Understanding In-context Learning of Addition via Activation Subspaces

Xinyan Hu[*1], **Kayo Yin**[1], **Michael I. Jordan**[1 2], **Jacob Steinhardt**[†1], **Lijie Chen**[†1]
[1]UC Berkeley, [2] INRIA

Abstract

To perform in-context learning, language models must extract signals from individual few-shot examples, aggregate these into a learned prediction rule, and then apply this rule to new examples. How is this implemented in the forward pass of modern transformer models? To study this, we consider a structured family of few-shot learning tasks for which the true prediction rule is to add an integer k to the input. We find that Llama-3-8B attains high accuracy on this task for a range of k, and localize its few-shot ability to just three attention heads via a novel optimization approach. We further show the extracted signals lie in a six-dimensional subspace, where four of the dimensions track the unit digit and the other two dimensions track overall magnitude. We finally examine how these heads extract information from individual few-shot examples, identifying a self-correction mechanism in which mistakes from earlier examples are suppressed by later examples. Our results demonstrate how tracking low-dimensional subspaces across a forward pass can provide insight into fine-grained computational structures.

1 Introduction

Large language models (LLMs) have demonstrated remarkable in-context learning (ICL) capabilities, where they can adapt to various new tasks through few-shot demonstrations in the prompt. Motivated by this, researchers have constructed detailed models of in-context learning for small synthetic language models (Garg et al., 2022; Akyürek et al., 2023; von Oswald et al., 2023) as well as coarser-grained treatments of large pretrained models (Olsson et al., 2022; Hendel et al., 2023; Todd et al., 2024). However, little is known about the fine-grained computational structure of in-context learning for large models.

To address this, we perform a detailed analysis of ICL for a structured family of tasks: learning to add a constant number (*"add-k"*). For these tasks, models are given an input sequence "$x_1 \rightarrow y_1\#...\#x_n \rightarrow y_n\#x_q \rightarrow$" where $y_i = x_i + k$ for all i. This task is mathematically well-defined and tractable for an in-depth mechanistic analysis, while exhibiting the core challenge of extracting and applying a pattern from ICL examples. Moreover, modern language models are empirically capable of solving it (they correctly predict $y_q = x_q + k$).

To understand the mechanism behind in-context *add-k*, we study it in Llama-3-8B. We first discover that only a small number of model components are responsible for *add-k*: out of 1024 attention heads, only three do most of the work (§3). We find the task information (i.e., the number k) for each head is encoded in a six-dimensional subspace; zooming into each head, we further decompose it into a four-dimensional space tracking the unit digit using sine and cosine functions and a two-dimensional one tracking the overall magnitude (§4). We also examine how this head aggregates information from individual in-context examples (§5).

To discover the attention heads that are responsible for *add-k*, we introduce a novel approach inspired by previous work on function vectors (Todd et al., 2024), which are vectors that imitate ICL when they are patched into the residual stream (§2.2). We formulate an optimization method that identifies 33 important heads whose outputs yield good ICL performance (§3.1). Through further ablation,

[†]Equal advising.
[*]Correspondence to: Xinyan Hu <xinyanhu@berkeley.edu>.

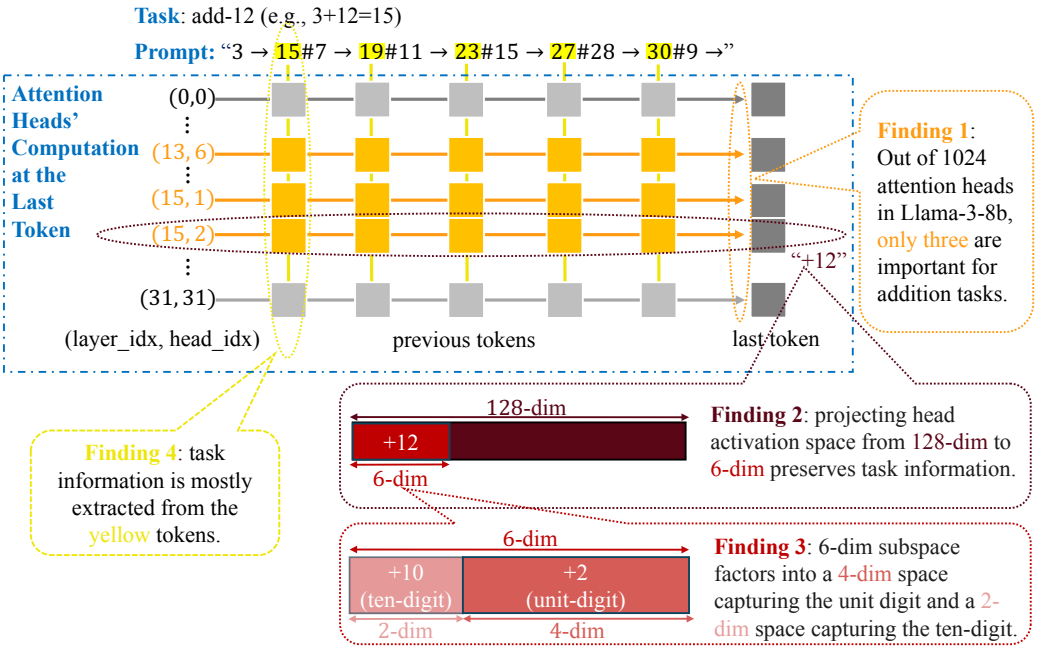


Figure 1: In this paper, we study the activation space of attention heads in Llama-3-8B during in-context learning (ICL) for addition tasks. Our main findings: (1) out of 1024 attention heads in Llama-3-8B, only three are important for these tasks (§4); (2) each head encodes the task information in a six-dimensional subspace (§4.1); (3) a six-dimensional subspace further decomposes into a four-dimensional subspace capturing the unit digit of the task information and a two-dimensional subspace capturing its ten digit (§4.3); and (4) task information is mostly extracted from a subset of tokens (§5). The figure illustrates these findings using an example *add-k* prompt.

we find that all but 3 of these 33 heads can be replaced with their mean value without significantly decreasing the performance (§3.2).

Since these three heads are responsible for computing *add-k*, they must extract signal about k from individual examples and then represent k in the residual stream. To understand how k is represented, zooming into each head, we first find that the effect of each head factors through only a six-dimensional subspace (out of 128 in total) via principal component analysis (PCA) (§4.1). Furthermore, we find that it factors into a four-dimensional subspace tracking the unit digit through trigonometric functions, and a two-dimensional subspace tracking the coarse-grained magnitude (§4.2-4.3). This offers insight on how LLMs internally represent addition, aligning with recent findings showing that (non-ICL) addition is encoded via trigonometric (Zhou et al., 2024) or helical functions (Kantamneni & Tegmark, 2025).

Finally, we examine how this head aggregates information from individual in-context examples. We find that the information is mostly extracted from the y_i token positions, and that this extracted signal encodes a vector representation of $y_i - x_i$ (§5.2). Moreover, we identify a self-correction mechanism: if the signal extracted from one token has an error, later tokens are likely to write in the opposite direction of the error (§5.3). This means that ICL goes beyond simple averaging of inputs and has stateful dynamics.

Our findings suggest that ICL tasks can be implemented through specialized activation subspaces from a small number of attention heads that extract, represent and aggregate task-related information in interpretable ways. We introduce a novel optimization method to identify important attention heads and demonstrate how a focused analysis of a simple ICL task can reveal delicate latent space structures and sophisticated computational strategies in a large language model.

2 Preliminaries

2.1 Model and Task

We begin by specifying the model and the task studied in this paper.

Model. We study Llama3-8B-Instruct, which has 32 layers and 32 attention heads per layer, and a residual dimension of 4096. We denote each head as a tuple (layer index, head index), where both indices range from 0 to 31.

Task. For a constant k, the *add-k* task is to add k to a given input integer x to predict $y = x + k$. In an n-shot ICL prompt, we provide n examples of the format "$x_i \rightarrow y_i$" (where each $y_i = x_i + k$). The examples are concatenated in the prompt using the separator "#", followed by a query "$x_q \rightarrow$"; see Figure 1 for an example of a five-shot ICL prompt. A key advantage of this family of tasks is that ICL prompts for different tasks only differ in k (i.e. $y_i - x_i$ in the prompts) but not the input domain, in contrast to most previous studies (Todd et al., 2024; Hendel et al., 2023).

