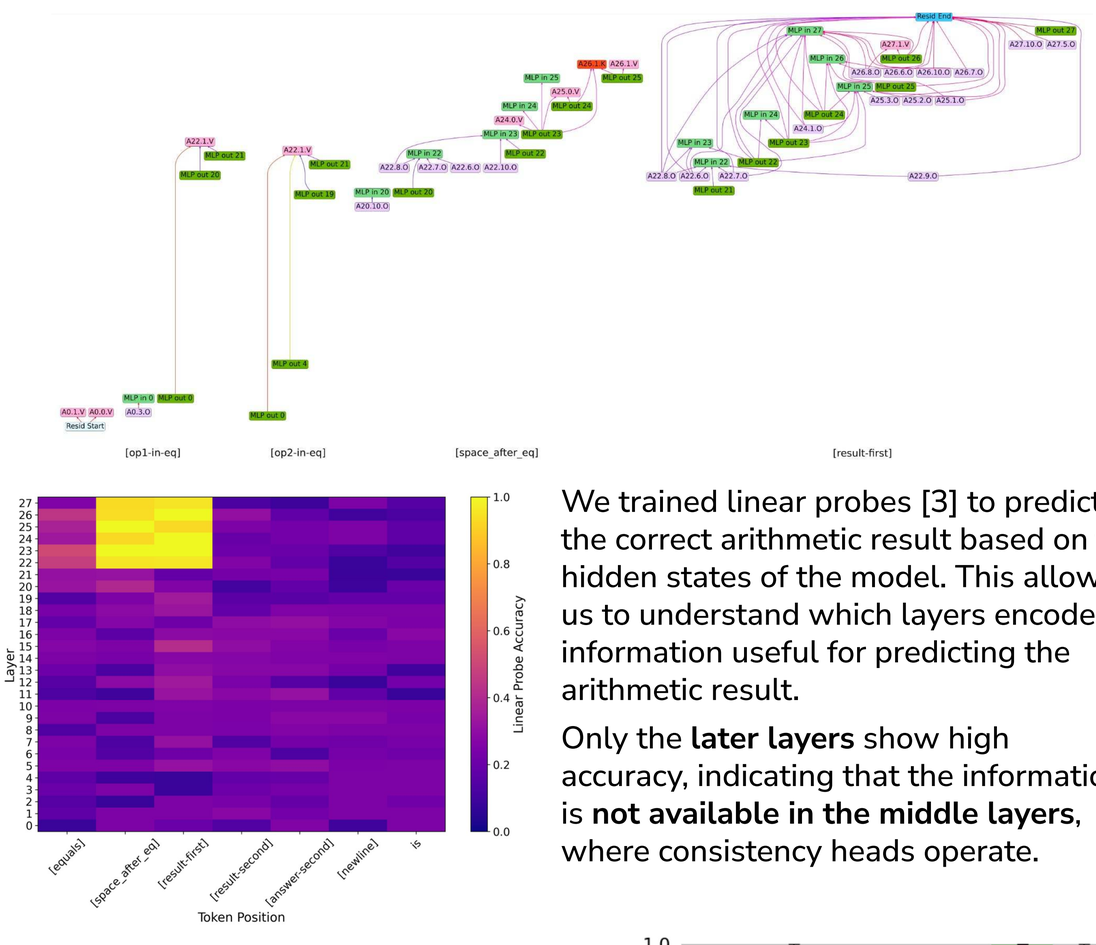
1. We found that models exhibit a **strong bias** toward classifying prompts with errors in both positions as "correct."
2. Causally intervening on these heads **improves** the models' performance on such prompts.

Computation vs. Validation

Our results suggest that models validate prompts by relying on **surface-level consistency checks**.

But what about the arithmetic result? Can models compute it correctly? (100%!)

We used EAP to identify a circuit responsible for **computing the arithmetic result**.

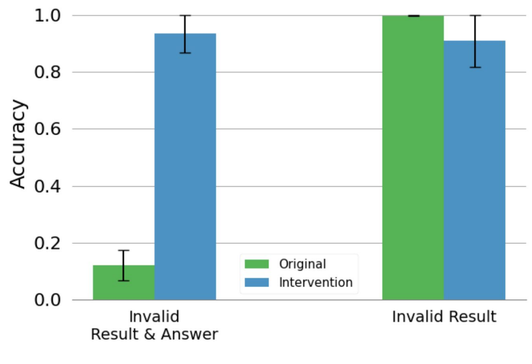


We trained linear probes [3] to predict the correct arithmetic result based on the hidden states of the model. This allowed us to understand which layers encode information useful for predicting the arithmetic result.

Only the **later layers** show high accuracy, indicating that the information is **not available in the middle layers**, where consistency heads operate.

Bridging the gap between **computation** and **validation** by adding representations from higher layers to earlier ones is highly effective.

This approach boosts the accuracy of models on prompts with errors in both positions, without negatively affecting prompts with only one type of error.



Paper **Code**

References

[1] Ryo Kamoi, Yusen Zhang, Nan Zhang, Jiawei Han, and Rui Zhang. 2024. **When can LLMs actually correct their own mistakes? a critical survey of self-correction of LLMs**. *Transactions of the Association for Computational Linguistics*.

[2] Aaquib Syed, Can Rager, and Arthur Conmy. 2024. **Attribution patching outperforms automated circuit discovery**. *Proceedings of the 7th BlackboxNLP Workshop: Analyzing and Interpreting Neural Networks for NLP*.

[3] John Hewitt and Christopher D. Manning. 2019. **A Structural Probe for Finding Syntax in Word Representations**. *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics*.