All for One: LLMs Solve Mental Math at the Last Token With Information Transferred From Other Tokens

Siddarth Mamidanna[1], Daking Rai[2], Ziyu Yao[2], Yilun Zhou[3]

[1]University of California, Santa Cruz, [2]George Mason University, [3]Datadog AI Research

Correspondence to: spmamida@ucsc.edu
https://github.com/siddarth-pm/all-for-one

Abstract

Large language models (LLMs) demonstrate proficiency across numerous computational tasks, yet their inner workings remain unclear. In theory, the combination of causal self-attention and multilayer perceptron layers allows every token to access and compute information based on all preceding tokens. In practice, to what extent are such operations present? In this paper, on mental math tasks (i.e., direct math calculation via next-token prediction without explicit reasoning), we investigate this question in three steps: inhibiting input-specific token computations in the initial layers, restricting the routes of information transfer across token positions in the next few layers, and forcing all computation to happen at the last token in the remaining layers. With two proposed techniques, Context-Aware Mean Ablation (CAMA) and Attention-Based Peeking (ABP), we identify an All-for-One subgraph (AF1) with high accuracy on a wide variety of mental math tasks, where meaningful computation occurs very late (in terms of layer depth) and only at the last token, which receives information of other tokens in few specific middle layers. Experiments on a variety of models and arithmetic expressions show that this subgraph is sufficient and necessary for high model performance, transfers across different models, and works on a variety of input styles. Ablations on different CAMA and ABP alternatives reveal their unique advantages over other methods, which may be of independent interest.

1 Introduction

Large language models (LLMs) perform well on a multitude of computational tasks, and one of the biggest contributing factors is the transformer architecture (Vaswani et al., 2017). Unlike its recurrent neural network (RNN) predecessor, a transformer allows for any token to immediately access all preceding tokens for information transfer via self-attention and enables each token to carry out its independent computation in parallel via multilayer perceptron (MLP). However, it is not clear to what extent these operations are actually happening.

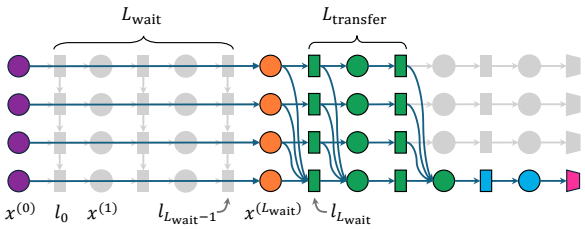


Figure 1: The full AF1 subgraph consists of three stages. First, input-specific computation is suppressed where the input embeddings skip the first L_{wait} layers with context-aware mean ablation (CAMA). Then, the resulting activations $x^{(L_{\text{wait}})}$ pass through L_{transfer} layers of attention-based peeking (ABP) where the only cross-token attentions are those from the last token to preceding ones. Last, for the remaining remaining ABP layers, the last token only attends to itself without any cross-token attention to finish computation, ending in the outputs. In this diagram, $L_{\text{wait}} = 3$ and $L_{\text{transfer}} = 2$.

In this paper, we focus on "mental math" tasks of two and three operands (i.e., arithmetic problems that can be solved with a one token response without explicit chain-of-thought reasoning by the model, such as $42 + 20 - 15$) and investigate the "least amount of computation" that still allows the model to perform well. Specifically, we ask the following questions. First, although each token can access all preceding tokens at every layer, is such access actually executed from the beginning? Second, do all tokens need to perform computation, or is the last-token computation sufficient, given that the next token is predicted from its final residual stream representation? Last, does the last-token computation need access to all other tokens in all layers, or can it function after a (short) period of information transfer?

To answer these questions, we progressively modify the vanilla transformer architecture with two techniques, Context-Aware Mean Ablation

Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, pages 30736–30749
November 4-9, 2025

(CAMA) and Attention-Based Peeking (ABP), and get a surprisingly sparse subgraph that performs well on a wide variety of mental math prompts, with three stages (Fig. 1). First, in the early layers, all tokens wait to access other tokens and instead perform task-general computation (e.g., understanding that the next-token prediction requires three-operand arithmetic) without input-specific information (e.g., the numerical value of the first operand) from other input tokens. Second, in a few middle layers, all tokens transfer their information to the last token. Finally, in the remaining layers, the last token continues the computation to yield the next token prediction. Since input-specific computation is only computed at the last token with information transferred from other tokens, we call this computation subgraph All-for-One (AF1). We use L_{wait} and L_{transfer} to represent the number of layers in the first two stages and study this family of subgraphs extensively. Notably, for Llama-3-8B and Llama-3.1-8B (Grattafiori et al., 2024), and, to a lesser extent, Pythia (Biderman et al., 2023) and GPT-J (Wang and Komatsuzaki, 2021), the first stage can be extended quite long, to nearly half of all layers, while the second stage only needs as few as two layers to recover high performance.

This paper makes three main contributions. First, the fact that AF1 works well on a two-hop arithmetic task (e.g., $A + B + C$) suggests a lack of compositionality at different token positions (e.g., computing $A + B$ in initial layers, storing it in the token B and then adding C in later layers). Second, the short information transfer period may not be unique to arithmetic, possibly implying that the token residual streams spend most time computing rather than communicating. Last, an ablation study on CAMA and ABP shows that different alternative designs fail to uncover this subgraph, making them potentially useful new tools for investigating other LLM behaviors as well.

2 Related Work

Mechanistic Interpretability Mechanistic interpretability (MI) seeks to reverse-engineer the internal mechanisms of LLM behaviors by analyzing their weights and activations (Bereska and Gavves, 2024; Elhage et al., 2021; Rai et al., 2024). Recent MI work has shed light on a range of LLM capabilities, including in-context learning (Elhage et al., 2021; Hendel et al., 2023; Ren et al., 2024), reasoning (Biran et al., 2024; Dutta et al., 2024; Nikankin et al., 2024; Rai and Yao, 2024; Stolfo et al., 2023), and factual recall (Chughtai et al., 2024; Geva et al., 2023; Hernandez et al., 2023; Meng et al., 2022). Building on these advances, we investigate how LLMs perform arithmetic reasoning. In addition, prior MI studies have introduced a range of investigative techniques, including ablation, activation patching, and logit lens (Chan et al., 2022; Goldowsky-Dill et al., 2023; Li and Janson, 2024; Meng et al., 2022; nostalgebraist, 2020; Wang et al., 2022). We employ logit lens in our study and also introduce two new methods: CAMA, an ablation approach inspired by mean ablation, and ABP, a technique for enforcing selective information transfer through attention. Relatedly, Haklay et al. (2025) introduced a position-aware circuit method with similarities to our ABP technique, highlighting the position-dependent nuances that LLMs employ.

Interpreting LLMs in Arithmetic Reasoning Tasks Several prior studies have examined the internal mechanisms of LLMs to understand how they perform arithmetic reasoning (Maltoni and Ferrara, 2024; Nanda et al., 2023; Rai and Yao, 2024; Wu et al., 2023). Zhou et al. (2024) observed that LLM leverages Fourier space features to perform addition. Most relevant to our work, Stolfo et al. (2023) and Zhang et al. (2024) mapped out the general structure of arithmetic circuits in LLMs, detailing how operands and operators are processed, how information is transferred across layers, and how results are ultimately computed. Similarly, Nikankin et al. (2024) identified circuits responsible for arithmetic reasoning, which suggests that LLMs rely on a collection of heuristics, each effective over a limited input distribution, rather than a single monolithic algorithm. However, their analysis is restricted to a two-operand template of `A∘B=`. In contrast, we identify a general-purpose subgraph that faithfully handles both two- and three-operand arithmetic tasks, including both symbolic (e.g., "$3 + 4 + 5$") and verbal (e.g., "The sum of 3 and 4 is") formulations.