We construct data for the task by varying $x \in [1, 100]$ and $k \in [1, 30]$ (thus $y \in [2, 130]$) since Llama3-8B is empirically capable of solving the addition task in this range. We consider the following three types of prompts:

1. *five-shot ICL prompt*, where all 5 demonstration examples satisfy $y_i = x_i + k$ for a fixed $k \in [1, 30]$. We also call this add-k ICL.
2. *mixed-k ICL prompt*, where the five demonstration examples are $y_i = x_i + k_i$ for five possibly different k_i values.
3. *zero-shot prompt*, where there are no demonstration examples: the prompt is "$x_q \rightarrow$" for some x_q.

2.2 Activation Patching and Function Vectors

Next, we briefly introduce *activation patching*, a common interpretability technique that will be used throughout the paper, and the *function vector*, a construction that we will use to identify important heads for our ICL tasks.

Activation patching. Activation patching is performed by taking the activations of a model component when the model is run on one prompt, then patching in these activations when the model is run on a different prompt. Patching can either *replace* the model's base activations or *add to* them; we will primarily consider the latter.

Specifically, if z_l is the original value of the residual stream at layer l at the final token position "$\rightarrow$", we patch in the replacement $z_l + v$, where v is constructed from activations on a different input prompt. We choose the layer l at one third of the network's depth and construct v from "function vector" heads, following Todd et al. (2024), as described next.

Function vectors. Function vectors v_k are vectors constructed from the outputs of selected attention heads, designed so that adding v_k to the residual stream of a zero-shot prompt approximates the effect of the five-shot *add-k* task. For example, v_k might be the average output of one or more attention heads across a set of five-shot add-k examples. We define the *intervention accuracy* of v_k as the average accuracy obtained on zero-shot prompts when v_k is added to the residual stream across all tasks k. This metric captures how effectively v_k encodes task-specific information about k. For comparison, we define *clean accuracy* as the accuracy on five-shot prompts without any intervention, also averaged across all tasks k.

In more detail, consider an attention head h: let $h(p)$ denote the output of head h on prompt p at the last token position, and define h_k as the average of $h(p)$ across all five-shot *add-k* prompts.[1] Todd et al. (2024) identified a subset $\mathcal{H}$ of attention heads (for a different set of tasks) such that the vector $v_k = \sum_{h \in \mathcal{H}} h_k$ has high intervention accuracy—that is, adding it to the residual stream of zero-shot prompts effectively recovers few-shot task performance.

[1] Similar to Todd et al. (2024), we approximate this average using 100 randomly generated five-shot *add-k* prompts, where each prompt contains random demonstration examples x_i and the final query input x_q is chosen exactly once from each integer in $[1, 100]$.

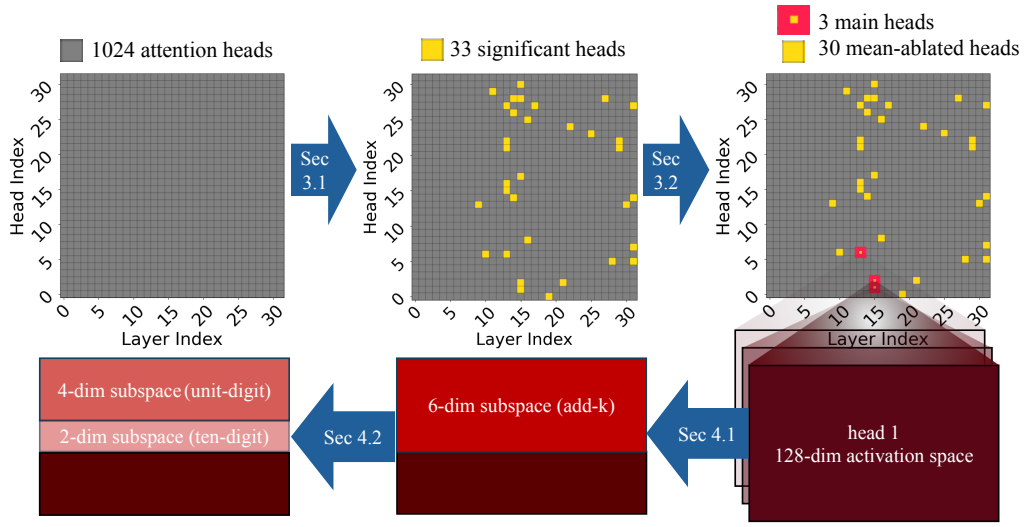


Figure 2: The chain of localization in §3 and §4. We first identify 33 significant attention heads (out of 1024) via a global optimization method (§3.1), then narrow down to 3 main heads while mean-ablating the remaining 30 (§3.2). We next study the structure of the representation of each main head by localizing it to a six-dimensional subspace (§4.1), and decompose it into a four-dimensional subspace encoding the unit digit and a two-dimensional subspace encoding the tens digit (§4.3).

Building on this framework, we consider multiple ways to construct v_k: (1) the task-specific mean h_k over the *add-k* task (as described above); (2) the overall mean $\bar{h}$ (i.e., average across all k), which removes task-specific information about k; or (3) the specific value $h(p)$ on an individual prompt p. Throughout the paper, we call $\bar{h}$ the *mean-ablation* of head h and call h_k the *head vector* of h. Beyond the three choices above, we sometimes project a head's output onto a lower-dimensional subspace or scale it by a coefficient. Unlike Todd et al. (2024), we recover a different set of heads through solving a novel global optimization problem (§3.1) and systematic ablation studies (§3.2), which achieves higher performance (Figure 3).

3 Identifying Three Aggregator Heads

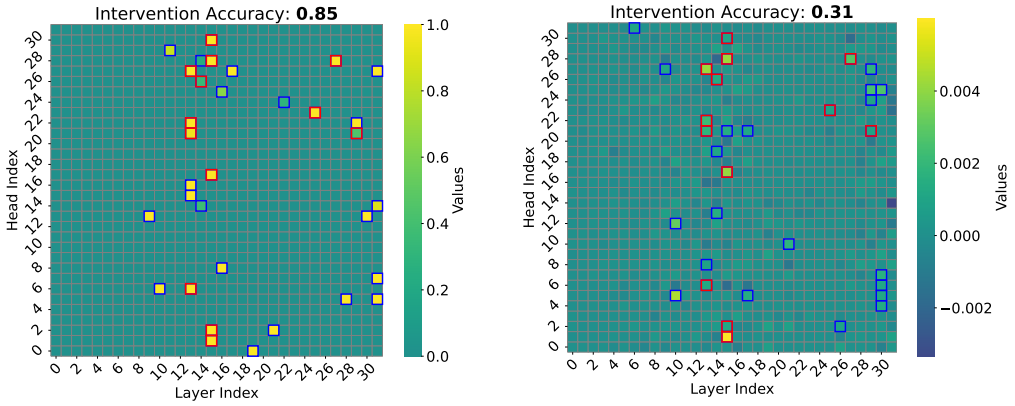
To understand the mechanism behind the *add-k* ICL task, we first need to find out what model components are responsible for performing it. In this section, we find that three heads do most of the work for *add-k*. We find these heads by first solving an optimization problem to identify 33 significant heads out of 1024 (§3.1), and then further narrowing down to three main heads via systematic ablations (§3.2). This process is illustrated in the first row of Figure 2.

3.1 Identifying Significant Heads via Sparse Optimization

We will search for a set of heads that store the information for the *add-k* task, in the sense that their output activations yield good function vectors for *add-k* (§2.2).

Formulating the sparse optimization. More formally, define $v_k(c) = \sum_h c_h \cdot h_k$, the sum of head outputs weighted by c, where h goes over all 1024 heads in the model. (Recall that h_k is the average output of head h averaged across a large dataset of five-shot *add-k* prompts.) We will search for a sparse coefficient vector $c \in [0,1]^{1024}$ such that adding $v_k(c)$ to the residual stream of a zero-shot prompt achieves high accuracy on the *add-k* task.

Let $\ell(x_q, y_q; v)$ be the cross-entropy loss between y_q and the model output when intervening the vector v onto the input "$x_q \rightarrow$"(i.e. replacing z_l with $z_l + v$ in the forward pass on "$x_q \rightarrow$", where z_l



(a) Coefficients of heads.

(b) Average indirect effects of heads (Todd et al., 2024).

