Increasing Trust in Language Models through the Reuse of Verified Circuits

Anonymous ACL submission

Abstract

Language Models (LMs) are increasingly used for a wide range of prediction tasks, but their training can often neglect rare edge cases, reducing their reliability. Here, we define a stringent standard of trustworthiness whereby the task algorithm and circuit implementation must be verified, accounting for edge cases, with no known failure modes. We show that a model can be trained to meet this standard if built using mathematically and logically specified frameworks. In this paper, we fully verify an auto-regressive transformer model that performs n-digit integer addition. To exhibit the reusability of verified modules, we insert the trained integer addition model into a larger untrained model and train the combined model to perform both addition and subtraction. We find extensive reuse of the addition circuits for both tasks, easing verification of the more complex subtractor model. We discuss how inserting verified task modules into LMs can leverage model reuse to improve verifiability and trustworthiness of LMs built using them. The reuse of verified circuits reduces the effort to verify more complex composite models which we believe to be a significant step towards *safety* and *interpretability* of LMs.

1 Introduction

Transformer-based large language models (LLMs) are powerful (Barak et al., 2022) yet largely inscrutable due to their complex, nonlinear interactions in dense layers within high-dimensional spaces. Given this complexity, their deployment in critical settings (Zhang et al., 2022) highlights the need for understanding their behavior. Hendrycks and Mazeika (2022) argue that making these models interpretable is key to their safe use. Mechanistic interpretability focuses on demystifying and validating the algorithms behind model weights, translating complex computations into more human-understandable components (Raukur et al., 2022).

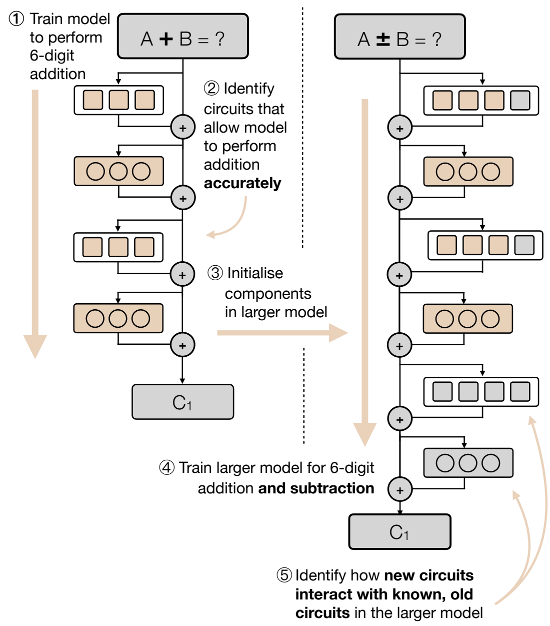

Figure 1: An overview of our methodology: (1) We trained an accurate 6-digit integer addition model. (2) We reverse-engineered the model to find the algorithms that were implemented to perform addition. (3) We inserted the addition model into a new model, by copying the weights of the attention heads and MLPs (in brown) into the larger model during initialization. (4) We then train the new model on 80% subtraction and 20% addition questions. (5) We find that the resulting model predicts accurately and reuses the inserted addition circuits for both addition and subtraction questions.

This understanding aids in predicting model behavior in new situations and fixing model errors.

In creating and training a model, we aim for high accuracy and trustworthiness. We achieve this by holding the model to a standard we term *known-good*. We define a model performing a task to be *known-good* if:

1. The model's algorithm for the task and the mechanisms it implements (the "circuits") are understood.

2. All possible task edge cases have been identified and tested.

3. Empirically, the model prediction accuracy is 99.9999% (a standard reliability measure used in industry and abbreviated as "six nines"). That is, it can perform the task one million times with at most 1 wrong prediction.

Exhaustive testing of a model task may be infeasible. For instance, when adding two 5-digit integers (e.g. 12345+67890) there are ten billion variations. Some tasks can be conceptualized within an existing formal framework that allows identification of all edge cases the model must handle. For example, in 5-digit addition, the most uncommon edge case is 55555+44445=100000, which requires a carry bit to cascade through all digits, occurring in only 0.002% of cases. A known-good model must incorporate algorithms to manage all known edge cases. A known-good model must have verified[1] circuits that perform the task accurately.

In this paper, we detail the development and interpretation of a known-good model for addition. Our findings indicate that the model constructs a specific circuit for each edge case, with these circuits sharing intermediate results. We confirm the validity of the entire set of circuits, ensuring they cover all identified edge cases. The model achieves a very low training loss and has six nines (99.9999%) accuracy. The model hence achieves our *known-good* standard. Additionally, we develop a "mixed" model capable of both addition and subtraction, incorporating the known-good addition model. This mixed model has six nines accuracy, and extensively reuses the addition circuits for both operations, facilitating the interpretation of the model's algorithm. We make progress toward a known-good model for both addition and subtraction.

Hence, our main contributions are three-fold:

- Defining several known-good n-digit addition models with six nines accuracy which all use the same algorithm.

- Demonstrating a proof of concept for re-using a known good model in the training of another larger, more-capable model, simplifying the interpretation of the new model's algorithm.

- Defining several n-digit addition and subtraction models with six nines accuracy, that reuse established addition model circuits for both operations, and detailing progress towards these models being known-good.

[1] In this paper, 'verified' has the mechanistic interpretability meaning that a specific group of interconnected neurons within a neural network reliably and causally contributes to the model's output in a meaningful, understandable way, with supporting empirical evidence.

2 Related Work

Mechanistic interpretability aims to reverse engineer neural networks to find interpretable algorithms that are implemented in a model's weights (Olah et al., 2020). Mathematical frameworks (Elhage et al., 2021a) explain how transformer attention heads can work with each other to implement complex algorithms.

Causal Scrubbing (Jenner et al., 2023) recommends explaining a model algorithm by documenting a low-level computation graph, mapping from the graph to the model nodes that implement the computation, and performing experimentation verification. Investigative techniques such as ablation interventions, activation unembeddings (nostalgebraist, 2020), and sparse autoencoders (Nanda, 2023; Cunningham et al., 2023), underpinned by the more theoretical frameworks (Elhage et al., 2021b; Geva et al., 2022), provide tools to help confirm a mapping.

Investigating pre-trained LMs on Arithmetic. Even though basic arithmetic can be solved following a few simple rules, pre-trained LMs often struggle to solve simple math questions (Hendrycks et al., 2021). Causal mediation analysis (Stolfo et al., 2023) has been used to investigate how large pre-trained LMs like Pythia and GPT-J performed addition to solve word problems. It is also possible to improve a model's arithmetic abilities used supervised fine tuning - including enriched training data (Liu and Low, 2023).

Studying Toy Models for Arithmetic. Doing mechanistic interpretability on toy transformers can help to better isolate clear, distinct circuits given the highly specific experimental setup for the model studied (Nanda et al., 2023). Quirke and Barez (2024) detailed a 1-layer, 3-head transformer model that performs 5-digit addition, showing it failed on rare edge cases (e.g. "77778+22222=100000" where a "carry 1" cascades through 4 digits), highlighting the importance of understanding and testing all edge cases for trustworthiness.

Many natural prediction problems decompose into a finite set of knowledge and skills that are "quantized" into discrete chunks (quanta) (Michaud et al., 2023). Models must learn these quanta to reduce loss. Understanding a network reduces to enumerating its quanta. Other studies (Schaeffer et al., 2023) prove useful ways to measure quanta in mathematical prediction problems.

3 Methodology

Transformer models may learn addition algorithms different from traditional human methods. We define an alternative, mathematically-equivalent framework for addition and demonstrate our model implements this approach.

3.1 Mathematical Framework

Consider the task of adding two n-digit numbers together. We define the first number as $D = \{D_{n-1}, D_{n-2}, \ldots, D_0\}$ and the second number as $D' = \{D'_{n-1}, D'_{n-2}, \ldots, D'_0\}$ and the answer as $A = \{A_n, A_{n-1}, \ldots, A_0\}$. Figure 2 shows an illustrative example.

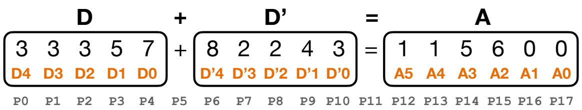

Figure 2: For 5-digit addition, our model has 12 input (question) and 6 output (answer) token positions. We name the question tokens D4, ..., D0, and D'4, ..., D'0 and the answer tokens A5, ..., A0. For n-digits, we use the terms D_n, D'_n and A_n.