Interpreting LLMs in Implicit Reasoning Several prior studies have investigated how LLMs reason implicitly over parametric knowledge (Li et al., 2024; Sakarvadia et al., 2023; Yang et al., 2024). However, they do not track how information flows across different token positions and layers for multi-hop reasoning. More relevant to our study, Biran et al. (2024) and (Wang et al., 2024) studied multi-hop factual recall and showed that LLMs resolve

single-hop answers in earlier layers and propagate the result to final token positions in middle layers, with second-hop factual recall occurring in later layers. We build on this line of work to study the multi-hop reasoning for the three-operand task, but with contradictory findings suggesting a lack of such inter-layer hopping behavior. Additionally, Csordás et al. (2025) observed non-compositional tendancies in models similar to ours, highlighting how LLMs often do not rely on systematic compositional reasoning across novel task formulations.

3 Methods

3.1 LLM Architecture Notation

To standardize presentation, we use the following notations to describe the LLM architecture (see Fig. 1). Let $\mathbf{x} = \{x_1, ..., x_T\}$ be an input sequence of T tokens. Use $x_t^{(0)}$ to denote the original (token and positional) input embedding for token x_t. For L layers, layer $l \in \{0, ..., L-1\}$ takes in $x_t^{(l)}$ and computes $x_t^{(l+1)}$, resulting in the sequence of $x_t^{(1)}, x_t^{(2)}, ..., x_t^{(L)}$ via self-attention and multi-layer perceptron (MLP), which constitute the residual stream.[1] For model m, we write $m(\mathbf{x}, t, l)$ as the function that computes $x_t^{(l)}$ by the first l layers.

3.2 From Transformer to AF1

We develop the AF1 subgraph for Llama-3-8B with respect to the $A + B + C$ task, where A, B, and C are numerical tokens. First, we progressively replaced the first L_{wait} layers of the full transformer with a "waiting" period in which each token can compute independently but not access any other tokens. These computations are task-general, such as understanding numerical tokens and arithmetic structure, rather than input-specific, such as performing the operation in the input. We continue this replacement until performance significantly drops, at which point we conclude that information transfer is necessary. We propose a novel waiting mechanism in Sec. 3.3.

In the second phase, with the waiting period in place, we modify token attentions in all subsequent layers such that the last token x_T can attend to all tokens, while other tokens $x_1, ..., x_{T-1}$ can only attend to themselves.[2] Despite this drastic attention pruning, performance on the task remains high, matching the full-computation case. The attention pruning is detailed in Sec. 3.4.

Finally, with the previous attention pruning already in place, we only allow x_T to attend to (and hence receive information from) all tokens during the first L_{transfer} layers after the waiting, for $L_{\text{transfer}} \in \{0, 1, ..., \leq L - L_{\text{wait}}\}$. In the remaining layers, we force it to only attend to itself,[2] similar to all other tokens in these layers. These L_{transfer} layers are thus sufficient for transferring all input-specific information to x_T, whose final activation $x_T^{(L)}$ is used by later layers to predict the next token.

We present the detailed results of these three steps for Llama-3-8B on the task of $A + B + C$ in Sec. 4.2. Qualitatively, we observe that L_{wait} can be quite long, to almost half of the total number of layers, while L_{transfer} can be as small as 2, suggesting that a very brief burst of information transfer is sufficient for this arithmetic task. More broadly, although self-attention allows for a quadratic number of information transfer routes, the sufficiency of only a linear number of such routes echoes the success of linear attention transformers (Katharopoulos et al., 2020).

3.3 Token Waiting with Context-Aware Mean Ablation (CAMA)

The transformer architecture immediately gathers and distributes information from and to all tokens starting from the first self-attention layer.[3] However, such information fusion may not be present, especially in early layers, which more likely focus on low-level token features (Tenney et al., 2019).

There are many alternative approaches to test for the existence of this waiting period, such as to copy the input embedding directly to the output of the waiting layers, or to allow each token to only attend to itself in these layers. However, as Sec. 4.8 shows, none of them preserve model performance with a long waiting period, most likely because they lead to out-of-distribution representations. Furthermore, these layers may also perform task-general computation, such as processing a number token to "understand" its numerical value.

In this paper, we propose **context-aware mean ablation** (CAMA),[4] a more "in-distribution" ablation approach which is aware of and tailored to the

[1] We use 1-based indexing for token, and 0-based indexing for layer (and attention head), following common practice.

[2] For technical reasons discussed in Sec. 3.4, we also allow an additional attention to the BOS token.

[3] Technically, the causal attention means that a token can only receive information from its preceding tokens but we ignore this detail for ease of presentation.

[4] Fun fact: Cama is a hybrid between a Camel and a Llama.

input context distribution and also allows for such task-general computation. Given the distribution $\mathbb{P}(\mathbf{x})$ over all input sequences, making x_t wait for L_{wait} layers (i.e., layers 0 to $L_{\text{wait}} - 1$) under the CAMA means replacing $x_t^{(L_{\text{wait}})}$ with

$$\tilde{x}_t^{(L_{\text{wait}})} = \mathbb{E}_{\mathbf{x}' \sim \mathbb{P}(\mathbf{x}|x_t)}\big[\, m(\mathbf{x}',\, t,\, L_{\text{wait}})\big]. \quad (1)$$

As CAMA replaces the true $x_t^{(L_{\text{wait}})}$ with the expected representation at the same position over all inputs conditioned on t-th token being x_t, it preserves the general effect of the context on the representation and allows for task-general computation, while erasing any input-specific information carried by the particular sequence $\mathbf{x}$ (other than that of x_t). In other words, given the input distribution, *the CAMA value of a specific token yields no new information about other tokens in the input*.

As we show in Sec. 4.8, the CAMA formulation is crucial to uncover the minimal AF1 circuit proposed in this paper, with alternative designs being infeasible, suggesting broader applicability to other tasks. In addition, it may be generalizable to ablating out information fusion that does not start in the first layer, which we leave to future work.

3.4 Selective Information Transfer with Attention-Based Peeking (ABP)

CAMA enforces *waiting* of a token by using context-aware ablation to block a token from accessing input-specific information from other tokens, but it must be applied from the first layer and runs continuously. Here, we introduce an alternative mechanism to control information access at arbitrary layer(s), called **attention-based peeking** (ABP). It is implemented by modifying the attention mask so that, in the target layer, each query position is allowed to attend only to (or "peek at") a subset of previous key positions.

For a (query) token x_t, we use $K_t \subseteq \{1, ..., t\}$ to denote the index set of (key) tokens whose information we want to transfer to x_t. Let $M \in \mathbb{R}^{T \times T}$ be the (pre-softmax) attention matrix for the entire sequence $\mathbf{x}$ and $K_1, ..., K_T$ be the peeking index sets. We replace each $M_{q,k}$ with $-\infty$ if $k \notin K_q$, so the softmax zeroes out the attention to any keys not in the peek set.

While ABP can be implemented to attend to any subset of (preceding) tokens, in this paper, we consider two specific cases, **full-peeking**, $K_t = \{1, ..., t\}$ where x_t attends to all tokens (and recovers the standard causal attention) and **self-peeking**, $K_t = \{t\}$ where x_t attends only to itself.

Input template	Tokenization	Operator $\circ$
"<BOS>A∘B="	<BOS> A ∘ B =	$+, -, *, /$
"<BOS>A ∘ B ∘ C = "	<BOS> A ␣∘ ␣ B ␣∘ ␣ C ␣= ␣	$+, -$

Table 1: The input templates along with their tokenization (space represented by ␣). A, B, and C are randomly selected from $\{0, 1, ..., 100\}$ with an additional constraint that the answer from selected input is an integer in the range $\{0, 1, ..., 999\}$. The first template includes spaces around the two operators (which can be different) and the second template does not contain any spaces.

First token attention. A common quirk of attention patterns is the attention sink phenomenon (Cancedda, 2024; Xiao et al., 2023), where tokens strongly attend to the first one, which is often the special <BOS> token. In our experiments, we find that removing the attention to <BOS> is indeed devastating to model performance. Thus, we always keep it with $K_t \leftarrow K_t \cup \{1\}$.