Figure 3: Comparison of significant heads identified by our optimized coefficients (left) and by average indirect effects (AIE) from the previous method (Todd et al., 2024) (right). Colors indicate the magnitude of each head's importance (coefficients or AIE). The top 33 heads identified by both methods are highlighted with frames (13 heads common across both methods in red and other 20 heads in blue). Our identified heads yield an intervention accuracy of 0.85, compared to the previous method's accuracy of 0.31. Both methods select heads from similar layers, but our optimization approach is significantly more effective.

is the layer-l residual stream at the last token). We optimize c with respect to the loss

$$\mathcal{L}(c) = \mathbb{E}_{k\in[30]}\mathbb{E}_{x_q\in[100]}[\ell(x_q, x_q+k; v_k(c))] + \lambda\|c\|_1, \tag{1}$$

where the regularization term with weight λ promotes sparsity.

Training details. We randomly select 25 *add-k* tasks from our total 30 tasks for training and in-distribution testing, and use the remaining five tasks only for out-of-distribution testing. For each task *add-k*, we generate 100 zero-shot prompts "x_q $\rightarrow$", where x_q ranges over all integers from $[1, 100]$ and the target output is $x_q + k$, yielding one data point for each (prompt, task) pair. We randomly split the data points of the 25 tasks into training, validation, and test sets in proportions of 0.7, 0.15, and 0.15, respectively. We use AdamW with learning rate 0.01 and batch size 128. We set the regularization rate λ as 0.05, which promotes sparsity while incurring little loss in accuracy. During training, we clip the coefficients c back to $[0, 1]$ if they go out of range after each gradient step.

Results. We get coefficients that achieves high intervention accuracy. In particular, at the final epoch, the average accuracies of intervening on test data for the 25 tasks and the five tasks (0.83 and 0.87) are both close to the average accuracies of doing no intervention (0.89 and 0.92). To identify the important heads for the tasks, we plot the values of the coefficients in the final epoch for each layer and head index (Figure 3a). We find 33 heads have coefficients c_h greater than 0.2, most (21) of which are one. In contrast, the other (991) coefficients are all smaller than 0.01, most (889) of which are zero. We call the 33 heads *significant heads* and denote the set of them as $\mathcal{H}_{\text{sig}}$.

Comparison to previous approach. We compare our optimization approach with the previous method from Todd et al. (2024) for identifying important heads, which selects heads based on average indirect effect (AIE). To perform a fair comparison, we construct our function vector by summing the outputs of our selected heads directly without weighting by coefficients, matching their methodology. Using this construction, we achieve an intervention accuracy of 0.85, close to the clean accuracy of 0.87, indicating that our 33 heads captures most of the necessary information for the *add-k* task. In contrast, selecting the top 33 heads according to AIE yields a much lower intervention accuracy of 0.31.[2] In Figure 3, we visualize the coefficients and AIE values of heads from both methods and highlight the 13 heads that are common to both approaches.

[2] Todd et al. select the top 10 heads in their work, but this yields an even lower accuracy of 0.05 in our setting.

Head	Coefficient	Accuracy
No intervention	N/A	0.87
$(15, 2)$:=Head 1	6	0.85
$(15, 1)$:=Head 2	5	0.83
$(13, 6)$:=Head 3	5	0.66
Any other head	Optimal	0.19

Table 1: Intervened accuracies for scaling up each single head's output by an optimal coefficient. Each of the top three heads (in red) achieves much higher intervention accuracy compared to any other significant head in layer 13 and 15 (in blue).

3.2 Further Refinement via Ablations

We suspect many heads are primarily responsible for storing formatting information (such as ensuring the output appears as a number) rather than encoding information about k itself. Intuitively, while we require the *overall* signal transmitted by these heads, we do not need any information about the specific value of k. To test this hypothesis, we perform *mean-ablations*: replacing each task-specific signal h_k with the overall mean $\overline{h}$ across all values of k (§2.2).

Specifically, we conduct mean-ablations over subsets of the 33 significant heads and measure the resulting intervention accuracy of the corresponding function vectors. Formally, when ablating a subset $\mathcal{H}_0$, the resulting function vector is given by

$$v = \sum_{h \in \mathcal{H}_0} \overline{h} + \sum_{h \in \mathcal{H}_{\text{sig}} \setminus \mathcal{H}_0} h_k. \tag{2}$$

Here, heads in $\mathcal{H}_0$ contribute only their overall mean signals, while heads in $\mathcal{H}_{\text{sig}} \setminus \mathcal{H}_0$ retain their task-specific information.

Focusing on two layers via layer-wise ablation. To efficiently narrow down the important heads, we first perform mean-ablations at the level of layers. From Figure 3a, we observe that the significant heads $\mathcal{H}_{\text{sig}}$ are concentrated primarily in the middle and late layers. We speculate that heads in the late layers mainly contribute to formatting the output, as they appear too late in the computation to meaningfully interact with the query. After trying different sets of layers, we found that mean-ablating all significant heads outside layers 13 and 15 still achieves an intervention accuracy of 0.83, while mean-ablating any other combination of layers causes negligible drops in accuracy (Appendix A, Table 3). After these ablations, only 11 heads located in layers 13 and 15 remain.

Identifying three final heads via head-ablation. To understand the individual contributions of each head within layers 13 and 15, we perform mean-ablations at the level of individual heads. We first assess the intervention accuracy when retaining only the output of a single head while mean-ablating all other significant heads; however, this generally results in low accuracy. We hypothesize that the output magnitude of a single head is too small to significantly influence the model output, even if it encodes task-relevant information. To amplify each head's effect, we scale its output by a coefficient (e.g., 5). We find that three heads—head 1 = $(15, 2)$, head 2 = $(15, 1)$, and head 3 = $(13, 6)$—achieve intervention accuracies close to the clean accuracy when appropriately scaled, while all other heads show much lower accuracies regardless of scaling (Table 1). This suggests that these three heads individually encode the task information much better than any others.

Finally, to remove the need for scaling while maintaining high intervention accuracy, we sum the outputs of these three heads (each with a coefficient of one) and mean-ablate all others. We find that summing the top three, top two, and only the top head yields intervention accuracies of 0.79, 0.61, and 0.21, respectively. The three heads are thus collectively sufficient for performing the *add-k* task.

Validating necessity of the three heads via ablating them in five-shot ICL. So far, we have studied these three heads mainly through their contribution to the function vector v_k. We next directly test their necessity in the original five-shot ICL setting, by ablating the outputs of these three heads when running the model on many random five-shot ICL prompts. Our experiment shows that mean-ablating these three heads in five-shot prompts yields an accuracy of 0.43, sharply decreasing from the clean accuracy 0.87 by half. For comparison, we mean-ablate 20 random sets of three significant heads

(other than head 1,2,3); their accuracies remain close to the clean accuracy: 95% of them have accuracy at least 0.86.

4 Characterizing the Aggregator Subspace

For the model to perform *add-k*, it has to infer the task information (the number k) from the ICL examples. Our next goal is thus to understand how task information is represented in the activation space. Since we have identified three aggregator heads that carry almost all of this information, we can now focus on analyzing the representation space of these three heads.

In this section, we dissect their activation spaces in three stages: (1) **Localize** a six–dimensional task subspace in each head via principal component analysis (PCA) (§4.1); (2) **Rotate** this subspace into orthogonal *feature directions* aligned with sinusoidal patterns across k (§4.2); (3) **Decompose** the six-dimensional space into a four-dimensional *unit-digit* subspace and a two-dimensional *magnitude* subspace that separately encode the units and tens of the answer (§4.3).

4.1 Localizing to Six-dimensional Subspace

To reduce the 128-dimensional head activation to a more tractable space to study, we first perform PCA on the 30 task vectors and find that just six directions can explain 97% of the task variance (Figure 4). We then check that the function vectors found earlier remain effective after projecting onto the subspace. Specifically, we replace each head vector h_k with its projection onto the subspace $\tilde{h}_k$ to obtain a new function vector $\tilde{v}_k = \sum_{h \in \mathcal{H}} \tilde{h}_k$ (a variant of $v_k = \sum_{h \in \mathcal{H}} h_k$ in §2.2). We find that $\tilde{v}_k$ has intervention accuracy 0.76, which is close to the intervention accuracy of 0.79 before projection (§3.2). Thus, we confine our study to this concise six-dimensional subspace for each head. The rest of the section focuses on head 1 and we defer analogous results for head 2 and 3 to appendix B.

4.2 Identifying Feature Directions Encoding Periodic Patterns

To understand how the six-dimensional subspace of a head represents the task information, we first examine the coordinates of the head vectors (their inner products with the PCs) as a function of k. This reveals partially periodic patterns in the first five components (Figure 5).