First, we adopt the framework from Quirke and Barez (2024) for our model's addition process. The "Simple Addition" sub-task $A_n.SA$, which naively calculates the sum of digit pairs, is defined as:

$$A_n.SA = (D_n + D'_n) \mod 10 \tag{1}$$

When there is no carry bit from the previous digit $A_n = A_n.SA$. The "Simple Carry" sub-task $A_n.SC$ determines whether the addition creates a carry bit:

$$A_n.SC = \begin{cases} 1 & \text{if } (D_n + D'_n) \geq 10, \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

While Quirke et al.'s model is capable of handling simple addition and carry bits generated directly from digit pair addition, it encounters difficulties with 'cascading carry' bits, where a carry bit from one digit position propagates to the next.

Consider "00144+00056=000210". Adding 4+5 in the tens position doesn't generate a carry bit directly, but a carry bit propagates from the ones position. A model only summing digit pairs and their direct carry bits would fail, producing an incorrect result like "00144+50006=000110". The Quirke et al. 1-layer model could cascade carry bits across two digits, but not three or more.

3.2 Extending the Mathematical Framework

To answer "44444+55556=" with "100000", an accurate model must predict the first answer digit A5 as "1". To do so, an accurate model must implement a "carry one cascade" circuit, which combines the "carry one" information from all five digits. This is especially hard as the model predicts answer tokens from left to right.

We introduce a digit-level sub-task called TriCase that calculates the essential "carry one" information for a single pair of digits D_n and D'_n. TriCase has 3 possible outputs representing a **definite** carry one (*ST10*), a **possible** carry one depending on the results of other calculations (*ST9*), or definitely **not** a carry one (*ST8*):

$$A_n.ST = \underbrace{\text{TriCase}}_{(D_n, D'_n)} = \begin{cases} ST10 & \text{if } (D_n + D'_n) \geq 10, \\ ST9 & \text{if } (D_n + D'_n) = 9, \\ ST8 & \text{if } (D_n + D'_n) \leq 8, \end{cases} \tag{3}$$

To perform the "cascading carry one" calculation, we introduce a TriAdd sub-task. It handles the case where a **possible** carry one becomes a **definite** carry one because the next lower digit pair generated a carry one. TriAdd is defined as:

$$A_n.SV = \underbrace{\text{TriAdd}}_{(A_n.ST, A_{n-1}.ST)} = \begin{cases} ST10 & \text{if } A_n.ST = ST10 \text{ or} \\ & \quad (A_n.ST = ST9 \text{ and} \\ & \quad A_{n-1}.ST = ST10), \\ ST8 & \text{otherwise.} \end{cases} \tag{4}$$

The model can use ST and SV to accurately calculate A5 as 1 or 0 by using *A5 = A4.SV = TriAdd(A4.ST, TriAdd(A3.ST, TriAdd(A2.ST, TriAdd(A1.ST, A0.ST))))*. Note that in calculating A5, the model has also calculated an accurate carry bit for **each** answer digit. For example, the carry bit for A4 is *A3.SV = TriAdd(A3.ST, TriAdd(A2.ST, TriAdd(A1.ST, A0.ST)))*.

With this framework, the model only needs the sub-tasks $A_n.SA$, $A_n.ST$ and $A_n.SV$ to accurately perform addition (but some models also use the redundant $A_n.SC$ sub-task). This framework, if implemented by a model, is sufficient for the model to perform n-digit addition accurately.

Figure 3 diagrams how our model's algorithm uses SA, SC, ST and SV to perform addition. This algorithm is like the 99% accurate Quirke et al. algorithm but contains an additional circuit (the shaded boxes) to calculate "cascading carry one" data to predict with 99.9999% accuracy.

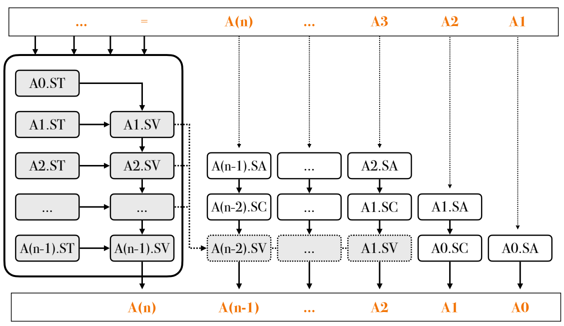

Figure 3: To predict with 99.9999% accuracy, the addition algorithm first calculates "carry one" values $(A_n.ST)$, combining them into "cascading carry one" values $(A_n.SV)$. At the "+" token, $A_{n-1}.SV$ gives the first answer digit as 1 or 0. The other answer digits are calculated by combining "base add" $(A_n.BA)$ and "carry one" $(A_n.SC)$ calculations with the pre-calculated $A_n.SV$ values.

3.3 Techniques

In investigating the model circuits, we want to understand what each attention head or MLP layer is doing across each token position, and how it relates to our mathematical framework. Hence, we define a *node* as the computation done by an attention head or MLP layer for a given token position. To investigate what each node is doing, we use the following techniques:

1. **Intervention Ablation.** To find out how the model depends on the output of a node, we replace the output of that node with the vector that is the mean of all of its outputs across a batch and measure how that impacts downstream performance. We also use (automated, n-digit) intervention ablation tests, specific to each sub-task, to test for the expected sub-task behavior.

2. **Attention Patterns.** To find out what the model attends to at a node, we take the attention pattern at that token position and take the significant tokens attended to (> 0.01 post softmax).

3. **Principal Component Analysis (PCA).** We use PCA to investigate the outputs of attention heads, especially where our framework suggests the head output may be tri-state or bi-state.

4. **Question Complexity.** We categorized questions by computational complexity (App. G). Addition categories (S0-S4) and subtraction categories (M0-M4) reflect the number of sequential digits a "carry one" or "borrow one" cascades through, respectively. We analyzed which nodes were necessary for correct predictions in each category.

4 Experiments

4.1 Training a Five-Digit Addition Model

The Quirke et al. 5-digit 1-layer addition model achieved an accuracy of $\sim$ 99%. Our experiments suggested that a 2-layer, 3-head model was the smallest configuration capable of achieving 99.9999% accuracy (see App. D for alternatives tested). This configuration effectively doubled the computational power compared to the 1-layer model (see App. B for mode model configuration details). Moreover, a 2-layer model introduces the capability to "compose" the attention heads in novel ways, facilitating the implementation of more complex algorithms (Elhage et al., 2021b).

We trained a 5-digit, 2-layer, 3-head model, with a 14 token vocabulary (0, .., 9, +, -, =, *, /), batch size of 64, learning rate of 0.00008 and weight decay of 0.1. Training used an infinite dataset enriched with rare edge cases. Loss was defined as the mean across all answer tokens of their negative log likelihood loss. After 30 thousand training batches, the model's final training loss was $\sim 2.3 \times 10^{-8}$. Testing showed this model has six nines accuracy. (More details in App. C and Tab. 6).

4.2 Investigating Five-Digit Addition

Ablation experiments targeting the nodes revealed that the model depends only on nodes located in nine token positions (Figs 6 and 7). Further ablation experiments show that for these nine token positions, the model uses 36 nodes in predictions. The effects of node ablation on our complexity and answer-impact metrics were analyzed (see Figs 5

<table>
<tr><th></th><th>(P6)
D'4</th><th>(P9)
D'1</th><th>(P10)
D'0</th><th>(P11)
=</th><th>(P12)
+</th><th>(P13)
A5</th><th>(P14)
A4</th><th>(P15)
A3</th><th>(P16)
A2</th><th>(P17)
A1</th></tr>
<tr><td>L0H0</td><td></td><td></td><td rowspan="3">A5..3</td><td></td><td></td><td rowspan="2">A4</td><td></td><td rowspan="2">A2</td><td rowspan="2">A1</td><td></td></tr>
<tr><td>L0H1</td><td rowspan="3">A5</td><td>A5</td><td></td><td rowspan="6">A5</td><td>A3</td><td>A0</td></tr>
<tr><td>L0H2</td><td></td><td rowspan="2">A5..1</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>L0MLP</td><td></td><td>A5..2</td><td>A4</td><td>A3</td><td>A2</td><td>A1</td><td>A0</td></tr>
<tr><td>L1H0</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>L1H2</td><td></td><td></td><td></td><td></td><td rowspan="2">A4</td><td rowspan="2">A3</td><td rowspan="2">A2</td><td></td><td></td></tr>
<tr><td>L1MLP</td><td></td><td></td><td></td><td></td><td>A1</td><td>A0</td></tr>
</table>

Table 1: For a sample model, all nodes used in predictions are shown by token position (horizontally) and model layer (vertically), detailing the **answer digits** they impact. Here, the attention heads in token position P10 labelled A5..3 help predict the answer digits A3, A4 and A5. For all addition and mixed models studied, before the "=" token, each node often calculates data used to predict **multiple** answer digits. After the "=" token, all nodes in a given token position are used to predict a **single** answer digit.

and Table 1), providing insight into the specific computations performed at each node. For each answer digit A_n, using test questions corresponding to the ST8, ST9 and ST10 categories, we performed PCA on the nodes yielding interpretable results. Specifically nine "node and answer-digit" combinations (see Figs 8 and 9) showed strong clustering of the questions aligned to the ST8, ST9 and ST10 categories.