4 Experiments

After introducing the experiment setup in Sec. 4.1, we present the results of our AF1 subgraph discovery process in Sec. 4.2 for Llama-3-8B on the $A + B + C$ task. Then in subsequent sections, we study various properties of this subgraph to demonstrate its generality.

4.1 Experiment Setup

We consider both two- and three-operand arithmetic, with task templates and their tokenization summarized in Tab. 1. Each operand is a randomly sampled integer from 0 to 100, subject to the additional constraint that the final answer is an integer from 0 to 999. The CAMA implementation for these tasks can be found in App. A.1.

We mainly study Llama-3-8B and Llama-3.1-8B, but, to study cross-model generalization, also consider two earlier models, Pythia-6.9B and GPT-J-6B. Since the latter two have very poor three-operand performance, we only study two-operand tasks for them. Tab. 2 presents the raw model accuracy on randomly sampled legal inputs for all tasks and models.

As model-task accuracy can vary significantly, we use the **faithfulness** of a subgraph s to measure its performance, defined as its accuracy on prompts (x, y) for which the full model m is correct:

$$\text{faith.}(s) = \mathbb{E}_{x,y}[s(x) = y | m(x) = y]. \quad (2)$$

Operation	Model	Raw Acc.	Operation	Model	Raw Acc.
$A+B+C$	Llama-3-8B	0.994	$A+B$	Llama-3-8B	0.962
$A+B-C$	Llama-3-8B	0.932	$A-B$	Llama-3-8B	0.899
$A-B+C$	Llama-3-8B	0.296	$A\times B$	Llama-3-8B	0.937
$A-B-C$	Llama-3-8B	0.748	$A\div B$	Llama-3-8B	0.966
$A+B+C$	Llama-3.1-8B	0.998	$A+B$	Llama-3.1-8B	0.953
$A+B-C$	Llama-3.1-8B	0.956	$A-B$	Llama-3.1-8B	0.518
$A-B-C$	Llama-3.1-8B	0.956	$A\div B$	Llama-3.1-8B	0.814
$A-B+C$	Llama-3.1-8B	0.977	$A\times B$	Llama-3.1-8B	0.737
$A+B$	Pythia	0.584	$A+B$	GPT-J	0.559
$A-B$	Pythia	0.226	$A-B$	GPT-J	0.357
$A\times B$	Pythia	0.131	$A\times B$	GPT-J	0.316
$A\div B$	Pythia	0.220	$A\div B$	GPT-J	0.313

Table 2: Raw accuracy of various models for different math operations.

To calculate CAMA values for each token, we draw input $\mathbf{x}'$ with different operand values but of the same task. In other words, we randomize all operands (other than the token itself, if applicable) while fixing the operator(s), along with the equality sign and any space tokens, when present.

4.2 AF1 Subgraph Discovery Result

Sec. 3.2 details the discovery process of the "All-for-One" (AF1) subgraph for the three-operand task $A+B+C$ via a three-phase ablation and peeking procedure. Here, we present the results for Llama-3-8B.

Fig. 2 (top) tracks the faithfulness when the first L_{wait} layers of the full transformer are replaced with Context-Aware Mean Ablation (CAMA). Faithfulness remains high for $L_{\text{wait}} \leq 15$ and then collapses sharply at $L_{\text{wait}} = 16$, indicating that no cross-token interaction before layer index 14 is necessary, but information transfer at layer index 15 is critical (recall that layer index is 0-based).

We then investigate the necessity of information transfer at layer 15 and onward, replacing the attention components with full-peeking using ABP. In

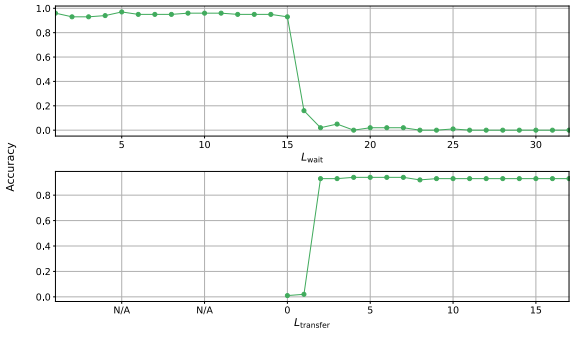


Figure 2: Top: Llama-3-8B performance after making tokens to wait for the first L_{wait} layers with CAMA, as a function of L_{wait}. Bottom: for $L_{\text{wait}} = 15$ and self-peeking on all non-last tokens, performance of model with last token full-peeking in the next L_{transfer} layers and self-peeking afterward, as a function of L_{transfer}.

Operation	Model	Faithfulness	Operation	Model	Faithfulness
$A+B+C$	Llama-3-8B	0.995	$A+B$	Llama-3-8B	0.854
$A+B-C$	Llama-3-8B	0.944	$A-B$	Llama-3-8B	0.987
$A-B+C$	Llama-3-8B	0.312	$A\times B$	Llama-3-8B	0.710
$A-B-C$	Llama-3-8B	0.995	$A\div B$	Llama-3-8B	0.887
$A+B+C$	Llama-3.1-8B	0.995	$A+B$	Llama-3.1-8B	0.771
$A+B-C$	Llama-3.1-8B	0.974	$A-B$	Llama-3.1-8B	0.889
$A-B+C$	Llama-3.1-8B	0.967	$A\times B$	Llama-3.1-8B	0.503
$A-B-C$	Llama-3.1-8B	0.983	$A\div B$	Llama-3.1-8B	0.779

Table 3: $\text{AF1}_{\text{llama}}$ circuit faithfulness for different math operations, on both Llama-3-8B and Llama-3.1-8B.

particular, we prohibit all non-last tokens to attend to any other token except for itself and the BOS token, while allowing the last token to still attend to every previous token. This operation drastically changes the amount of information flow across tokens. However, we observe no performance degradation, suggesting that non-last tokens do not need to receive information from elsewhere and only need to send information to the last one.

Finally, we incrementally remove the last token full-peeking and restrict it to self-peeking (where the last token attends only to itself and the BOS token) from the back. Fig. 2 (bottom) tracks the faithfulness when we modify layer L_{transfer} to from 0 to 17 (both inclusive) with the layers $L \geq L_{\text{transfer}} + L_{\text{wait}}$ being self-peeking. We find that performance stays high as long as $L_{\text{transfer}} \geq 2$, or in other words, the last token can access other tokens at least layer 15 and 16.

Thus, we have identified a subgraph for Llama-3-8B on the task of $A+B+C$ that retains almost full performance, with 14 CAMA layers and 2 information transfer layers followed by last token self-computation. We call this $\text{AF1}_{\text{llama}}$. In subsequent sections, we study its performance on other tasks for both Llama-3-8B and Llama-3.1-8B, as well as analogous subgraphs in Pythia and GPT-J.

4.3 $\text{AF1}_{\text{llama}}$ Subgraph Performance

Tab. 3 presents faithfulness of the same $\text{AF1}_{\text{llama}}$ subgraph on eight tasks across both Llama models. Despite the sparsity of connections, this subgraph demonstrates high performance in many tasks. The most notable exception is on $A-B+C$ for Llama-3-8B, with faithfulness just over 0.3. Considering that the original model accuracy is below 0.3 (Tab. 2), the low subgraph faithfulness and low model accuracy may be both caused by inconsistent problem-solving logic used by the model on these inputs. By contrast, when Llama-3.1-8B achieves a high raw accuracy on this task, $\text{AF1}_{\text{llama}}$ attains similarly high subgraph faithfulness.