This motivates us to linearly transform the six PCs to find directions that encode pure periodic patterns. Mathematically, if we can find a linear transformation of the six PC-coordinate functions that fits trigonometric functions, then by applying the transformation on the PCs, we can obtain six directions whose coordinate functions encode the periodicity.

To find trigonometric functions to fit, we searched over different periods and phases and performed least squares regression. We found six trigonometric functions at periods 2, 5, 10, 10, 25, and 50 that could be expressed as functions of the top 6 PCs with low regression error (Figure 6). We apply the resulting linear transformation to the six PCs to obtain a new set of directions that encode these six pure periodic patterns, which we call *feature directions*.

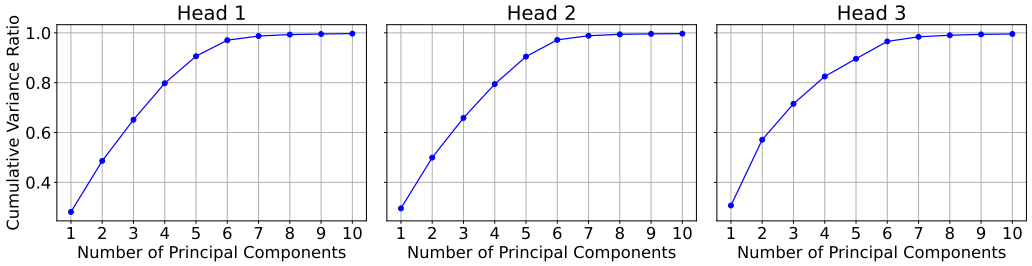


Figure 4: The explained variance ratio vs. number of PCs for each head. The first six PCs make up most of the explained variance (97%) for each head.

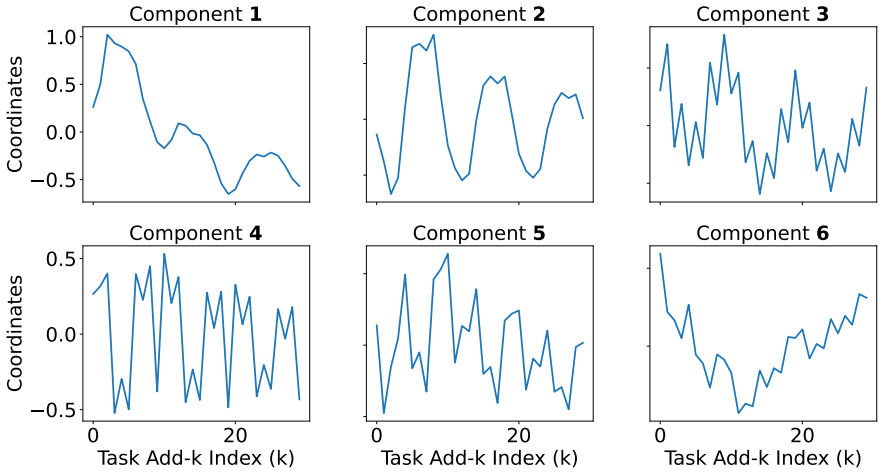


Figure 5: Coordinates of head 1's vectors (their inner products with a PC) for the first six PCs across different add-k tasks. The coordinates in the first five PCs reveal periodic patterns.

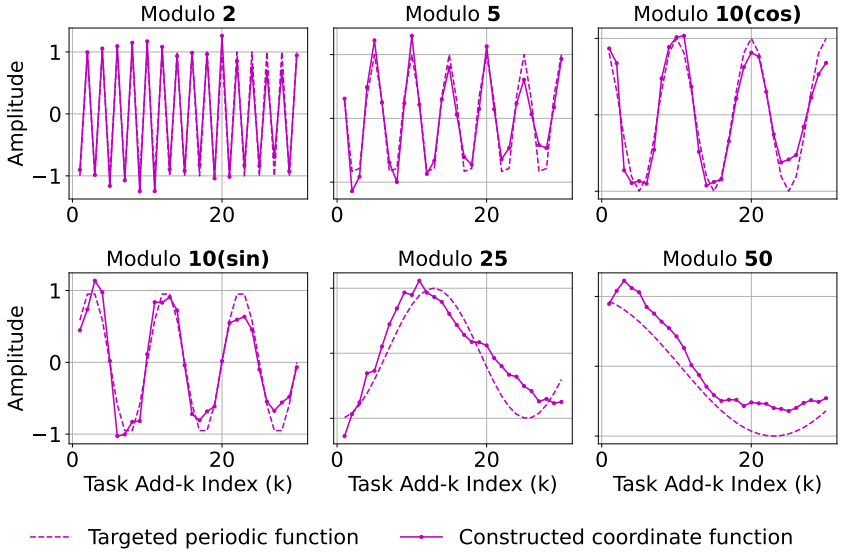


Figure 6: The coordinate functions (inner product with feature directions) of head 1's vectors can fit six trigonometric functions with periods $2, 5, 10, 10, 25$, and 50 well by linear transformation.

4.3 Decomposing to Subspaces Encoding Subsignals

Leveraging the feature directions identified previously, we decompose the head activation subspace into lower-dimensional components that separately encode different subsignals relevant to the task—in this case, the units digit and tens digit.

By construction, the coordinate function of each feature directions (viewed as linear projections of h_k) is a periodic function of k. Mathematically, a feature direction with period T carries task information from the head vectors with "modulo T". Based on this, we hypothesize: (i) the feature direction corresponding to period two, which we call the "parity direction", encodes the parity of k in the *add-k* task; (ii) the subspace spanned by the feature directions with periods $2, 5, 10$, which we call the "unit subspace", encodes the unit digit of k; (iii) the subspace spanned by the directions with periods $25, 50$, which we call the "magnitude subspace", encodes the coarse magnitude (i.e., the tens digit) of k.

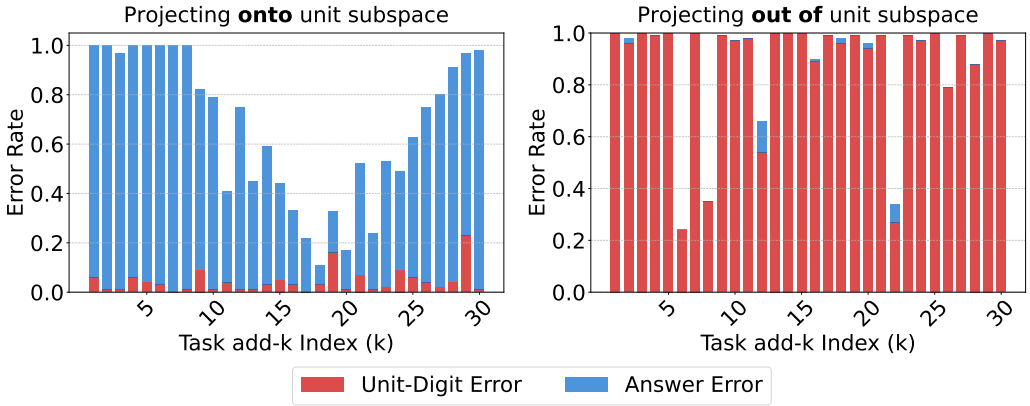


Figure 7: The error rates for the unit digit and the final answer across tasks when projecting head 1's vectors **onto** the "unit subspace" (left) and **out of** the "unit subspace" (right). Projecting onto the unit subspace results in a low unit-digit error rate even when final-answer error remains high, while projecting out leads to high unit-digit error rates that almost fully account for the final-answer errors. This confirms that the unit subspace specifically encodes the unit-digit signal.

We verify these hypotheses through causal intervention. In Figure 7, we validate hypothesis (ii): we establish (1) **sufficiency** by showing that projecting a head vector *onto* the subspace preserves the relevant task signal; and (2) **necessity** by showing that projecting a head vector *out of* the subspace (i.e., onto its orthogonal complement) destroys the relevant task signal. We include analogous experimental results for the two other hypotheses of head 1 in Appendix B.2.

5 Signal Extractors of ICL Examples

Previously, we localized the model's behavior to three heads and their corresponding six-dimensional subspaces, then examined how the model represents the task information (k for *add-k*) inferred from the ICL examples in one subspace. Now, we analyze *how* the model extracts the task information from the ICL examples.

In this section, we find that: (1) the signal is primarily gathered from the y_i tokens in each example $x_i \to y_i$; (2) each example $x_i \to y_i$ individually contributes a signal $y_i - x_i$ in the subspace even on "mixed" in-context examples with conflicting task information; and (3) when all examples $x_i \to y_i$ share the same value for $y_i - x_i$, the extracted signals exhibit a *self-correction* behavior. For simplicity, we focus on head 1 in this section.