The algorithm predicts the first answer digit, A5, at position P11. A5, which is always 0 or 1, is the most challenging to predict as it may rely on a long carry one cascade (e.g. 55555+44445=100000). An accurate algorithm must compute this cascade using the nodes located in positions P8 to P11. As illustrated in Figure 7, these nodes attend to all digit pairs from D4 D'4 to D0 D'0. Additionally, the PCA data, as shown in Figures 8 and 9, suggest that these nodes produce tri-state outputs. After our first two algorithm hypothesises failed testing (see App. H and I), we discovered that the model utilizes a minimal set of "carry one" information, leading to the development of the TriCase quanta. The model performs $A_n.ST$ using bigrams (see App. F) to map two input tokens to one result token e.g. "6" + "7" = ST10. In positions P8 to P11, the model does $A_n.ST$ calculations on all digit pairs from D4 D'4 to D0 D'0.

An MLP layer can be thought of as a "key-value pair" memory (Meng et al., 2022; Geva et al., 2021) that can hold many bigrams and trigrams. We posit our MLP implements the TriAdd function using bigrams and trigrams to calculate $A_n.SV$ values from $A_n.ST$ values.

For a specific 5-digit addition model instance, we mapped the algorithm to individual nodes and verified each node's role using ablation intervention. Figure 3 diagrams the algorithm, with node details in App. J. The model adheres to all known constraints and achieves six nines accuracy. We concluded this model instance is well-understood, well-functioning, and hence known-good.

The 1-layer model uses 21 nodes to achieve two nines (99%) accuracy. This model uses 36 nodes (an increase of 71%) to achieve six nines accuracy.

4.3 Training n-digit Addition models

To investigate whether this algorithm is used widely, we first trained seven 2-layer addition models with 5-, 6- and 10-digits, using different seeds, a different optimizer, and changing the answer format to include a sign token (e.g. 111111+222222=+0333333).

These seven models all have very low loss (e.g. 1.5e-8) and six nines accuracy. (Details in Tab.6.)

4.4 Investigating n-digit Addition models

We developed a declarative method to outline each sub-task. For instance, a sub-task may focus on question digits D2 and D'2, affect answer digit A3, influence S0 but not S1 complexity questions, and have specific PCA results and ablation tests. Using this declaration, we identified nodes performing these sub-tasks across all models.

An algorithm hypothesis, such as the one in section 4.2, is described by the required sub-tasks and their relationships. For example, our addition algorithm specifies that the model must execute $A_n.ST$

	(P11) D'1	(P12) D'0	(P13) =	(P14) +	(P15) A6	(P16) A5	(P17) A4	(P18) A3	(P19) A2	(P20) A1
L0H0	A2.ST	A3.ST	A1.ST	A4.ST	A4.SC	A3.SC	A2.SC	A1.SC	A0.SC	
L0H1	A1.ST		A0.ST		A5.SA	A4.SA	A3.SA	A2.SA	A1.SA	A0.SA
L0H2				A5.ST						

Table 2: All addition models studied implement our addition algorithm. The algorithm *SA*, *SC* and *ST* sub-tasks all exist for each digit and in appropriate token positions. For a sample model, this map shows the subtask locations. Interestingly, here each *SA* sub-task is shared across two attention heads.

sub-tasks for each question digit before the “=” token. We created a framework for declaring n-digit algorithm hypotheses and testing them against the sub-tasks found in each model, mapping the results (see Table 2 for an example).

Our seven 2-layer addition models all implement our addition algorithm. Given their six nines accuracy and implementation of the same algorithm, we can confirm these models as known-good.

4.5 Training n-digit Mixed models

To explore reuse, we initialized untrained models with a known-good addition-only model, then trained them to perform **both** subtraction and addition. We call these “mixed" models.

Specifically, we trained seven larger (6- or 10-digit, 2- or 3-layer, 3- or 4-head) models after initializing them with the weights from a known-good 2-layer 3-head addition model. The first 2 layers, first 3 heads of the mixed model were initialized with the addition model weights. We trained the mixed model with 80% subtraction and 20% addition batches. We enriched the (infinite) training dataset with rare addition and subtraction edge cases.

Some models achieved six nines accuracy and the others five nines. Attempts to “freeze" the inserted attention heads and/or MLP layers by periodically copying the addition weights back into the mixed model every 100 training steps resulted in lower accuracy. (Refer App.M and Tab.6)

4.6 Investigating n-digit Mixed models

Unlike addition, subtraction question answers can be either positive or negative. Similar looking positive-answer (e.g. 10009-10000=+000009) and negative-answer (e.g. 10009-20000=-009991) subtraction questions can give answers that differ at several digit positions. We posited that the model treats three distinct question classes “addition", “positive-answer subtraction" and “negative-answer

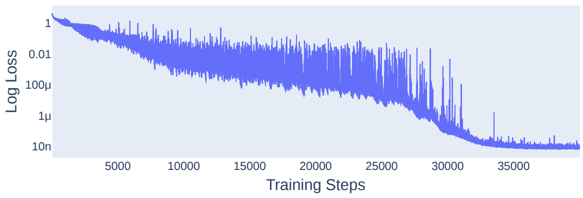

Figure 4: Each mixed model was initialized with the weights from a known-good addition model, then was trained on 80% subtraction and 20% addition batches. A sample log loss graph (final loss 8.0e-9) is shown.

subtraction" differently. Ablation of nodes showed that some nodes only help predict one question class, some help predict two classes and some all three classes. It also showed that the inserted addition nodes are heavily used and the majority become polysemantic, performing both addition and subtraction calculations. (Details in Tab.3.)

	Used		Inserted	
Question class	#	%	#	%
All questions	96		48	
Addition	61	64%	42	88%
Positive-answer sub	70	73%	40	83%
Negative-answer sub	53	55%	29	60%

Table 3: Mixed models re-use most inserted addition-model nodes. Many inserted nodes become polysemantic during training - performing addition, positive-answer subtraction **and** negative-answer subtraction sub-tasks simultaneously. For a sample mixed model that uses 96 nodes and had 48 nodes inserted, this table shows inserted node reuse.

To investigate the subtraction algorithm, and paralleling addition algorithm subtasks, we defined subtraction-specific sub-tasks Base Diff and Borrow One (see Tab. 4) and complexity measures (see App. G and N).

We found that the inserted nodes that performed *SA* in the addition model, perform *SA*, *MD* **and** *ND* in the mixed model. (Refer Table 5.) These three

Name	+ve Sub	-ve Sub	Definition
Base Diff	*MD*	*ND*	D_n- D'_n% 10
Borrow One	*MB*	*NB*	D_n- D'_n< 0

Table 4: We define 2 “positive-answer subtraction" and 2 “negative-answer subtraction" sub-tasks that parallel the addition sub-tasks Base Add *SA* and Carry One *SC*

sub-tasks are similar in that each performs a mapping from 100 input cases (10 D_nx 10 D'_n) to 10 output cases (0..9). The mixed model “upgraded" these nodes to be polysemantic during training. Similarly, some *SC* addition nodes became polysemantic and now process *SC, MB* and/or *NB* .

An accurate subtraction model must answer “cascading borrow one" questions like 100000-000001=+0099999. We define the “essence of borrow one" subtask MT used by both positive-answer (M) and negative answer (N) subtraction questions. MT is like addition’s ST subtask:

$$A_n.MT = \underbrace{\text{TriCase}}_{(D_n, D'_n)} = \begin{cases} MTN & \text{if } D_n < D'_n \\ MT0 & \text{if } D_n = D'_n \\ MT1 & \text{if } D_n > D'_n \end{cases} \tag{5}$$

For M questions, MTN is a **definite** borrow one, MT0 is a **possible** borrow one (depending on the results of other digit calculations) and MT1 is definitely **not** a borrow one. For N questions, the interpretation is the opposite.

Paralleling addition’s “cascading carry one" $A_n.SV$ calculation, we define “cascading borrow one" calculation sub-tasks $A_n.MV$ for positive-answer subtraction and $A_n.NV$ for negative-answer subtraction. We posit the MLP implements TriAdd-like functions using bigrams and trigrams to calculate $A_n.MV$ and $A_n.NV$ values from $A_n.MT$ values.

4.7 Mixed model question class detection

We posit that there is a specific circuit to detect whether a question is in the S, M or N class. If the question operator is “+" then the class is S, but if the question operator is “-" then the model must calculate if $D >= D'$ to distinguish between the M and N classes. For accuracy, it needs this answer by the “=" token to predict the first answer token (the answer sign) as “+" or “-".