Fig. 3 plots the faithfulness of the subgraph with

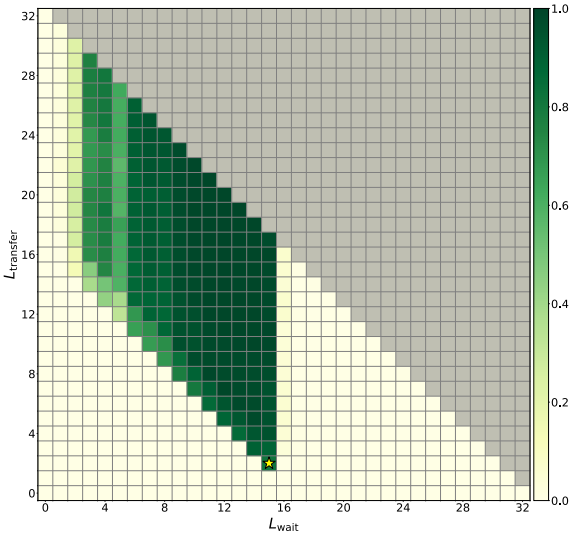


Figure 3: Faithfulness of different AF1 configurations over $L_{\text{transfer}} \in [0, 32]$, $L_{\text{wait}} \in [0, 32]$ for $A + B + C$ task on Llama-3-8B model. The minimal subgraph $\text{AF1}_{\text{llama}}$ is marked with a yellow star. Two conditions are necessary for preserving model accuracy: (1) Waiting can never occur past layer index 14 ($L_{\text{wait}} \leq 15$); (2) Information transfer must cover layer 16 (i.e., $L_{\text{wait}} + L_{\text{transfer}} \geq 17$).

various L_{wait} and L_{transfer} values (with remaining layers implementing last-token self-peeking), for Llama-3-8B on the $A + B + C$ task. We observe the following necessary conditions for high performance: (1) waiting must stop at layer 14 *at the latest* (i.e., $L_{\text{wait}} \leq 15$), and (2) the subsequent information transfer must be at least two layers long (i.e., $L_{\text{wait}} + L_{\text{transfer}} \geq 17$).

Additionally, replacing CAMA layers with information transfer layers too early (i.e., at or before layer 6) actually hurt the performance. Since in an information transfer layer, each non-last token can only attend to itself, without access to task-general context as in CAMA, its ability to perform task-general computation is limited, which proves to be important especially for early layers. A full grid of L_{wait}, L_{transfer} for all other tasks for both Llama-3-8B and Llama-3.1-8B can be found in App. A.2.

4.4 Necessity of Information Transfer Layers

To test the necessity of the information transfer layers identified (layers 15 *and* 16) in Llama-3-8B, we run the full model computation but, for each *single* layer, remove the attentions from the last token to every other non-BOS tokens, while keeping all other attention connections. The resulting model

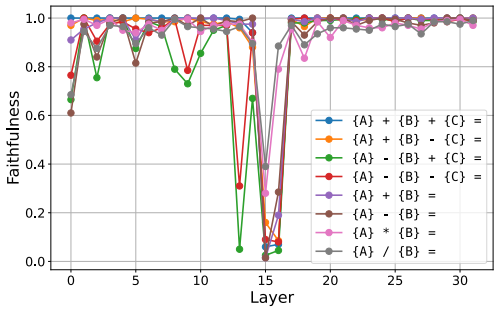


Figure 4: Faithfulness of the full Llama-3-8B but with the attention from the last token to every other non-BOS token removed individually in each layer.

performances are shown in Fig. 4.

Despite removing only $T-2$ out of $T \cdot L \cdot N(N-1)/2$ connections, the effect can be drastic. Without an exception, removing layer 15 attentions to a large performance drop on all tasks, while removing layer 16 attentions affects all but two tasks. This provides further evidence that these two specific attention layers are indeed critical for information transfer. This information, combined with the model's inability to wait past layer 14, suggests special importance of information transfer in layer 15 and 16, agreeing with prior work (Nikankin et al., 2024) that identifies their attention heads being responsible for two-operand arithmetic. As a side note, layer 13 attention is apparently also important for $A - B \pm C$ tasks, although $\text{AF1}_{\text{llama}}$ (with layer 13 being in the CAMA waiting stage) performs extremely well on $A - B - C$. We leave this investigation to future work.

In $\text{AF1}_{\text{llama}}$, since only layer 15 and 16 participate in information transfer, we further investigate the importance of each attention head. To this end, we identify the minimal set of heads that preserves the model's performance, through an iterative process as follows (in Llama-3-8B). In Llama, each layer has 32 heads, leading to 64 heads in total for layers 15 and 16. We start with the full $\text{AF1}_{\text{llama}}$ subgraph and remove attention heads iteratively. At every iteration, for each attention head remaining, we compute the model accuracy without this head, and remove the one with the lowest impact. We repeat this procedure until no head remains.

Tab. 4 shows the accuracy following head removal. We see that very few heads are actually important for arithmetic computation, with 95% accuracy preserved after removing nearly 60 heads. Furthermore, we identified some common heads, marked with asterisks.

$A+B+C$		$A+B-C$	
Heads Removed	Accuracy	Heads Removed	Accuracy
59 Least Important	95.5%	56 Least Important	95.0%
L15H31	93.0%	L16H20	93.5%
L16H1 *	64.5%	L15H6	91.5%
L15H13 *	8.5%	L15H3 *	83.5%
L15H3 *	1.5%	L16H1 *	46.0%
L16H21 *	0.5%	L16H2	30.5%
-	-	L16H3	3.5%
-	-	L16H21 *	1.5%
-	-	L15H13 *	0.5%

Table 4: Preserved accuracy across *cumulative* head removals in layers 15 and 16 on $\text{AF1}_{\text{llama}}$ for $A+B+C$ and $A+B-C$. Asterisks denote heads shared across both tasks.

Further attention analysis on these individual heads reveals that some mostly attend to the BOS token, and the remaining heads attend to one of the numerical operands. For example, we show in Fig. 5 that heads , L15H13 and L15H3 attend to the first, second, and third operand respectively, for the $A+B+C$ task. This lines up quite neatly with our information transfer hypothesis, with the focused, operand-heavy attention patterns transferring the operand information to the last token. While this analysis is performed on the $\text{AF1}_{\text{llama}}$ subgraph, we observe highly similar results for full Llama-3-8B model as well (App. A.3), suggesting that $\text{AF1}_{\text{llama}}$ captures the "essence" of computation.

4.5 Internal Representation Analysis

We probe the final token's residual stream at each layer using **logit lens analysis**, which projects the residual stream values after each MLP layer to the vocabulary space with the unembedding matrix (nostalgebraist, 2020). Additionally, we also apply this technique to each attention head. To observe the effect of the $\text{AF1}_{\text{llama}}$ modification, we compare the results on both the full Llama-3-8B

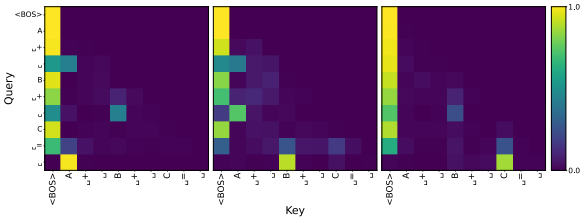


Figure 5: Attention patterns for the three key operand heads in the $A+B+C$ task (Llama-3-8B): L16H21 (left), L15H13 (middle), L15H3 (right), attending from the last token to the first, second and third operands. Activation values are averaged across 100 prompts.

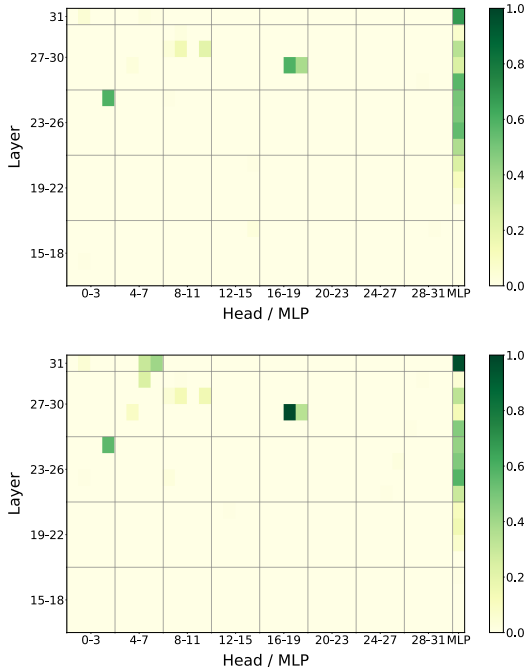


Figure 6: Logit lens top-3 accuracy of for each attention head and the MLP layer on the full Llama-3-8B model (top) vs. its AF1 subgraph (bottom) on Llama-3-8B.

model and its $\text{AF1}_{\text{llama}}$ subgraph.