5.1 Mathematical Observation: Tracing Subspace back to Previous Tokens

We begin with a mathematical observation that lets us trace the subspace at the final token back to corresponding subspaces at earlier token positions. Intuitively, a head's output at the last token is a weighted sum of transformed residual streams from the previous tokens, with the weights given by the attention scores. Thus, the signal extracted from previous tokens is the transformed residual stream at that token.

Formally, a head h's output at the last token of a prompt p can be written as

$$h(p) = \sum_{t \in p} \alpha_t \cdot O_h V_h \cdot z_t, \tag{3}$$

where α_t is the attention score from the last token to each token t, satisfying $\sum_{t \in p} \alpha_t = 1$, z_t is the residual stream input to the head h at token t, V_h is the value matrix, and O_h is the output matrix mapping from head-dimensional space to model-dimensional space.

Let W_h denote the projection matrix onto the six-dimensional subspace for head h. Then the projected signal at the final token, $W_h \cdot h(p)$, can be decomposed into contributions from previous tokens as $W_h \cdot \alpha_t O_h V_h z_t$, each lying in the transformed head subspace defined by the matrix $W_h \cdot O_h V_h$. In

the following subsection, we analyze the magnitudes and directions of these signals, and study how signals from different examples interact.

5.2 Signal Extractor for each Example

To understand how examples contribute to model generation at the last token, we identify which tokens contribute the most, then examine what information they provide. By the analysis above, the task-signal contribution of each previous token to the final token through the head h is $\alpha_t W_h O_h V_h z_t$. This can be decomposed into two parts: (1) **extracted information:** $W_h O_h V_h z_t$, the residual stream input projected into the relevant subspace; and (2) **aggregation weight:** α_t, the attention score of the final token to the previous token. We plot the norms of the extracted information and the aggregation weights for a random mixed-k ICL prompt in Figure 8. Both the strength of the extracted information and the aggregation weights peak at y_i tokens.

We next examine what specific information is extracted from each of these tokens. To do so, we measure the inner product $\langle W_h O_h V_h z_t, \tilde{h}_k \rangle$ between the extracted information and the head vector (projected onto the subspace and normalized to have unit norm) for each task k. In Figure 9, we plot this quantity for a random mixed-k ICL prompt for each token y_i ($i \in \{1, \ldots, 5\}$) and each task $k \in \{1, \ldots, 30\}$. We find that the inner product consistently peaks at $k = y_i - x_i$, indicating that the model extracts the information of $y_i - x_i$ from the corresponding example $x_i \to y_i$.

5.3 Signal Correlation among Examples

Having studied the signal extracted from each individual example in the previous subsection, we next study how signals from different examples interact to execute the ICL task. To do so, we compute the correlation between the extracted signals from different demonstration examples: for each y_i token, we first compute the inner product between the residual stream input to head 1, z_t, and the corresponding task vector h_k, where $k = y_i - x_i$. Then, we compute the correlation of these measures across each pair of five positions over 100 *add-k* prompts, yielding $\binom{5}{2}$ correlation values per task.

To analyze the correlation, we sum the negative correlation values and positive correlation values respectively for each task, and calculate the various statistics (max, average and min) over all tasks (Table 2). The negative correlation sum is significantly higher than the positive correlation sum for all three statistics, indicating that the signals from any two demonstration examples are mostly negatively correlated. This suggests a *self-correction* mechanism: intuitively, when the head extracts a noisy signal from one example, signals from subsequent examples are more likely to correct the error, thereby stabilizing the final representation.

6 Related Work

Our work builds on a growing body of research that aims to uncover circuits and internal computations of language models (Olah et al., 2020; Wang et al., 2022; Wu et al., 2023; Cunningham et al., 2023;

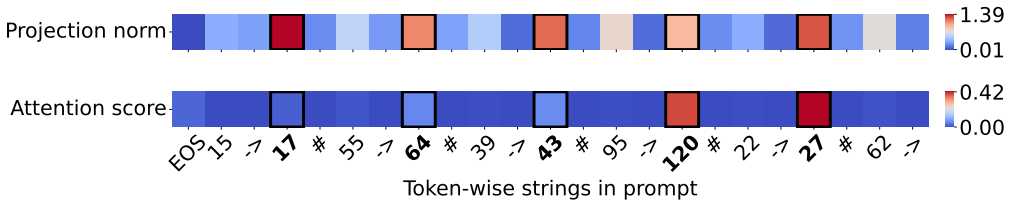


Figure 8: For a mixed-k ICL prompt, the strength of the task-signal contribution of each previous token t to the final token, $\|\alpha_t W_h O_h V_h z_t\|$. Decomposing it into two parts: (1) the norm of the extracted information $\|W_h O_h V_h z_t\|$ (top), and (2) the attention score from the final token α_t (bottom), both parts consistently peak at the tokens $t = y_i$ (in bold).

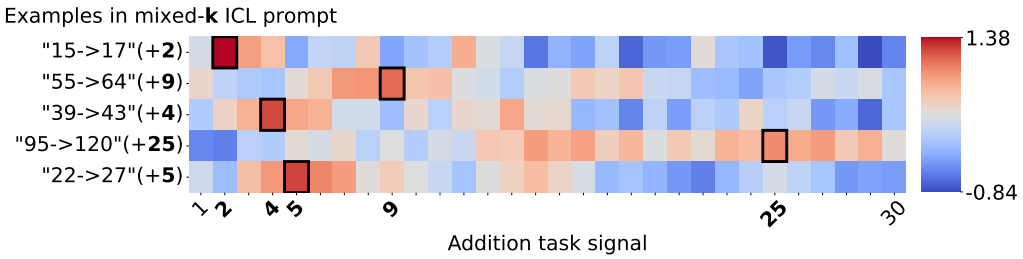


Figure 9: For a random mixed-k ICL prompt, the inner product between the projected signal from each y_i and each head vector (projected onto the subspace and normalized to have unit norm) $\tilde{h}_k$. For each example $x_i \to y_i$, the inner product peaks for $k = y_i - x_i$, indicating that the model extracts the information of $y_i - x_i$ from the corresponding example $x_i \to y_i$.

Statistics over tasks	Sum of **negative** correlations	Sum of **positive** correlations
Average	-2.01	0.27
Min	-1.40	0.07
Max	-2.34	0.54

Table 2: The statistics (min, max, and average) of the absolute values of negative correlation sum and positive correlation sum respectively over the 30 tasks. The negative correlation sum is significantly higher than the positive correlation sum for all three statistics, which indicates that the signals from any two demonstration examples are mostly negatively correlated, suggesting a *self-correction* mechanism.

Feng & Steinhardt, 2023; Gould et al., 2023; Park et al., 2024). Below we discuss three particular threads that are most relevant to this paper.

Interpreting arithmetic tasks. A recent line of work examines how LLMs perform arithmetic (Stolfo et al., 2023; Hanna et al., 2023; Nikankin et al., 2024; Maltoni & Ferrara, 2024), and in particular addition (Nanda et al., 2023; Zhong et al., 2023; Zhou et al., 2024; Kantamneni & Tegmark, 2025). Zhou et al. (2024) find that pre-trained LLMs perform addition using Fourier features, and Kantamneni & Tegmark (2025) find that mid-sized LLMs compute addition using a "clock" algorithm via a helix representation of numbers. Different from them, our work is the first to analyze addition for in-context learning in LLM, and interestingly we find similar representation structures to Zhou et al. (2024) and Kantamneni & Tegmark (2025).

Interpreting in-context learning. Researchers have constructed detailed models of in-context learning (ICL) for small transformer models in standard supervised learning problems such as linear regression (Garg et al., 2022; Akyürek et al., 2023; Zhang et al., 2023; Li et al., 2023; Wu et al., 2024), as well as more complex settings (von Oswald et al., 2023; Bai et al., 2023; Bietti et al., 2023; Reddy, 2023; Guo et al., 2023; Nichani et al., 2024). For large pretrained models, there exist coarser-grained treatments attributing ICL performance to either induction heads (Olsson et al., 2022; Singh et al., 2024; Crosbie & Shutova, 2025; Bansal et al., 2023) or function vector (FV) heads (Todd et al., 2024; Hendel et al., 2023). Yin & Steinhardt (2025) compares the two types of heads and finds that few-shot ICL performance depends primarily on FV heads. Motivated by this, we study function vector heads in details for a family of ICL tasks, introducing a novel optimization method, which achieves better performance than the method in Todd et al. (2024). Another difference from Todd et al. (2024) is that our tasks have the same input domain, ensuring that the ICL prompts for different tasks differ only in the task information.