$D >= D'$ can be derived from the $A_n.MT$ data. Alternatively, this calculation could be a distinct circuit using a new sub-task we define as:

$$A_n.GT = \begin{cases} 1 & \text{if } D_n \geq D'_n, \\ 0 & \text{otherwise.} \end{cases} \tag{6}$$

For a 4 digit question, $D >= D'$ can be calculated as *A3.GT=1 or (A3.GT=0 and (A2.GT=1 or (A2.GT=0 and (A1.GT=1 or (A1.GT=0 and A0.GT=1)))*. Our test for *GT* is that ablation causes the answer to change sign. The MT calculation is very similar.

We found $A_n.MT$ and $A_n.GT$ subtasks in all mixed models. Usually (but not always) both sub-tasks are calculated by the same node. (Table 2 shows an example.) As both the MT and *GT* approaches are valid, models can learn valid algorithm sub-task implementations that differ per answer digit.

We further posit that a node that implements say *A2.SA, A2.MD* and *A2.ND* does not know whether it is dealing with a S, M or N question, so it outputs all three possible answers to the residual stream. Another node must calculate the S, M or N distinction likely by attending to the question operator (OPR) and the answer sign (SGN). We define attention sub-tasks OPR and SGN for these calculations. Table 2 shows that for each answer digit one attention head attends to both OPR and SGN. We believe these heads transfer sufficient data to the residual stream to allow the MLP layer(s) to calculate the question class, and so select the appropriate output from the polysemantic *A2.SA/MD/ND* node.

4.8 Mixed model summary

The 7 mixed experimental models achieve five or six nines accuracy and contain the same sub-tasks, implying a common algorithm. However, without fully understanding this algorithm, we can’t confirm these models as known-good.

5 Conclusion

We successfully trained and verified known-good 5-, 6- and 10-digit, 2-layer, 3-head addition models implementing the same algorithm with minor variations.

We demonstrated component reuse by integrating an existing addition model into a larger “mixed" model for both addition and subtraction, achieving six nines accuracy. This integration helped us understand the mixed model’s algorithm. The mixed

<table>
<tr><th></th><th>(P9)
D'3</th><th>(P10)
D'2</th><th>(P11)
D'1</th><th>(P12)
D'0</th><th>(P13)
=</th><th>(P14)
A7</th><th>(P15)
A6</th><th>(P16)
A5</th><th>(P17)
A4</th><th>(P18)
A3</th><th>(P19)
A2</th><th>(P20)
A1</th></tr>
<tr><td>L0H0</td><td>A4.MT
A4.GT</td><td></td><td></td><td>A3.MT
A3.MT</td><td></td><td>A4.ST
A4.MT
OPR</td><td>A4.SC</td><td>A3.SC</td><td>A2.SC
A2.NB</td><td>A1.SC
A1.MB
A1.NB</td><td>A0.SC
A0.MB
A0.NB</td><td>OPR
SGN</td></tr>
<tr><td>L0H1</td><td>A4.ST</td><td>A2.ST
A2.MT
A2.GT</td><td>A1.ST
A1.MT
A1.GT</td><td>A3.ST
A3.GT</td><td>A0.ST
A0.MT
A0.GT</td><td></td><td>A5.SA
A5.MD</td><td rowspan="2">A4.SA
A4.MD
A4.ND</td><td rowspan="2">A3.SA
A3.MD
A3.ND</td><td rowspan="2">A2.SA
A2.MD
A2.ND</td><td rowspan="2">A1.SA
A1.MD
A1.ND</td><td rowspan="2">A0.SA
A0.MD
A0.ND</td></tr>
<tr><td>L0H2</td><td></td><td></td><td></td><td></td><td></td><td>A5.ST
OPR
SGN</td><td>A5.SA
A5.MD
A5.ND</td></tr>
<tr><td>L0H3</td><td></td><td></td><td></td><td></td><td></td><td></td><td>OPR
SGN</td><td>OPR
SGN</td><td>OPR
SGN</td><td>OPR
SGN</td><td>OPR
SGN</td><td>OPR
SGN</td></tr>
</table>

Table 5: For mixed models, in later tokens, polysemantic attention heads simultaneously generate outputs for the three question classes addition S, M and N. Other heads calculate the question class by attending to the question operation (OPR) token and the answer sign (SGN) token. The MLP layers then select the output appropriate for the class. In this sample map, from P16, the first 3 rows contain many polysemantic nodes, while the 4th row calculates the question class.

model reuses most inserted addition nodes, upgrading many to become polysemantic - performing both addition and subtraction subtasks simultaneously.

Our work supports Michaud et al. (2023)'s assertion that many prediction problems can be broken into finite "quanta" computations essential for loss minimization. It also aligns with the idea that understanding a network's functionality involves identifying and comprehending its sub-quanta.

5.1 Future Work

In the future, we aim to develop a comprehensive, known-good model for n-digit addition, subtraction, and multiplication. Our approach could also be applied to create known-good models in logical reasoning and planning.

Further exploration of using known-good models to improve LLMs is a promising direction for enhancing LLM trustworthiness and capabilities. This aligns with current research in the field, including model composition (Bansal et al., 2024), LM up-scaling which emulates fine-tuning a large model using a small model (Mitchell et al., 2023), and inserting accurate models into untrained ones (as demonstrated in this paper). Additionally, research on "spare" neurons in LLMs (Voita et al., 2023; Hu et al., 2021) suggests potential for small-scale modifications to fix erroneous circuits, further supporting this approach.

Developing methods to incorporate compact known-good models into LLMs could democratize AI Safety research, allowing small teams to focus on specific areas and create quality components to improve LLMs.

6 Limitations

While we identify and test the *role* of each node in the mixed model algorithm, we do not detail the *data representation* of the polysemantic nodes, SGN nodes, and OPR nodes output in the residual stream. This limitation means we can not detail the *transformation* of this data performed by MLP layer nodes that support accurate answer digit prediction.

Our automated framework for discovering algorithm sub-tasks in models, while instrumental in accelerating our research, has limitations. Some aspects are specific to our math models and may not be directly applicable to other domains.

Furthermore, while we have made progress on a declarative language to describe algorithms in terms of necessary sub-tasks and a framework to test these descriptions against specific models, this work is still in its early stages.

7 Impact Statement

Our work aims to explain the inner workings of transformer-based language models, which may have broad implications for a wide range of applications. A deeper understanding of generative AI has dual usage. While the potential for misuse exists, we discourage it. The knowledge gained can be harnessed to safeguard systems, ensuring they operate as intended. It is our sincere hope that this research will be directed towards the greater good, enriching our society and preventing detrimental effects. We encourage responsible use of AI, aligning with ethical guidelines.

References

Rachit Bansal, Bidisha Samanta, Siddharth Dalmia, Nitish Gupta, Shikhar Vashishth, Sriram Ganapathy, Abhishek Bapna, Prateek Jain, and Partha Talukdar. 2024. Llm augmented llms: Expanding capabilities through composition. *Preprint*, arXiv:2401.02412.

Boaz Barak, Benjamin L. Edelman, Surbhi Goel, Sham M. Kakade, Eran Malach, and Cyril Zhang. 2022. Hidden progress in deep learning: Sgd learns parities near the computational limit. *ArXiv*, abs/2207.08799.

Hoagy Cunningham, Aidan Ewart, Logan Riggs, Robert Huben, and Lee Sharkey. 2023. Sparse autoencoders find highly interpretable features in language models. *Preprint*, arXiv:2309.08600.

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, Dario Amodei, Tom Brown, Jack Clark, Jared Kaplan, Sam McCandlish, and Chris Olah. 2021a. A mathematical framework for transformer circuits. *Transformer Circuits Thread*. Https://transformer-circuits.pub/2021/framework/index.html.

Nelson Elhage, Neel Nanda, Catherine Olsson, et al. 2021b. A mathematical framework for transformer circuits. https://transformer-circuits.pub/2021/framework/index.html.

Mor Geva, Avi Caciularu, Kevin Ro Wang, and Yoav Goldberg. 2022. Transformer feed-forward layers build predictions by promoting concepts in the vocabulary space. *Preprint*, arXiv:2203.14680.

Mor Geva, Roei Schuster, Jonathan Berant, and Omer Levy. 2021. Transformer feed-forward layers are key-value memories. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 5484–5495, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the math dataset. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*.

Dan Hendrycks and Mantas Mazeika. 2022. X-risk analysis for ai research. *ArXiv*, abs/2206.05862.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *Preprint*, arXiv:2106.09685.

Erik Jenner, Adrià Garriga-alonso, and Egor Zverev. 2023. A comparison of causal scrubbing, causal abstractions, and related methods. https://www.lesswrong.com/posts/uLMWMeBG3ruoBRhMW/a-comparison-of-causal-scrubbing-causal-abstractions-and.