Fig. 6 depicts the logit lens top-3 accuracy for the full Llama-8-8B (top) and $\text{AF1}_{\text{llama}}$, defined as the fraction of inputs for which the correct answer appears among the top-3 vocab logits, at each attention head as well as the MLP layer. We see high accuracy emerging clearly around layer 24 from the logit lens analysis in Fig. 6. The similarity of the two plots suggest AF1's captures almost the full prediction power and underlying mechanisms as the full model, instead of finding an alternative computational pathway or a "hacky shortcut" to solve the arithmetic.

L28H18 and L26H3 are the only attention heads that consistently predict the correct answer, even with the answer appearing in the MLP components at earlier layers. However, attention visualization on these heads revealed no more insights, as the pattern shows a common attention sink pattern on the BOS token (App. A.4). The logit lens analysis for Llama-3.1-8B yields similar results, as shown in App. A.5.

4.6 Generalized Arithmetic Inputs

We investigate whether $\text{AF1}_{\text{llama}}$ can generalize to other arithmetic forms representing the operations $A+B$ and $A-B$ on Llama-3-8B. We consider several input templates such as describing the op-

Style	Task	Model Acc.	$AF1_{llama}$ Faith.	Template and Tokenization
Original	$A+B$	0.962	0.854	<BOS> A + B =
	$A-B$	0.899	0.987	<BOS> A - B =
Verbal Math	$A+B$	1.000	1.000	<BOS> The ␣sum ␣of ␣ A ␣and ␣ B ␣is ␣
	$A-B$	1.000	1.000	<BOS> The ␣difference ␣of ␣ A ␣and ␣ B ␣is ␣
Question Answering	$A+B$	0.999	1.000	<BOS> What ␣is ␣the ␣sum ␣of ␣ A ␣and ␣ B ? ␣Answer : ␣
	$A-B$	0.991	1.000	<BOS> What ␣is ␣the ␣difference ␣of ␣ A ␣and ␣ B ? ␣Answer : ␣
Instruction	$A+B$	0.999	0.995	<BOS> If ␣you ␣add ␣ A ␣to ␣ B , ␣you ␣will ␣get ␣
	$A-B$	1.000	0.905	<BOS> If ␣you ␣subtract ␣ A ␣to ␣ B , ␣you ␣will ␣get ␣
Math Word Problem	$A+B$	0.998	0.005	<BOS> John ␣has ␣ A ␣cookies . ␣Jane ␣has ␣ B ␣cookies . ␣Together ␣they ␣have ␣
	$A-B$	0.994	0.020	<BOS> John ␣has ␣ A ␣cookies . ␣He ␣gave ␣Jane ␣ B ␣cookies . ␣John ␣now ␣has ␣
Python Program	$A+B$	0.999	0.001	<BOS> a ␣= ␣ A ; ␣b ␣= ␣ B ; ␣print (a ␣+ ␣b) ␣# ␣should ␣print ␣
	$A-B$	1.000	0.020	<BOS> a ␣= ␣ A ; ␣b ␣= ␣ B ; ␣print (a ␣- ␣b) ␣# ␣should ␣print ␣

Table 5: Prompt templates and results for alternative representations of $A+B$ and $A-B$ tasks on the full Llama-3-8B model and corresponding $AF1_{llama}$ subgraph. A and B are replaced with actual numerical values, and everything else is rendered verbatim.

eration verbally and embedding the operation in word problems or Python code. The concrete templates and resulting model performance are shown in Tab. 5. Even for text-based arithmetic prompts, $AF1_{llama}$ retains considerable accuracy for direct arithmetic tasks without additional semantic contexts. However, it completely fails on tasks requiring semantic understanding such as word problem and Python, suggesting that additional components are needed for other capabilities, like understanding natural language or Python program inputs.

4.7 Pythia and GPT-J Models

In addition to extensively exploring the $AF1_{llama}$ subgraph, we also applied our experimental procedure using CAMA and ABP to investigate if similar AF1 subgraphs exist in other models, namely Pythia and GPT-J. Due to their poor performance on three-operand tasks, we only study two-operand tasks, as done by Nikankin et al. (2024).

Fig. 7 plots the faithfulness for different AF1 configurations on $A+B$ for Pythia (top) and GPT-J (bottom). There are several notable differences from that for Llama in Fig. 3. First, the Pythia and GPT-J grids don't have as definitive of a boundary where performance is nearly completely retained vs. destroyed. Second, the waiting layer must end earlier, with maximum L_{wait} being around 11 to keep decent performance for both models, suggesting that critical information transfer also happens earlier. Finally, these models require a larger minimum L_{transfer}, suggesting that the efficiency of information transfer is lower in them than in Llama. Based on these factors, we choose $(L_{\text{wait}}, L_{\text{transfer}})$

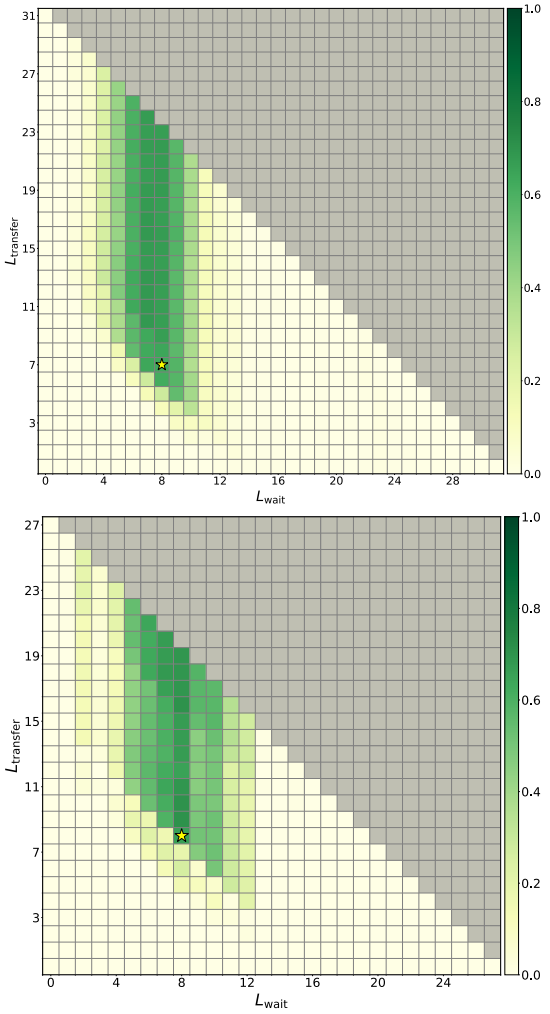


Figure 7: Faithfulness on $A+B$ for different AF1 configurations for Pythia (top) and GPT-J (bottom) explored across all possible values of L_{wait} and L_{transfer}. Tab. 6 reports the best faithfulness score of the AF1 subgraphs, marked with a yellow star in the figure.

Model	$A+B$	$A-B$	$A\times B$	$A\div B$
$\text{AF1}_{\text{Pythia}}$	0.620	0.551	0.780	0.490
$\text{AF1}_{\text{GPT-J}}$	0.647	0.506	0.794	0.440

Table 6: Faithfulness of $\text{AF1}_{\text{Pythia}}$ and $\text{AF1}_{\text{GPT-J}}$ for two-operand arithmetic operations.

to be $(9, 7)$ for $\text{AF1}_{\text{Pythia}}$ and $(9, 8)$ for $\text{AF1}_{\text{GPT-J}}$, marked by stars in Fig. 7.

The faithfulness of these two AF1 subgraphs for all two-operand tasks are reported in Tab. 6. Even though these numbers are usually significantly lower than the Llama counterpart in Tab. 3, we see that they still often recover more than half of the original model's accuracy.