Causal analysis. There has been a line of research that proposes methods to understand the causal influence of model components on model behavior, such as by probing (Conneau et al., 2018; Hewitt & Manning, 2019; Clark et al., 2019). Our methodological approach follows recent developments in revealing causal effects of model components by interventions on internal states of models (Vig et al.,

2020; Geiger et al., 2021). In particular, we draw inspiration from causal mediation analysis used in Todd et al. (2024), activation patching (Meng et al., 2022), and causal scrubbing (Chan et al., 2022).

7 Discussion

In this work, we provided a detailed mechanistic analysis of in-context learning for addition tasks in Llama-3-8B. We found that a small number of attention heads operating in low-dimensional subspaces can extract, represent, and aggregate ICL task information in structured and interpretable ways. We performed our analysis in five steps:

1. Use sparse optimization to identify important attention heads whose outputs construct effective function vectors for ICL tasks (§3.1).
2. Localize task information to a smaller subset of heads via ablations (§3.2).
3. Further localize to low-dimensional subspaces via PCA on each remaining head (§4.1).
4. Examine this subspace qualitatively, which uncovered periodic patterns in the activation space (§4.2) that decomposed into interpretable subspaces encoding unit-digit and magnitude information (§4.3).
5. Exploit algebraic structure in the transformer to connect "aggregation" subspaces at the final token position with "extraction" subspaces at the earlier y_i tokens (§5).

This same methodology (identify important heads, restrict to relevant subspaces, and examine the remaining information qualitatively) could be extended to other models and tasks. Most steps in our methodology also scale easily: the sparse optimization is fully automatic. While mean ablation involved some qualitative judgment, we can fold both of these steps into a single optimization task that mean ablates some heads while fully removing others. PCA is also automatic. For the final step that involves a qualitative examination of the subspaces, future work could explore automating this step using AI systems.

There are a few other questions stemming from our work. First, the three important heads are capable of individually representing and extracting information, but Llama3-8B relies on all of them—how does it jointly aggregate information from these three heads to produce a final answer? Second, we observe that heads implement a "self-correction" mechanism indicated by the negative correlation between examples signals, but the exact mechanism of this self-correction remains to be discovered.

Our findings demonstrate how a focused study can reveal sophisticated latent structures and delicate computational strategies within a LLM. We hope this motivates more work to dive deep into the LLM inner structures and obtain more understanding of LLM latent spaces and computations.

Acknowledgments

We thank Lisa Dunlap, Alex Pan, Tony Lian, Anya Ji, Jiaxin Ge, Junyi Zhang, Hanlin Zhu, Shu Liu and Baifeng Shi for their helpful feedback and suggestions. KY is supported by the Vitalik Buterin Ph.D. Fellowship in AI Existential Safety. MJ is supported by the European Union (ERC-2022-SYG-OCEAN-101071601). LC is supported by a Miller Research Fellowship. JS is supported by Open Philanthropy, Simons Foundation, and NSF.

References

Ekin Akyürek, Dale Schuurmans, Jacob Andreas, Tengyu Ma, and Denny Zhou. What learning algorithm is in-context learning? investigations with linear models, 2023. URL https://arxiv.org/abs/2211.15661.

Yu Bai, Fan Chen, Huan Wang, Caiming Xiong, and Song Mei. Transformers as statisticians: Provable in-context learning with in-context algorithm selection, 2023. URL https://arxiv.org/abs/2306.04637.

Hritik Bansal, Karthik Gopalakrishnan, Saket Dingliwal, Sravan Bodapati, Katrin Kirchhoff, and Dan Roth. Rethinking the role of scale for in-context learning: An interpretability-based case study at

66 billion scale. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (eds.), *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 11833–11856, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-long.660. URL https://aclanthology.org/2023.acl-long.660/.

Alberto Bietti, Vivien Cabannes, Diane Bouchacourt, Herve Jegou, and Leon Bottou. Birth of a transformer: A memory viewpoint, 2023. URL https://arxiv.org/abs/2306.00802.

Lawrence Chan, Adria Garriga-Alonso, Nicholas Goldowsky-Dill, Ryan Greenblatt, Jenny Nitishinskaya, Ansh Radhakrishnan, Buck Shlegeris, and Nate Thomas. Causal scrubbing: A method for rigorously testing interpretability hypotheses. In *AI Alignment Forum*, pp. 10, 2022.

Kevin Clark, Urvashi Khandelwal, Omer Levy, and Christopher D. Manning. What does BERT look at? an analysis of BERT's attention. In Tal Linzen, Grzegorz Chrupała, Yonatan Belinkov, and Dieuwke Hupkes (eds.), *Proceedings of the 2019 ACL Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pp. 276–286, Florence, Italy, August 2019. Association for Computational Linguistics. doi: 10.18653/v1/W19-4828. URL https://aclanthology.org/W19-4828/.

Alexis Conneau, German Kruszewski, Guillaume Lample, Loïc Barrault, and Marco Baroni. What you can cram into a single vector: Probing sentence embeddings for linguistic properties, 2018. URL https://arxiv.org/abs/1805.01070.

Joy Crosbie and Ekaterina Shutova. Induction heads as an essential mechanism for pattern matching in in-context learning, 2025. URL https://arxiv.org/abs/2407.07011.

Hoagy Cunningham, Aidan Ewart, Logan Riggs, Robert Huben, and Lee Sharkey. Sparse autoencoders find highly interpretable features in language models, 2023. URL https://arxiv.org/abs/2309.08600.

Jiahai Feng and Jacob Steinhardt. How do language models bind entities in context? *arXiv preprint arXiv:2310.17191*, 2023.

Shivam Garg, Dimitris Tsipras, Percy S Liang, and Gregory Valiant. What can transformers learn in-context? a case study of simple function classes. *Advances in Neural Information Processing Systems*, 35:30583–30598, 2022.

Atticus Geiger, Hanson Lu, Thomas Icard, and Christopher Potts. Causal abstractions of neural networks. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, volume 34, pp. 9574–9586. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/file/4f5c422f4d49a5a807eda27434231040-Paper.pdf.

Rhys Gould, Euan Ong, George Ogden, and Arthur Conmy. Successor heads: Recurring, interpretable attention heads in the wild. *arXiv preprint arXiv:2312.09230*, 2023.

Tianyu Guo, Wei Hu, Song Mei, Huan Wang, Caiming Xiong, Silvio Savarese, and Yu Bai. How do transformers learn in-context beyond simple functions? a case study on learning with representations, 2023. URL https://arxiv.org/abs/2310.10616.

Michael Hanna, Ollie Liu, and Alexandre Variengien. How does gpt-2 compute greater-than?: Interpreting mathematical abilities in a pre-trained language model, 2023. URL https://arxiv.org/abs/2305.00586.

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

John Hewitt and Christopher D. Manning. A structural probe for finding syntax in word representations. In Jill Burstein, Christy Doran, and Thamar Solorio (eds.), *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 4129–4138, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1419. URL https://aclanthology.org/N19-1419/.

Subhash Kantamneni and Max Tegmark. Language models use trigonometry to do addition. *arXiv preprint arXiv:2502.00873*, 2025.

Yingcong Li, Muhammed Emrullah Ildiz, Dimitris Papailiopoulos, and Samet Oymak. Transformers as algorithms: Generalization and stability in in-context learning. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (eds.), *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 19565–19594. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/li23l.html.

Davide Maltoni and Matteo Ferrara. Arithmetic with language models: From memorization to computation. *Neural Networks*, 179:106550, November 2024. ISSN 0893-6080. doi: 10.1016/j.neunet.2024.106550. URL http://dx.doi.org/10.1016/j.neunet.2024.106550.

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

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

Eshaan Nichani, Alex Damian, and Jason D. Lee. How transformers learn causal structure with gradient descent, 2024. URL https://arxiv.org/abs/2402.14735.

Yaniv Nikankin, Anja Reusch, Aaron Mueller, and Yonatan Belinkov. Arithmetic without algorithms: Language models solve math with a bag of heuristics, 2024. URL https://arxiv.org/abs/2410.21272.