Tiedong Liu and Bryan Kian Hsiang Low. 2023. Goat: Fine-tuned llama outperforms gpt-4 on arithmetic tasks. *Preprint*, arXiv:2305.14201.

Kevin Meng, David Bau, Alex Andonian, and Yonatan Belinkov. 2022. Locating and editing factual associations in gpt. https://proceedings.neurips.cc/paper_files/paper/2022/file/6f1d43d5a82a37e89b0665b33bf3a182-Paper-Conference.pdf.

Eric J. Michaud, Ziming Liu, Uzay Girit, and Max Tegmark. 2023. The quantization model of neural scaling. *Preprint*, arXiv:2303.13506.

Eric Mitchell, Rafael Rafailov, Archit Sharma, Chelsea Finn, and Christopher D. Manning. 2023. An emulator for fine-tuning large language models using small language models. *Preprint*, arXiv:2310.12962.

Neel Nanda. 2023. One layer sparce autoencoder. https://github.com/neelnanda-io/1L-Sparse-Autoencoder.

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

nostalgebraist. 2020. interpreting gpt: the logit lens. https://www.alignmentforum.org/posts/AcKRB8wDpdaN6v6ru/interpreting-gpt-the-logit-lens.

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

Philip Quirke and Fazl Barez. 2024. Understanding addition in transformers. In *The Twelfth International Conference on Learning Representations*.

Tilman Raukur, An Chang Ho, Stephen Casper, and Dylan Hadfield-Menell. 2022. Toward transparent ai: A survey on interpreting the inner structures of deep neural networks. *2023 IEEE Conference on Secure and Trustworthy Machine Learning (SaTML)*, pages 464–483.

Rylan Schaeffer, Brando Miranda, and Sanmi Koyejo. 2023. Are emergent abilities of large language models a mirage? *Preprint*, arXiv:2304.15004.

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.

Elena Voita, Javier Ferrando, and Christoforos Nalmpantis. 2023. Neurons in large language models: Dead, n-gram, positional. *Preprint*, arXiv:2309.04827.

Angela Zhang, Lei Xing, James Zou, and Joseph C. Wu. 2022. Shifting machine learning for healthcare from development to deployment and from models to data. *Nature Biomedical Engineering*, 6:1330 – 1345.

A Appendix: Terminology

These terms and abbreviations are used in this paper and the associated Colabs and python code:

- **Pn** : Model (input or output) token position. Zero-based. e.g. **P**18, **P**18L1H0
- **Ln** : Model layer n. Zero-based. e.g. P18**L**1H2
- **Hn** : Attention head n. Zero-based. e.g. P18L1**H**2
- **Mn** : MLP neuron n. Zero-based
- **PnLnHn** : Location / name of a single attention head, at a specified layer, at a specific token position
- **PnLnMn** : Location / name of a single MLP neuron, at a specified layer, at a specific token position
- **D** : First number of the pair question numbers
- **Dn** : nth numeric token in the first question number. Zero-based. D0 is the units value
- **D'** : Second number of the pair question numbers
- **D'n** : nth token in the second question number. Zero-based. D0 is the units value
- **A** : Answer to the question (including answer sign)
- **An** : nth token in the answer. Zero-based. A0 is the units value. The highest token is the "+" or "-" answer sign
- **S** : Prefix for Addition. Think S for Sum. Aka ADD.
- **SA** : Basic Add. An addition sub-task. An.SA is defined as (Dn + D'n) % 10. e.g. 5 + 7 gives 2
- **SC** : Carry One. An addition sub-task. An.SC is defined as Dn + D'n >= 10. e.g. 5 + 7 gives True
- **SS** : Make Sum 9. An addition sub-task. An.SS is defined as Dn + D'n == 9. e.g. 5 + 7 gives False
- **ST** : TriCase. An addition sub-task. Refer paper 2 for details
- **ST8, ST9, ST10** : Outputs of the ST TriCase sub-task.
- **M** : Prefix for Subtraction with a positive answer. Think M for Minus. Aka SUB
- : Basic Difference. A subtraction sub-task. An.MD is defined as (Dn - D'n) % 10. e.g. 3 - 7 gives 6
- : Borrow One. A positive-answer subtraction sub-task. An.MB is defined as Dn - D'n < 0. e.g. 5 - 7 gives True
- **MZ** : Make Zero. A positive-answer subtraction sub-task. An.MZ is defined as Dn - D'n == 0. e.g. 5 - 5 gives True
- **MT** : TriCase. A positive-answer subtraction sub-task.
- **MT1, MT0, MT-1** : Outputs of the MT TriCase sub-task.
- **N** : Prefix for Subtraction with a negative answer. Think N for Negative. Aka NEG
- **ND** : Basic Difference. A negative-answer subtraction sub-task. An.ND is defined as (Dn - D'n) % 10. e.g. 3 - 7 gives 6
- **NB** : Borrow One. A negative-answer subtraction sub-task. An.NB is defined as Dn - D'n < 0. e.g. 5 - 7 gives True
- **NZ** : Make Zero. A negative-answer subtraction sub-task. An.NZ is defined as Dn - D'n == 0. e.g. 5 - 5 gives True
- **NT** : TriCase. A negative-answer subtraction sub-task.

- **GT** : Greater Than. A (positive-answer or negative-answer) subtraction sub-task. An.GT is defined as Dn > D'n. e.g. 3 > 5 gives False
- **OPR** : Operator. A sub-task that attends to the + or - token in the question (which determines whether the question is addition or subtraction).
- **SGN** : Sign. A sub-task that attends to the first answer token, which is + or -
- **PCA** : Principal Component Analysis
- **EVR** : Explained Variance Ratio. In PCA, EVR represents the percentage of variance explained by each of the selected components.

B Appendix: Model Configuration

Addition, subtraction and mixed (addition and subtraction) training experiments were done in a Colab notebook. The Colab runs on a T4 GPU. Each training run takes up to 60 mins. The key parameters (and their common configurations) are:

- n_layers = 1, 2 or 3: Number of layers.
- n_heads = 3 or 4: Number of attention heads.
- n_digits = 5, 6 or 10: Number of digits in the question.

Each digit is represented as a separate token. (Liu and Low, 2023) state that LLaMa's "remarkable arithmetic ability ... is mainly atributed to LLaMA's consistent tokenization of numbers". The model's vocabulary contains 14 tokens (0, .., 9, +, -, =, *, /) to enable this and planned future investigations.

Training uses a new batch of data each step (aka Infinite Training Data) to minimise memorisation. Depending on the configuration, each training run processes 1 to 4 million training datums. For the 5-digit addition problem there are 100,000 squared (that is 10 billion) possible questions. So the training data is much less than 1% of the possible problems.

Addition and subtraction include rare edge cases. For example, the *SS* cascades (e.g. 44445+55555=100000, 54321+45679=1000000, 44450+55550=10000, 1234+8769=10003) are exceedingly rare. The data generator was enhanced to increase the frequency of all known edges cases. This lead to lower model loss.

Validation test data covering all edge cases was manually constructed. These test cases are not used during training.

The Colabs will be made available on publication.

C Appendix: Model Loss

The model defaults to batch size = 64, learning rate = 0.00008 and weight decay = 0.1. The loss function is simple:

- Per Digit Loss: For "per digit" graphs and analysis, for a given answer digit, the loss used is negative log likelihood.
- All Digits Loss: For "all answer digits" graphs and analysis, the loss used is the mean of the "per digit" loss across all the answer digits.

In our experimental models, the number of digits in the question varies from 5 to 10, the number of layers varies from 1 to 4, the number of heads varies from 3 to 4. Each experimental model's loss is detailed in Tab. 6.

D Appendix: Addition Model Shape

While we wanted a very low loss addition model, we also wanted to keep the model compact - intuiting that a smaller model would be easier to understand than a large model. Here are the things we tried to reduce loss that **didn't** work:

- Increasing the frequency of hard (cascading *SS*) examples in the training data so the model has more hard examples to learn from. This improved training speed but did not reduce loss.
- Increasing the number of attention heads from 3 to 4 or 5 (while still using 1 layer) to provide more computing power.
- Changing the question format from "12345+22222=" to "12345+22222equals" giving the model more prediction steps after the question is revealed before it needs to state the first answer digit.
- With n_layers = 1 increasing the number of attention heads from 3 to 4.
- Changing the n_layers to 2 and n_heads to 2.

The smallest model shape that did reduce loss significantly was 2 layers with 3 attention heads.

E Appendix: Experimental models

Twenty-one models were trained and analyzed (refer Tab. 6). The models and analysis output will be made available on HuggingFace on publication to support further research in AI Safety.