4.8 Alternative CAMA Designs

In addition to CAMA, we evaluated several alternative waiting mechanisms in $\text{AF1}_{\text{llama}}$. **Direct embedding copy (DEC)** simply uses the original embedding $x_t^{(0)}$ for $\tilde{x}_t^{(L_{\text{wait}})}$ in Eq. 1. **Random token mean ablation (RTMA)** uses completely random tokens in Eq. 1 rather than those drawn from the (context-aware) conditional distribution. **Self-peek as waiting (SPAW)** uses ABP in every waiting layer with $K_t = \{1, t\}$ to make each token $\{x_t\}_{t=1}^{T}$ attend to only itself and the BOS token. **Isolated forward pass (IFP)** runs each token as a standalone two-token prompt (with BOS prepended) for L_{wait} layers and takes the final representation as $\tilde{x}_t^{(L_{\text{wait}})}$ in Eq. 1. App. A.6 contains more detailed explanations of each method.

None of the others achieve any non-zero faithfulness. We attribute these failures to two main shortcomings: DEC and RTMA breaks the model's in-distribution assumptions by directly using input embedding values or averaging over random tokens; SPAW and IFP omit the background computation needed to encode operand structure, providing only minimal structural cues without actual value information. By contrast, CAMA both maintains in-distribution representations via marginalization and captures the model's evolving background computation through conditional expectation.

5 Discussion and Conclusion

In this paper, we explored uncovering the least amount of information transfer and computation that support mental math tasks, including those requiring compositional reasoning. Using two new techniques Context-Aware Mean Ablation (CAMA) and Attention-Based Peeking (ABP), we uncovered a highly sparse, three-stage AF1 subgraph which allows computation to only happen in the last token's residual stream with information transferred from other tokens in few layers. This subgraph demonstrates high performance on a wide variety of arithmetic tasks across model.

The discovery and identification of the AF1 subgraph provides significant insight into the computational flow of LLMs, particularly highlighting how minimal information transfer periods and targeted, precise computation suffices for high accuracy in arithmetic tasks. Our experiments reveal that although theoretically, tokens can independently process and transfer information between each other from early layers, in practice, meaningful cross-token computation can be (and is) significantly deferred. This underscores the importance of separating task-general computation (such as token recognition and numerical/structural encoding) from input-specific computation (such as carrying out arithmetic operations).

A key feature for AF1 is that all input-specific computation is carried out in the last token position, despite every token possessing the ability to implement its own computation. We hypothesize that the root cause lies in the training paradigm. Regardless of pre-training, supervised finetuning or preference alignment, the model receives token-level signal – predicting the next token based on the context. This dense signal could make the model fully focused on predicting the next token all the time and thus not able to allocate additional bandwidth for compositional reasoning. To remedy this, custom training signals may be explored that places heavier weight on rewarding more "important" tokens, which may potentially leading to the emergence of intermediate token computations.

Overall, this work contributes to the mechanistic understanding of arithmetic reasoning in LLMs and cross-token computation. In addition, it provides methodological innovations with CAMA and ABP that can serve broader applications beyond arithmetic tasks as well.

Limitations

The biggest limitation of this analysis is its dependency on a "cooperative" tokenizer that allocates a dedicated single token to represent each number, a common practice for recent studies on LLM arithmetic (e.g., Nikankin et al., 2024). While the

tokenizers for Llama, Pythia and GPT-J have this property, notably exceptions include Qwen (Yang et al., 2025) and Gemma (Team et al., 2024), which split numbers into individual digit tokens. Studying these models requires extensions of CAMA and ABP to handle multi-token numbers, which we leave to future work.

Additionally, the task scope is limited primarily to arithmetic operations with clearly defined computational boundaries, which can potentially limit generalization to more complex reasoning and more semantically challenging tasks. As demonstrated in Sec. 4.6, AF1 notably fails on tasks requiring deeper semantic understanding or context interpretation. Thus, exploring the additional components to AF1 that endow models with such capabilities could be useful.

Acknowledgments

DR and ZY are funded by the National Science Foundation (Award Number 2311468/2423813).

References

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

Stella Biderman, Hailey Schoelkopf, Quentin Gregory Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, USVSN Sai Prashanth, Edward Raff, and 1 others. 2023. Pythia: A suite for analyzing large language models across training and scaling. In *International Conference on Machine Learning*, pages 2397–2430. PMLR.

Eden Biran, Daniela Gottesman, Sohee Yang, Mor Geva, and Amir Globerson. 2024. Hopping too late: Exploring the limitations of large language models on multi-hop queries. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 14113–14130.

Nicola Cancedda. 2024. Spectral filters, dark signals, and attention sinks. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 4792–4808.

Lawrence Chan, Adrià Garriga-Alonso, Nicholas Goldwosky-Dill, Ryan Greenblatt, Jenny Nitishinskaya, Ansh Radhakrishnan, Buck Shlegeris, and Nate Thomas. 2022. Causal scrubbing, a method for rigorously testing interpretability hypotheses. *AI Alignment Forum*. https://www.alignmentforum.org/posts/JvZhhzycHu2Yd57RN/causal-scrubbing-a-method-for-rigorously-testing.

Bilal Chughtai, Alan Cooney, and Neel Nanda. 2024. Summing up the facts: Additive mechanisms behind factual recall in llms. *arXiv preprint arXiv:2402.07321*.

Róbert Csordás, Christopher D. Manning, and Christopher Potts. 2025. Do language models use their depth efficiently? *Preprint*, arXiv:2505.13898v2.

Subhabrata Dutta, Joykirat Singh, Soumen Chakrabarti, and Tanmoy Chakraborty. 2024. How to think step-by-step: A mechanistic understanding of chain-of-thought reasoning. *Trans. Mach. Learn. Res.*

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.

Mor Geva, Jasmijn Bastings, Katja Filippova, and Amir Globerson. 2023. Dissecting recall of factual associations in auto-regressive language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 12216–12235.

Nicholas Goldowsky-Dill, Chris MacLeod, Lucas Sato, and Aryaman Arora. 2023. Localizing model behavior with path patching. *arXiv preprint arXiv:2304.05969*.

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

Tal Haklay, Hadas Orgad, David Bau, Aaron Mueller, and Yonatan Belinkov. 2025. Position-aware automatic circuit discovery. *Preprint*, arXiv:2502.04577.

Roee Hendel, Mor Geva, and Amir Globerson. 2023. In-context learning creates task vectors. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 9318–9333.

Evan Hernandez, Arnab Sen Sharma, Tal Haklay, Kevin Meng, Martin Wattenberg, Jacob Andreas, Yonatan Belinkov, and David Bau. 2023. Linearity of relation decoding in transformer language models. *CoRR*.

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. 2020. Transformers are rnns: Fast autoregressive transformers with linear attention. In *International conference on machine learning*, pages 5156–5165. PMLR.

Maximilian Li and Lucas Janson. 2024. Optimal ablation for interpretability. *Advances in Neural Information Processing Systems*, 37:109233–109282.

Zhaoyi Li, Gangwei Jiang, Hong Xie, Linqi Song, Defu Lian, and Ying Wei. 2024. Understanding and patching compositional reasoning in LLMs. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 9668–9688, Bangkok, Thailand. Association for Computational Linguistics.

Davide Maltoni and Matteo Ferrara. 2024. Arithmetic with language models: From memorization to computation. *Neural Networks*, 179:106550.

Kevin Meng, David Bau, Alex Andonian, and Yonatan Belinkov. 2022. Locating and editing factual associations in gpt. *Advances in neural information processing systems*, 35:17359–17372.

Neel Nanda, Lawrence Chan, Tom Lieberum, Jess Smith, and Jacob Steinhardt. 2023. Progress measures for grokking via mechanistic interpretability. *arXiv preprint arXiv:2301.05217*.

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

nostalgebraist. 2020. Interpreting gpt: the logit lens. *AI Alignment Forum*. https://www.lesswrong.com/posts/AcKRB8wDpdaN6v6ru/interpreting-gpt-the-logit-lens.