Chris Olah, Nick Cammarata, Ludwig Schubert, Gabriel Goh, Michael Petrov, and Shan Carter. Zoom in: An introduction to circuits. *Distill*, 2020. doi: 10.23915/distill.00024.001. https://distill.pub/2020/circuits/zoom-in.

Catherine Olsson, Nelson Elhage, Neel Nanda, Nicholas Joseph, Nova DasSarma, Tom Henighan, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, et al. In-context learning and induction heads. *arXiv preprint arXiv:2209.11895*, 2022.

Kiho Park, Yo Joong Choe, and Victor Veitch. The linear representation hypothesis and the geometry of large language models, 2024. URL https://arxiv.org/abs/2311.03658.

Gautam Reddy. The mechanistic basis of data dependence and abrupt learning in an in-context classification task, 2023. URL https://arxiv.org/abs/2312.03002.

Aaditya K. Singh, Ted Moskovitz, Felix Hill, Stephanie C. Y. Chan, and Andrew M. Saxe. What needs to go right for an induction head? a mechanistic study of in-context learning circuits and their formation, 2024. URL https://arxiv.org/abs/2404.07129.

Alessandro Stolfo, Yonatan Belinkov, and Mrinmaya Sachan. A mechanistic interpretation of arithmetic reasoning in language models using causal mediation analysis. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 7035–7052, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.435. URL https://aclanthology.org/2023.emnlp-main.435/.

Eric Todd, Millicent L. Li, Arnab Sen Sharma, Aaron Mueller, Byron C. Wallace, and David Bau. Function vectors in large language models. In *Proceedings of the 2024 International Conference on Learning Representations*, 2024.

Jesse Vig, Sebastian Gehrmann, Yonatan Belinkov, Sharon Qian, Daniel Nevo, Yaron Singer, and Stuart Shieber. Investigating gender bias in language models using causal mediation analysis. *Advances in neural information processing systems*, 33:12388–12401, 2020.

Johannes von Oswald, Eyvind Niklasson, Ettore Randazzo, João Sacramento, Alexander Mordvintsev, Andrey Zhmoginov, and Max Vladymyrov. Transformers learn in-context by gradient descent, 2023. URL https://arxiv.org/abs/2212.07677.

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

Jingfeng Wu, Difan Zou, Zixiang Chen, Vladimir Braverman, Quanquan Gu, and Peter L. Bartlett. How many pretraining tasks are needed for in-context learning of linear regression?, 2024. URL https://arxiv.org/abs/2310.08391.

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

Kayo Yin and Jacob Steinhardt. Which attention heads matter for in-context learning? *2025 International Conference on Machine Learning*, July 2025. URL https://arxiv.org/abs/2502.14010.

Ruiqi Zhang, Spencer Frei, and Peter L. Bartlett. Trained transformers learn linear models in-context, 2023. URL https://arxiv.org/abs/2306.09927.

Ziqian Zhong, Ziming Liu, Max Tegmark, and Jacob Andreas. The clock and the pizza: Two stories in mechanistic explanation of neural networks, 2023. URL https://arxiv.org/abs/2306.17844.

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

A Additional table in §3.2

In §3.2, we did systematic ablation studies to narrow down the tasks to three main heads from 33 significant heads. The first step of the ablation studies is layer-wise ablation, where we mean-ablate the significant heads in a subset of layers. We include the experimental results here, which narrow down to layer 13 and 15.

Layer	Accuracy
No intervention	0.87
$[0, 31]$	0.85
$[0, 15]$	0.83
$[13, 15]$	0.83
$\{13, 15\}$	0.83
$\{14, 15\}$	0.69
$\{13, 14\}$	0.25
$\{15\}$	0.71
$\{13\}$	0.27
$\{14\}$	0.03
$[0, 31] \setminus \{13, 15\}$	0.05

Table 3: Intervention accuracies for keeping the significant heads in the selected layers and mean-ablating the significant heads in the remaining layers. We first narrow down to the layers before layer 15, then the range of $[13, 15]$ and finally $\{13, 15\}$ (in red), which all almost preserve the clean accuracy of 0.87, while other combinations lead to substantial drops in accuracy, especially when mean-ablating layers 13 and 15 (in blue).

B Additional figures in §4

In §4.2 and §4.3, we focused on head 1 for simplicity of exposition. Here we include the analogous experimental results for head 2 and 3.

B.1 Additional figures for §4.2

In §4.2, we found six trigonometric functions that can be linearly fitted by the coordinate functions of head 1 (Figure 6). Here we show analogous experimental results for head 2 and 3.

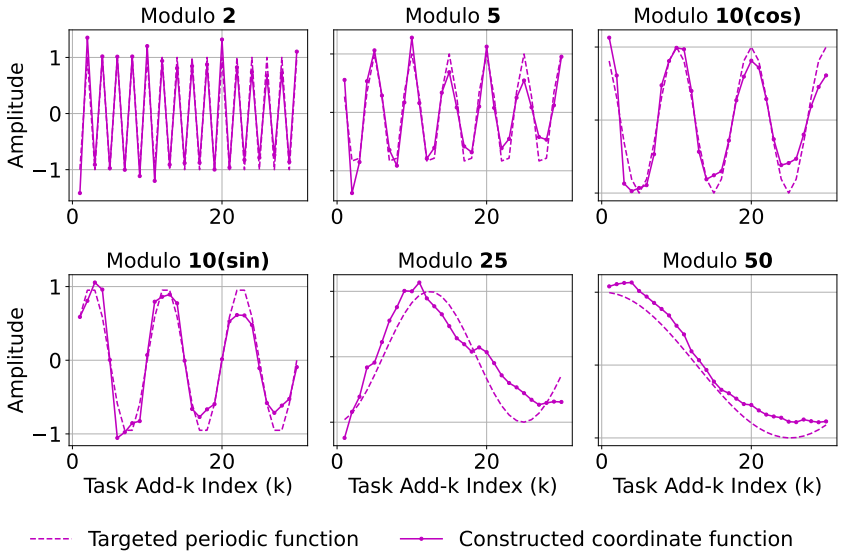


Figure 10: The coordinate functions (inner product with feature directions) of head 2's vectors can fit six trigonometric functions with periods $2, 5, 10, 10, 25$, and 50 well by linear transformation.

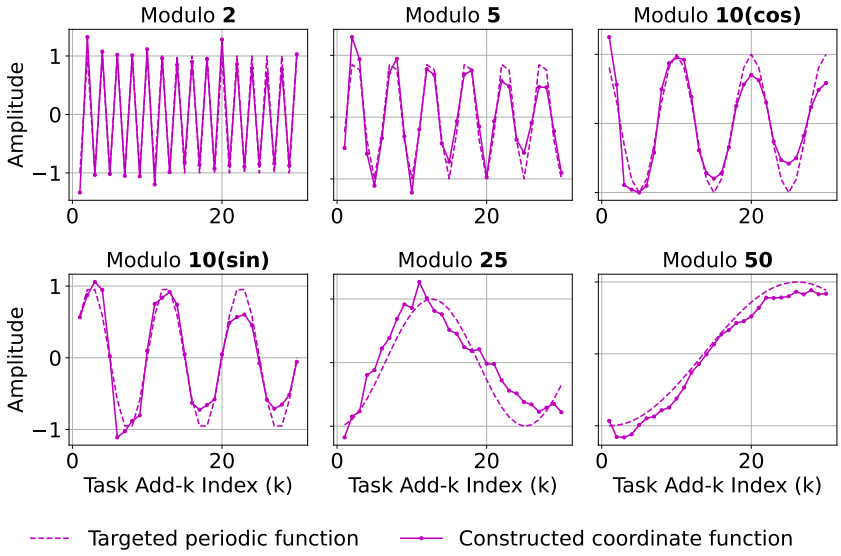


Figure 11: The coordinate functions (inner product with feature directions) of head 3's vectors can fit six trigonometric functions with periods $2, 5, 10, 10, 25$, and 50 well by linear transformation.