For each model the 'VerifiedArithmeticTrain' Colab notebook generates two files:

- A "XXXXXX.pth" file containing the model weights
- A "XXXXXX_train.json" file containing configuration information and training loss data

While, for each model the 'VerifiedArithmetic-Analysis' Colab notebook generates two more files:

- A "XXXXXX_behavior.json" file containing generic "behavior" facts learnt about the model by the Colab e.g. P18L0H0 attends to tokens D3 and D'3
- A "XXXXXX_maths.json" file containing "maths-specific" facts learnt about the model by the Colab e.g. P18L0H0 performs the A3.SC sub-task.

F Appendix: TriAdd Implementation

TriAdd transfers data from $A_{n-1}$to $A_n$by integrating the values of $A_{n-1}.ST$ and $A_n.ST$. This function can be represented as nine bigram mappings with three possible outputs. (Refer Tab.7.)

Note that in the case $A_n.ST$ = ST9 and $A_{n-1}.ST$ = ST10, the answer is indeterminate. The result could be ST8 or ST9 but importantly it can not be ST10. We choose to use ST8 in our definition, but ST9 would work just as well.

G Appendix: Complexity

To analyze question difficulty, we categorized addition questions by the complexity of the computation required to solve the question, as shown in Tab. 8. The categories are arranged according to the number of digits that a carry bit has to cascade through.

H Appendix: Addition Hypothesis 1

Given the 2-layer attention pattern's similarity to 1-layer attention pattern, and the above evidence, our first (incorrect) hypothesis was that the 2-layer algorithm:

- Is based on the same *SA, SC* and *SS* operations as the 1-layer.
- Uses the new early positions to (somehow) do the *SS* calculations with higher accuracy than the 1-layer model.
- The long double staircase still finalises each answer digit's calculation.
- The two attention nodes in the long double staircase positions do the *SA* and *SC* calculations and pull in *SS* information calculated in the early positions.

If this is correct then the 2-layer algorithm successfully completes these calculations:

- A0 = A0.SA
- A1 = A1.SA + A0.SC
- A2 = A2.SA + (A1.SC or (A1.SS & A0.SC))
- A3 = A3.SA + (A2.SC or (A2.SS & A1.SC) or (A2.SS & A1.SS & A0.SC))
- A4 = A4.SA + (A3.SC or (A3.SS & A2.SC) or (A3.SS & A2.SS & A1.SC) or (A3.SS & A2.SS & A1.SS & A0.SC))
- A5 = A4.SC or (A4.SS & A3.SC) or (A4.SS & A3.SS & A2.SC) or (A4.SS & A3.SS & A2.SS & A1.SC) or (A4.SS & A3.SS & A2.SS & A1.SS & A0.SC)

Our intuition is that there are not enough useful nodes in positions 8 to 11 to complete the A5 calculation this way. So we abandoned this hypothesis.

I Appendix: Addition Hypothesis 2

Our second (incorrect) hypothesis was that the 2-layer algorithm has a **more compact** data representation, so it can pack more calculations into each node, allowing it to accurately predict A5 in step 11.

We claimed the model stores the sum of each digit-pair as a single token in the range "0" to "18" (covering 0+0 to 9+9). We name this operator A_n.T, where T stands for "token addition":

- A_n.T = D_n+ D'_n

The A_n.T operation does not understand mathematical addition. Tab. 9 shows how the model implements the T operator as a bigram mapping.

Num Digits	Num Layers	Num Heads	Train Steps	Train Seed	Train loss	Addition Fails / M	Subtract Fails / M	Heads used	MLPs used
Addition models									
5	1	3	30K	372001	9.4e-2	12621	N/A	15	6
5	2	3	15K	372001	1.6e-8	0	N/A	30	16
5	2	3	40K	372001	2.0e-9	**0**	N/A	**22**	**15**
6	2	3	15K	372001	1.7e-8	2	N/A	31	17
6	2	3	20K	173289	1.5e-8	**0**	N/A	**28**	**17**
6	2	3	20K	572091	7.0e-9	0	N/A	35	17
6	2	3	40K	372001	2.0e-9	0	N/A	29	17
10	2	3	40K	572091	7.0e-9	**0**	N/A	**44**	**28**
Subtraction models									
6	2	3	30K	372001	5.8e-6	N/A	0	40	21
10	2	3	75K	173289	2.0e-3	N/A	6672	101	37
Mixed models									
6	3	4	40K	372001	5.0e-9	1	0	54	26
10	3	4	75K	173289	1.1e-6	2	295	143	53
Mixed models initialized with addition model									
6	2	3	40K	572091	2.4e-8	0	5	57	21
6	3	3	40K	572091	1.8e-8	0	3	70	35
6	3	3	80K	572091	1.6e-8	0	3	75	35
6	3	4	40K	372001	8.0e-9	**0**	**0**	**72**	**26**
6	3	4	40K	173289	1.4e-8	3	2	60	29
6	3	4	50K	572091	2.9e-8	0	4	79	29
10	3	3	50K	572091	6.3e-7	**6**	**7**	**90**	**45**
Mixed models initialized with add model. Reset useful heads every 100 steps									
6	4	4	40K	372001	1.7e-8	3	8	51	30
Mixed models initialized with add model. Reset useful heads & MLPs every 100 steps									
6	4	3	40K	372001	3.0e-4	17	3120	115	53

Table 6: Main experimental models studied. The number of addition and subtraction failures per million questions is shown. The best 5-, 6- and 10-digit models are bolded.

$A_{n-1}.ST$	$A_n.ST$ = ST8	$A_n.ST$ = ST9	$A_n.ST$ = ST10
ST8	ST8	ST9	ST10
ST9	ST8	ST9	ST10
ST10	ST8 *	ST10	ST10

Table 7: A_n.TriAdd can be calculated from $A_n.ST$ and $A_{n-1}.ST$ through nine bigram mappings and yielding the three distinct outputs ST8, ST9 and ST10

A_n.T is a compact way to store data. Tab. 10 show how, if it needs to, the model can convert a A_n.T value into a one-digit-accuracy *SA, SC* or *SS* value.)

Our notation shorthand for one-digit-accuracy these "conversion" bigram mappings is:

- A_n.*SA* = (A_n.T % 10) where % is the modulus operator
- A_n.*SC* = (A_n.T // 10) where // is the integer division operator
- A_n.SS= (A_n.T == 9) where == is the equality operator

The A0.T value is accurate. But the other A_n.T values are **not** accurate because each is constrained to information from just one digit. We define another more accurate operator A_n.T2 that has "two-digit accuracy". A_n.T2 is the pair sum for the nth digit plus the carry bit (if any) from the n-1th digit T:

- A_n.T2 = A_n.T + A_{n-1}.SC

A_n.T2 is more accurate than A_n.T. The A_n.T2 value is always in the range "0" to "19" (covering 0+0+0 to 9+9+CarryOne). Tab. 11 show how the model can implement the T2 operator as a mapping.

Name	Contains	Example	Freq
S0	*SA*	11111+12345=23456	∼5%
S1	SA,SC	11111+9=22230	∼21%
S2	SA,SCx2	11111+89=22300	∼34%
S3	SA,SCx3	11111+889=23000	∼28%
S4	SA,SCx4	11111+8889=30000	∼11%
S5	SA,SCx5	11111+88889=100000	∼2%

Table 8: We categorise addition questions into non-overlapping “calculation complexity" quanta, ordered by increased computational difficulty (and decreasing occurrence frequency). Five-digit addition questions quanta are *S0* to *S5*. Ten-digit addition question quanta are *S0* to *S10*. *S10*'s frequency is $\sim 3e-4$ showing the need to enrich training data for rare edge cases.

D_n vs D'_n	**0**	**1**	**...**	**4**	**5**	**...**	**8**	**9**
0	0	1	...	4	5	...	8	9
1	1	2	...	5	6	...	9	10
...	...	...	...	...	...	...	...	...
4	4	5	...	8	9	...	12	13
5	5	6	...	9	10	...	13	14
...	...	...	...	...	...	...	...	...
8	8	9	...	12	13	...	16	17
9	9	10	...	13	14	...	17	18

Table 9: Implementing the T operator as a bigram mapping from 2 input tokens to 1 result token.