Daking Rai and Ziyu Yao. 2024. An investigation of neuron activation as a unified lens to explain chain-of-thought eliciting arithmetic reasoning of llms. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7174–7193.

Daking Rai, Yilun Zhou, Shi Feng, Abulhair Saparov, and Ziyu Yao. 2024. A practical review of mechanistic interpretability for transformer-based language models. *arXiv preprint arXiv:2407.02646*.

Jie Ren, Qipeng Guo, Hang Yan, Dongrui Liu, Quanshi Zhang, Xipeng Qiu, and Dahua Lin. 2024. Identifying semantic induction heads to understand in-context learning. In *Findings of the Association for Computational Linguistics ACL 2024*, pages 6916–6932.

Mansi Sakarvadia, Aswathy Ajith, Arham Khan, Daniel Grzenda, Nathaniel Hudson, André Bauer, Kyle Chard, and Ian Foster. 2023. Memory injections: Correcting multi-hop reasoning failures during inference in transformer-based language models. In *Proceedings of the 6th BlackboxNLP Workshop: Analyzing and Interpreting Neural Networks for NLP*, pages 342–356, Singapore. Association for Computational Linguistics.

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.

Gemma Team, Thomas Mesnard, Cassidy Hardin, Robert Dadashi, Surya Bhupatiraju, Shreya Pathak, Laurent Sifre, Morgane Rivière, Mihir Sanjay Kale, Juliette Love, and 1 others. 2024. Gemma: Open models based on gemini research and technology. *arXiv preprint arXiv:2403.08295*.

Ian Tenney, Dipanjan Das, and Ellie Pavlick. 2019. Bert rediscovers the classical nlp pipeline. *arXiv preprint arXiv:1905.05950*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Ben Wang and Aran Komatsuzaki. 2021. GPT-J-6B: A 6 Billion Parameter Autoregressive Language Model. https://github.com/kingoflolz/mesh-transformer-jax.

Boshi Wang, Xiang Yue, Yu Su, and Huan Sun. 2024. Grokked transformers are implicit reasoners: A mechanistic journey to the edge of generalization. In *ICML 2024 Workshop on Mechanistic Interpretability*.

Kevin Wang, Alexandre Variengien, Arthur Conmy, Buck Shlegeris, and Jacob Steinhardt. 2022. Interpretability in the wild: a circuit for indirect object identification in gpt-2 small. *arXiv preprint arXiv:2211.00593*.

Zhengxuan Wu, Atticus Geiger, Thomas Icard, Christopher Potts, and Noah Goodman. 2023. Interpretability at scale: Identifying causal mechanisms in alpaca. *Advances in neural information processing systems*, 36:78205–78226.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. 2023. Efficient streaming language models with attention sinks. *arXiv preprint arXiv:2309.17453*.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, and 1 others. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

Sohee Yang, Elena Gribovskaya, Nora Kassner, Mor Geva, and Sebastian Riedel. 2024. Do large language models latently perform multi-hop reasoning? In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 10210–10229, Bangkok, Thailand. Association for Computational Linguistics.

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 *International Conference on Machine Learning*, pages 59932–59950. PMLR.

Tianyi Zhou, Deqing Fu, Vatsal Sharan, and Robin Jia. 2024. Pre-trained large language models use fourier features to compute addition. *arXiv preprint arXiv:2406.03445*.

A Appendix

A.1 Implementation Tricks for CAMA

Recall that CAMA replaces the true representation $x_t^{(L_{\text{wait}})}$ by the conditional expectation

$$\tilde{x}_t^{(L_{\text{wait}})} = \mathbb{E}_{\mathbf{x}' \sim \mathbb{P}(\mathbf{x}|x_t)}\big[m(\mathbf{x}', t, L_{\text{wait}})\big], \quad (3)$$

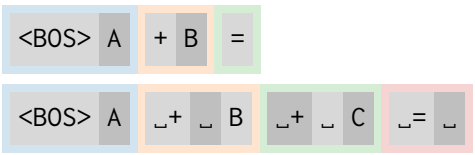
averaging over *in-distribution* prompts that fix the token at position t and vary the rest according to the task distribution. This preserves task-general computation while removing input-specific cross-token information.

Because the self-attention is causal, $x_t^{(L_{\text{wait}})}$ only depends on tokens at positions $\leq t$. Hence, when estimating $\tilde{x}_t^{(L_{\text{wait}})}$, it suffices to marginalize over *only prefixes* $x_{1:(t-1)}$ and not consider the value (or even presence) of $x_{(t+1):T}$. Therefore, when computing the CAMA value for x_t, we can safely consider only the first t tokens, sample $x_{1:t-1}$ from $\mathbb{P}(x_{1:t-1}|x_t)$, and then append the target token value for x_t to the prefix.

Furthermore, there are cases where given a specific value of x_t, the probability of the previous token x_{t-1} collapses to a single value, especially if the task distribution is quite restrictive. For example, suppose that the token "stein" can only follow "anken" in a task containing the word "Frankenstein". In this case, we can jointly compute the CAMA values for x_{t-1} = "anken", x_t = "stein" by randomizing their prefix $x_{1:t-2}$ and appending "ankenstein" to the input.

More generally, we consider $x_{s:t}$ as one token group if all of $x_{s:t-1}$ are fully determined given a particular value of x_t. In this case, computing the CAMA value involves randomizing $x_{1:s-1}$ with the target values of $x_{s:t}$ being appended.

In our tasks, since we only vary the operand values and keep everything else in the template fixed (e.g., the BOS token, operators, or spaces), we can form token groups of these fixed tokens and the subsequent operand token. Specifically, we set up the following token groups for the two- and three-operand tasks:



Light vs. dark shades represent tokenization, as in Tab. 1, and each background color represents a particular group.

With both the causal attention and the token grouping, computing the CAMA values for a specific group requires the conditional sampling of all values in preceding groups. For example, if we want to compute the CAMA value for the "␣+␣C" group in the three-operand template (green group), we need to sample conditional values of the first two groups (blue and orange). Since all non-operand values are fixed in the template, this amounts to only sampling values of "A" and "B" to empirically estimate the expectation in Eq. 1.

$A+B+C$		$A+B-C$	
Heads Removed	Accuracy	Heads Removed	Accuracy
59 Least Important	98.8%	58 Least Important	98.6%
L15H31	90.6%	L16H1	88.0%
L16H1	54.8%	L15H3	57.8%
L15H13	6.6%	L16H2	27.0%
L16H21	2.0%	L16H3	5.6%
L15H3	0.8%	L16H21	3.2%
-	-	L15H13	2.4%

Table 7: Preserved accuracy across *cumulative* head removals in layers 15 and 16 on the full Llama-3-8B model on tasks $A+B-C$, $A+B+C$. All important heads shown here also appear in the study on $\text{AF1}_{\text{llama}}$ as well (Tab. 4).

A.2 Performance Grid Visualization

Fig. 8 visualize the full grid search across two and three operand arithmetic tasks respectively. These grids are computed across both Llama models, Pythia, and GPT-J. The best AF1 circuit for each model was obtained by empirical observation of all grids for each arithmetic task by each model (and was consistent across both models and all tasks).

A.3 Analysis of Information Transfer Layers in the Full Llama-3-8B Model

To verify whether the information transfer behavior observed in $\text{AF1}_{\text{llama}}$ reflects the mechanisms of the full Llama-3-8B model, we repeat the head-importance analysis from Sec. 4.4 on the full Llama 3 8B model, without any AF1 modifications.