B.2 Additional figures for §4.3

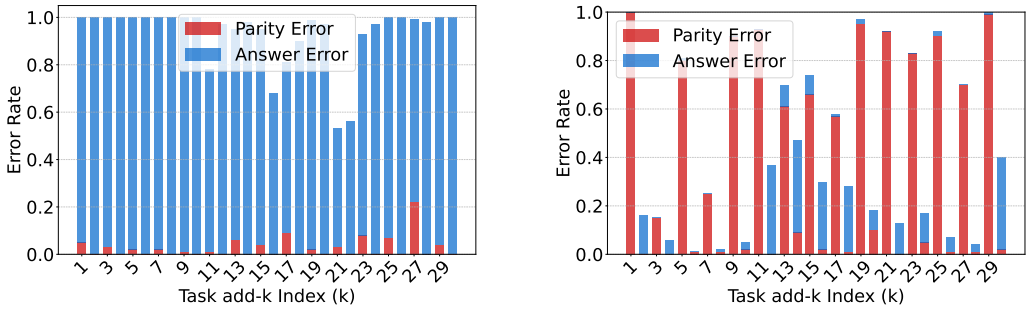
In §4.3, we showed experimental results validating the hypothesis (ii) for head 1 (Figure 7). Here we show analogous experimental results validating the hypotheses (i) and (iii) for head 1, as well as all three hypotheses for heads 2 and 3. Recall that the three hypotheses are as follows:

(i) the feature direction corresponding to period two, which we call the "parity direction", encodes the parity of k in the *add-k* task;

(ii) the subspace spanned by the feature directions with periods $2, 5, 10$, which we call the "unit subspace", encodes the unit digit of k;

(iii) the subspace spanned by by the directions with periods $25, 50$, which we call the "magnitude subspace", encodes the coarse magnitude (i.e., the tens digit) of k.

We first show the experimental results validating hypotheses (i) and (iii) for head 1. Projecting out of the parity direction doesn't lead to high errors for the parity and the final answer across all tasks, which might be because parity is relatively easy to obtain (e.g., random choice leads to 0.5 accuracy).



(a) Head 1 projected onto parity direction

(b) Head 1 projected out of parity direction

Figure 12: Validation of hypothesis (i) for head 1.

(a) Head 1 projected onto magnitude subspace

(b) Head 1 projected out of magnitude subspace

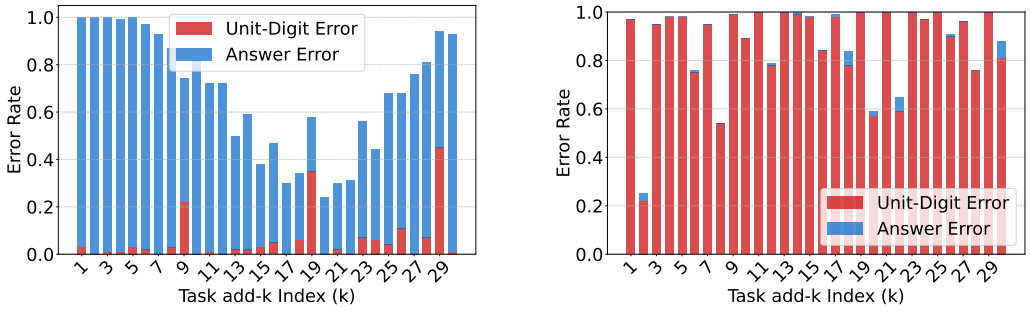
Figure 13: Validation of hypothesis (iii) for head 1.

We then show analogous results for head 2.

(a) Head 2 projected onto parity direction

(b) Head 2 projected out of parity direction

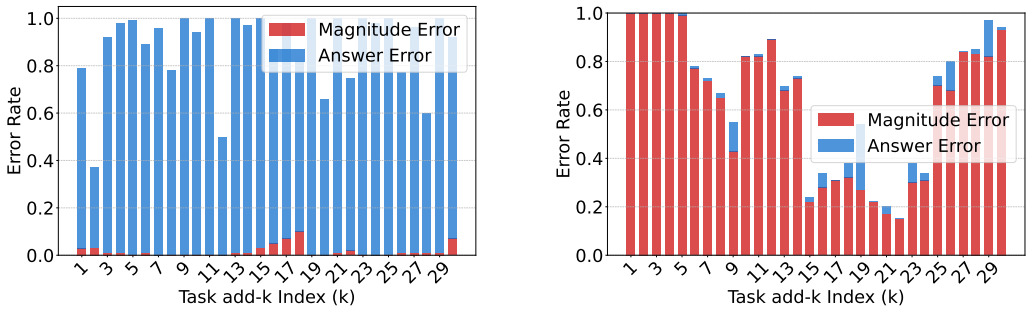
Figure 14: Validation of hypothesis (i) for head 2.



(a) Head 2 projected onto unit subspace

(b) Head 2 projected out of unit subspace

Figure 15: Validation of hypothesis (ii) for head 2.

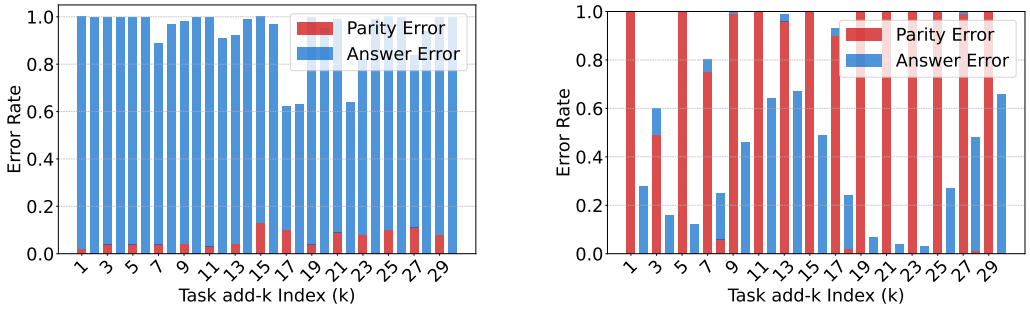


(a) Head 2 projected onto magnitude subspace

(b) Head 2 projected out of magnitude subspace

Figure 16: Validation of hypothesis (iii) for head 2.

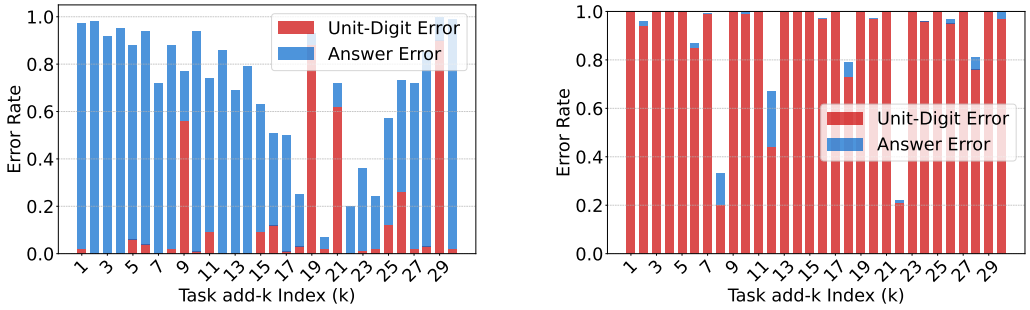
Finally, we show analogous results for head 3. The evidence for head 3 is slightly weaker than the first two heads, which aligns with the fact that the intervened accuracy of head 3 is slightly weaker than the ones of the first two heads (Tabel 1).



(a) Head 3 projected onto parity direction

(b) Head 3 projected out of parity direction

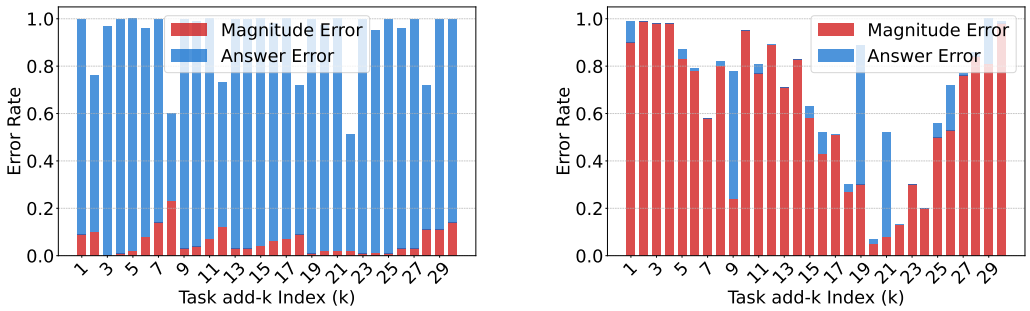
Figure 17: Validation of hypothesis (i) for head 3.



(a) Head 3 projected onto unit subspace

(b) Head 3 projected out of unit subspace

Figure 18: Validation of hypothesis (ii) for head 3.



(a) Head 3 projected onto magnitude subspace

(b) Head 3 projected out of magnitude subspace

Figure 19: Validation of hypothesis (iii) for head 3.