Following this pattern, we define operators A_n.T3, A_n.T4 and A_n.T5 with 3, 4 and 5 digit accuracy respectively:

- A_n.T3 = A_n.T + (A_{n-1}.T2 // 10) aka A_n.T + A_{n-1}.SC2
- A_n.T4 = A_n.T + (A_{n-1}.T3 // 10) aka A_n.T + A_{n-1}.SC3
- A_n.T5 = A_n.T + (A_{n-1}.T4 // 10) aka A_n.T + A_{n-1}.SC4

The value A4.T5 is accurate as it integrates *SC* and cascading *SS* data all the way back to and including A0.T. The values A1.T2, A2.T3, A3.T4 are also all accurate. If the model knows these values it can calculate answer digits accurately:

- A1 = A1.T2 % 10
- A2 = A2.T3 % 10
- A3 = A3.T4 % 10
- A4 = A4.T5 % 10

A_n.T	A_n.*SA*	A_n.*SC*	A_n.*SS*
0	0	0	0
1	1	0	0
...	...	...	...
8	8	0	0
9	9	0	1
10	0	1	0
...	...	...	...
17	7	1	0
18	8	1	0

Table 10: Converting a A_n.T value into a *SA* , *SC* or *SS* value.

A_n**.T**	A_n**.T2 if** A_{n-1}**.SC==0**	A_n**.T2 if** A_{n-1}**.SC==1**
0	0	1
1	1	2
...	...	...
9	9	10
10	10	11
...	...	...
17	17	18
18	18	19

Table 11: Calculating A_n.T2 from A_n.T and A_{n-1}.T

- A5 = A4.T5 // 10

In this hypothesis, **all** the answer digits are accurately calculated using the nodes in positions 8 to 11. This hypothesis 2 is feasible, elegant and compact - reflecting the authors (human) values for good code.

Experimenation shows the model does not implement this hypothesis. It retains the long staircase *SA* calculations in positions 11 to 16. Why? Two reasons suggest themselves:

- Hypothesis 2 is too compact. The model is not optimising for compactness. The long staircase is discovered early in training, and it works for simple questions. Once the overall algorithm gives low loss consistently it stops optimising.
- Hypothesis 2 accurately predicts **all** answer digits in step 11 - reflecting the authors (human) values for good code. The model is not motivated to do this. It just needs to accurately predict A5 as 1 or 0 in step 11 and A4 in step 12 - nothing more.

We abandoned this hypothesis.

J Appendix: Addition Hypothesis 3

The hypothesis 3 pseudo-code was derived iteratively by obtaining experimental results and mapping them to mathematical operations. Some of the experiments and mappings were:

- Ablation experiments show that the A5 value is **accurately** calculated in prediction step 11 using 5 attention heads and 5 MLP layers. The pseudo-code accurately calculates A5 while constraining itself to this many steps.
- Ablating the nodes one by one shows which answer digit(s) are reliant on each node (Ref Table 1). Most interestingly, ablating P10.L0.H1 impacts the answer digits A5, A4, A3, A2 (but not A1 and A0). This node is used in the calculation of A5, A4, A3, A2 in prediction steps 11, 12, 13 and 14. These relationships are constraints that are all obeyed by the pseudo-code.
- The pseudo-code has 4 instances where $A_n.ST$ is calculated using TriCase. PCA of the corresponding nodes (P8.L0.H1, P9.L0.H1, P11.L0.H2 and P14.L0.H1) shows tri-state output for the specified D_n. (see Figure 8).
- The pseudo-code has 4 instances where compound functions using TriCase and TriAdd to generate tri-state outputs. PCA of the corresponding nodes (P11.L0.H1, P12.L0.H1 and P13.L0.H1) shows tri-state output for the specified D_n. (see Figure 8).
- Activation patching (aka interchange intervention) experiments at attention head level confirmed some aspects of the calculations (see § K for details.
- The pseudo code includes calculations like A1.ST which it says is calculated in P9.L0.H1 **and** P9.L0.MLP. Ablation tells us both nodes are necessary. For the attention head we use the PCA results for insights. We didn't implement a similar investigative tool for the MLP layer, so in the pseudo-code we attribute the calculation of A1.ST to both nodes.
- For P10.L0.H1, the attention head PCA could represent either a bi-state or tri-state output (see Figure 9). The MLP layer at P10.L0.MLP could map the attention head output to either a bi-state or tri-state. We cannot see which. The pseudo-code shows a tri-state calculation at P10.L0.MLP, but with small alterations the pseudo-code would work with a bi-state output.
- For P15.L0.H1 the attention head PCA could represent either a bi-state or tri-state output (see Figure 9). The pseudo-code shows a bi-state calculation A0.SC at P15.L0.H1, but with small alterations the pseudo-code would work with a tri-state output.
- The calculation of A1.ST2 in P14.L0.H1 is a interesting case. The model needs A1.ST2 for A2 accuracy. The model could simply reuse the accurate A1.ST2 value calculated in P10. Activation patching shows that it does not. Instead the P14 attention heads calculate A1.ST1 from D1 and D'1 directly, and only relies on the P10.D1.ST2 value in the case where A1.ST2 != A1.ST. That is, the calculation is "use P14.A1.ST1 value else use A1.ST2 values". This aligns with the model learning the P10.A1.ST calculation early in training (for 90% accuracy) and later learning that P10.A1.ST2 contains additional information it can use to get to six nines accuracy.

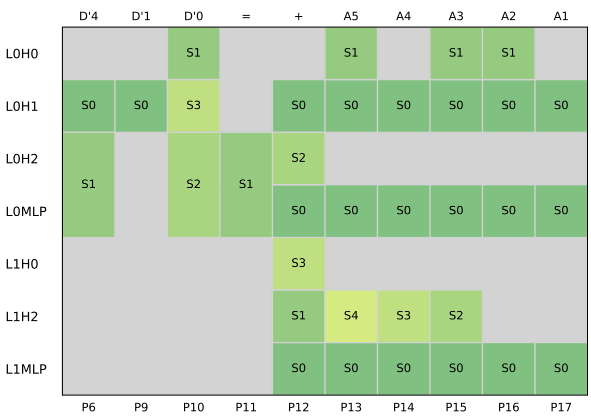

Figure 5: For a sample **5-digit** 2-layer 3-head **addition** model, this map shows a compacted view of all useful token positions (horizontally) and all useful attention heads and MLP layers (vertically) used in predictions as green cells. Each cell shows the simpliest (lowest **complexity**) quanta S0, S1, etc impacted when we ablate each node. To answer S0 questions, only the S0 nodes are used. To answer S1 questions, S0 and S1 nodes are used, etc. The model only uses nodes in nine token positions.

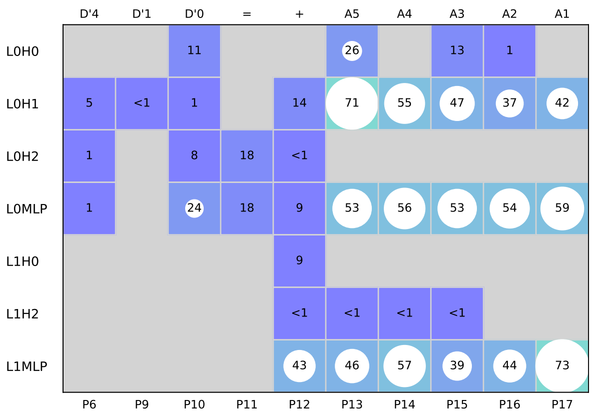

Figure 6: This map shows the **% of enriched questions** that fail when we ablate each node in a **5-digit** 2-layer 3-head addition model. The model only uses nodes in token positions P8 to P16 (i.e. tokens D'2 to A1). Lower percentages correspond to rarer edge cases. The grey space represents nodes that are not used by the model.

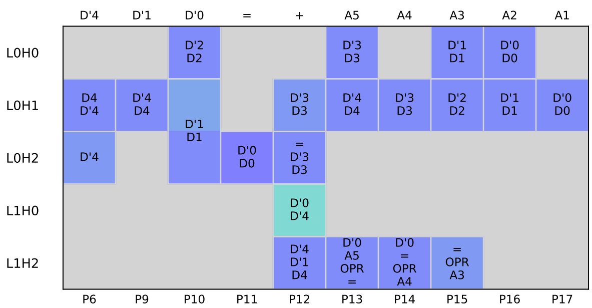

Figure 7: This map shows the input tokens each attention head attends to at each token position in a **5-digit** 2-layer 3-head addition model. At token position P11 the model predicts the first answer digit A5. **All** digit pairs (e.g. D2 D'2) are attended to by P11.