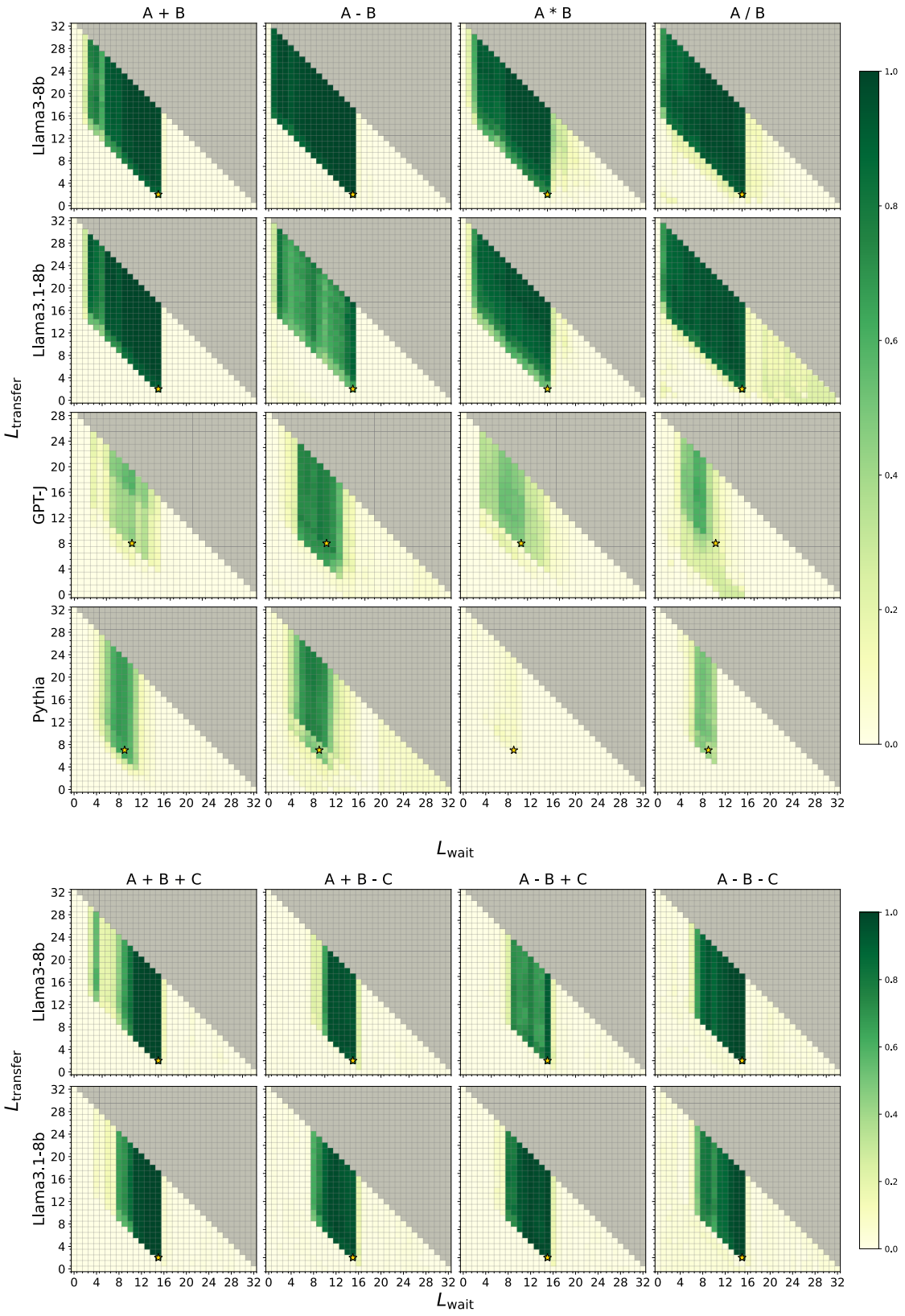


Figure 8: AF1 performance across all models on two- and three-operand tasks.

Results are presented in Tab. 7. We find that, similar to the AF1 setting, model accuracy remains above 98% even after 59 of the 64 heads across the two layers. Furthermore, all of the important heads are found in the AF1 subgraph analysis (i.e., appearing in Tab. 4). This strong correspondence

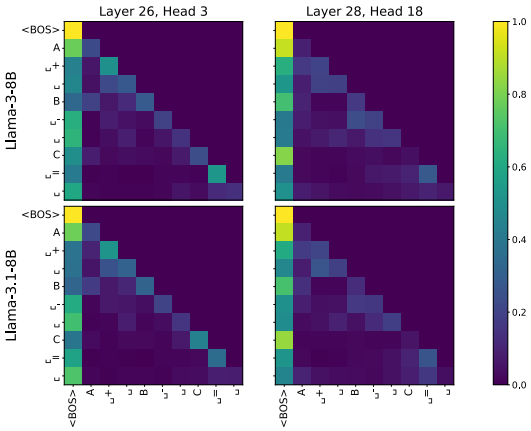


Figure 9: Attention pattern for A+B-C, at attention heads at Layer 26, Head 3 and Layer 28, Head 16 on both Llama-3-8B and Llama-3.1-8B. Attention values were averaged over 100 random samples.

indicates that the AF1 subgraph captures the same core information transfer heads as the full model, rather than exploiting an alternative mechanism.

These findings reinforce the interpretation that the AF1 subgraph is not a shortcut, but rather a faithful, sparsified version of the model's native computation: in both the subgraph and the full model, a small set of heads in layers 15 and 16 mediate the critical information transfer required for arithmetic reasoning.

A.4 Important Attention Heads Visualization

Fig. 9 visualizes the specific attention pattern in the heads which were consistently predicting the right answer (Sec. 4.5). As we can see, there is no clear or interpretable attention patterns in these heads barring a heavy reliance on the <BOS> token, yielding no further insights. Only one prompt is visualized across both key heads, but all tasks demonstrated similarly inconclusive patterns.

A.5 Llama-3.1-8B Internal Representation Analysis

Fig. 10 depicts the internal representation analysis via logit lens for Llama-3.1-8B. The close correspondence to the Llama 3 plot in Fig. 6 suggests that Llama-3.1-8B's arithmetic components remained largely the same despite the improvements on the $A - B - C$ task.

A.6 Details of Alternative CAMA Designs

Direct embedding copy (DEC) simply uses the original embedding $x_t^{(0)}$ for $\tilde{x}_t^{(L_{\text{wait}})}$ in Eq. 1.

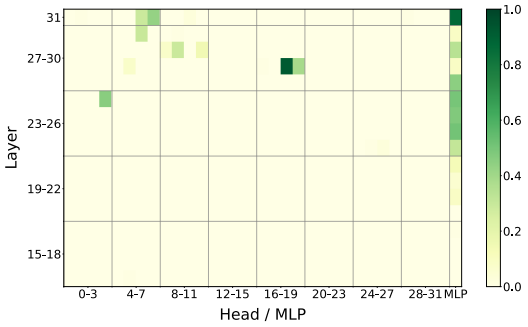


Figure 10: Logit lens top-3 accuracy of for each attention head and the MLP layer on the AF1 subgraph on Llama-3.1-8B.

As mentioned, this likely break's the model's in-distribution assumptions. Even though the information represented at later layers may be the same for these tokens, the embeddings may go through certain transformations which preserve meaning but project through different subspaces of the residual stream.

Random token mean ablation (RTMA) uses completely random tokens in Eq. 1 rather than those drawn from the (context-aware) conditional distribution. That is, instead of drawing tokens from the distribution $\mathbb{P}(\mathbf{x})$, we preserve the prompt length but draw each other token with equal probability from the entire vocab space. This too breaks the in-distribution assumption. While it theoretically captures how the tokens should be projected at any given layer, the lack of conditional awareness "pollutes" the context, erasing any arithmetic meaning encoded.

Self-peek as waiting (SPAW) uses ABP in every waiting layer with $K_t = \{1, t\}$ to make each token $\{x_t\}_{t=1}^{T}$ attend to only itself and the BOS token. In this way, all inter-token computation is erased, but as a result no arithmetic context is encoded through background computation. Additionally, a custom attention mask is directly applied to each token in every waiting layer in every forward pass, causing slowdown issues while CAMA simply substitutes the representations at a key layer.

Isolated forward pass (IFP) runs each token as a standalone two-token prompt (with BOS prepended) for L_{wait} layers and takes the final representation as $\tilde{x}_t^{(L_{\text{wait}})}$ in Eq. 1. Similarly to SPAW, the lack of arithmetic context leads to the inability of the model to perform the necessary information transfer in the key layers.