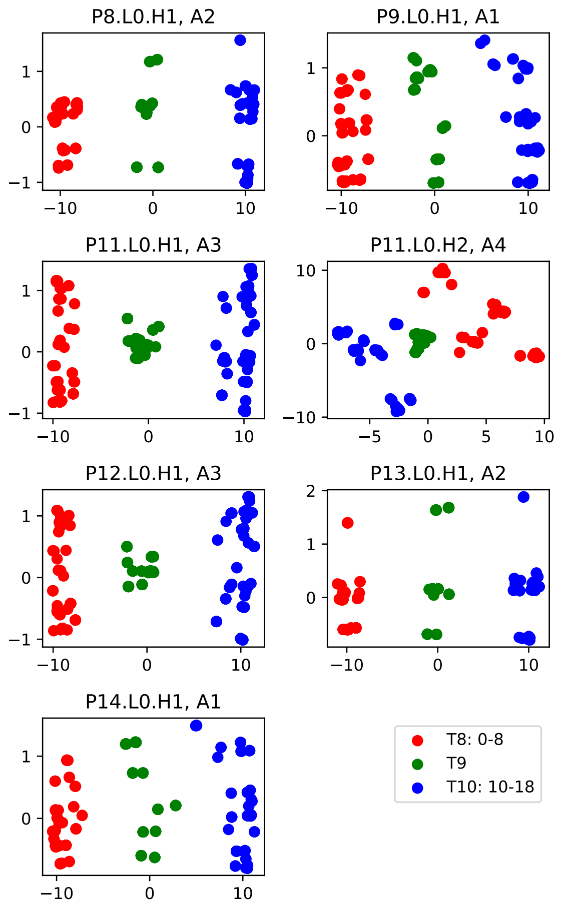

Figure 8: For 5-digit addition, for these attention heads, for exactly 1 answer digit $A_n$each, PCA shows these interpretable results. The dot colours show the TriCase value of each question. The PCA data and TriCase quanta are both tri-state and strongly correlated.

K Appendix: Addition Interchange Interventions

To test the hypothesis 3 mapping of the mathematical framework (casual abstraction) to the model attention heads, various "interchange interventions" (aka activation patching) experiments were performed on the model, where

- A particular claim about an attention head has selected for testing.
- The model predicted answers for sample test questions, and the attention head activations were recorded (stored).
- The model then predicted answers for more questions, but this time we intervened during the prediction to override the selected attention head activations with the activations from the previous run.

Using this approach we obtained the findings in Tab. 12.

L Appendix: N-Digit Addition

The addition models perform addition accurately. Visualizations that provided insights into the behavior of the model, aiding our interpretation of the algorithm, are below:

Some notes about the models:

- The models selected different attention heads in the early positions to use to do the same logical calculations.
- Some models use 2 attention heads per digit to do the *SA* calculation, whereas some models only uses one (and so are more compact).
- The PCA trigrams have difference appearances in different models (but the same interpretable clusters). Refer Figures 8

Nodes	Claim: Attention head(s) perform ...	Finding: Attention head(s) perform a function that ...
P8.L0.H1 and MLP	A2.ST = TriCase(D2, D’2) impacting A4 and A5 accuracy	Based on D2 and D’2. Triggers on a A2 carry value. Provides carry bit used in A5 and A4 calculation.
P9.L0.H1 and MLP	A1.ST = TriCase(D1, D’1) impacting A5, A4 and A3 accuracy	Based on D1 and D’1. Triggers on a A1 carry value. Provides carry bit used in A5, A4 and A3 calculation.
P10.L0.H1 and MLP	A1.ST2 = TriAdd(A1.ST, TriCase(D0, D’0)) impacting A5, A4, A3 and A2 accuracy	Based on D0 and D’0. Triggers on a A0 carry value. Provides carry bit used in A5, A4, A3 and A2 calculation.
P11.L0.H1 and MLP	A3.ST4 = TriAdd(TriCase(D3, D’3), TriAdd(A2.ST, A1.ST2)) impacting A5 accuracy	Based on D3 and D’3. Triggers on a A3 carry value. Provides carry bit used in A5 calculations.
P11.L0.H2 and MLP	A4.ST = TriCase(D4, D’4) impacting A5 accuracy A4	Based on D4 and D’4. Triggers on a A4 carry value. Provides carry bit used in A5 calculation.
P12.L0.H0+H2 and MLP	A4.SA = (D4 + D’4) % 10 impacting A4 accuracy A4	Sums D4 and D’4. Impacts A4.
P13.L0.H0+H2 and MLP	A3.SA = (D3 + D’3) % 10 impacting A3 accuracy	Sums D3 and D’3. Impacts A3.
P14.L0.H0+H2 and MLP	A2.SA = (D2 + D’2) % 10 impacting A2 accuracy	Sums D2 and D’2. Impacts A2.
P14.L0.H1 and MLP	(D1 + D’1) / 10 + P10.A1.ST2 info impacting A2 accuracy	Calculates P10.A1.ST1 but add P10.A1.ST2 info when A1.ST != A1.ST2. Impacts A2
P15.L0.H0+H2 and MLP	A1.SA = (D1 + D’1) % 10 impacting A1 accuracy	Sums D1 and D’1. Impacts A1.
P15.L0.H1 and MLP	A0.SC = (D0 + D’0) / 10 impacting A1 accuracy	Triggers when D0 + D’0 >10. Impacts A1 digit by 1
P16.L0.H0+H2 and MLP	A0.SA = (D0 + D’0) % 10 impacting A0 accuracy	Sums D0 and D’0. Impacts A0.

Table 12: Interchange Interventions experiments used activation patching to test the claims addition hypothesis 3 made for each attention head in a sample mixed model. Experimental results are consistent with hypothesis 3 for all nodes.

+ve Sub	Contains	-ve Sub	Contains	Like
M0	*MD*	N/A	N/A	*S0*
M1	*MD,MB*	*N1*	*ND,NB*	*S1*
M2	*MD,MB*x2	*N2*	*ND,NB*x2	*S2*
M3	*MD,MB*x3	*N3*	*ND,NB*x3	*S3*
M4	*MD,MB*x4	*N4*	*ND,NB*x4	*S4*

Table 13: We define “positive-answer subtraction" and “negative-answer subtraction" calculation complexity quanta that parallel the addition quanta.

- Per answer digit, some models use the *SC* calculation, whereas some models optimize it out and rely solely on the ST value (and so are more compact).

M Appendix: Mixed Model Initialization

We experimented with three approaches to re-using the trained addition model in the “mixed" (addition and subtraction) model:

- Initialize Only: Initialize the untrained mixed model with the addition model weights before training begins.

- Freeze Attention: As per “Initialize Only", but also every 100 training steps recopy the attention head weights from the addition model into the partially-trained mixed model.

- Freeze All: As per “Initialize Only", but also every 100 training steps recopy the entire ad-

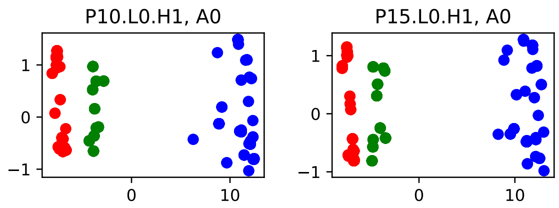

Figure 9: For 5-digit addition, for these attention heads, for exactly 1 digit $A_n$each, PCA shows these interpretable results. The dot colours show the TriCase value of each question. The PCA and TriCase data are strongly correlated, but the PCA data has 2 states.

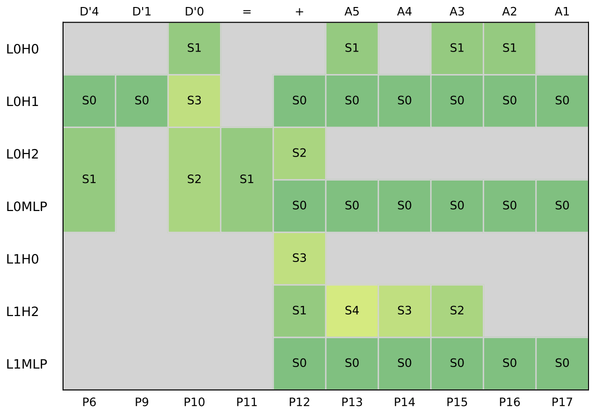

Figure 10: This map shows the simpliest (lowest **complexity**) quanta S0, S1, etc impacted when we ablate each node in the **5-digit** 2-layer 3-head **addition** model. To answer S0 questions, only the S0 nodes are used. To answer S1 questions, S0 and S1 nodes are used, etc.

dition model (attention heads and MLP layers) into the partially-trained mixed model.

Our intuition was that "Initialize Only" would give the mixed model the most freedom to learn new algorithms, but that the "Freeze Attention" and "Freeze All" approaches would make the resulting trained mixed model easier to interpret (as we could reuse our addition model insights).

After experimentation we found that the "Initialize Only" approach was the only one that quickly trained to be able to do both addition and subtraction accurately. We concluded that the other two methods constrainws the model's ability to learn new algorithms too much.

We also experimented with "where" in the model we inserted the addition (6-digit, 2-layer, 3-head) model into the slightly larger (6-digit, 3-layer, 4-head) mixed model. That is, do we initialize the first 2 layers or the last 2 layers of the mixed model? Also do we initialize the first 3 attention heads or the last 3 attention heads of the mixed model? Our

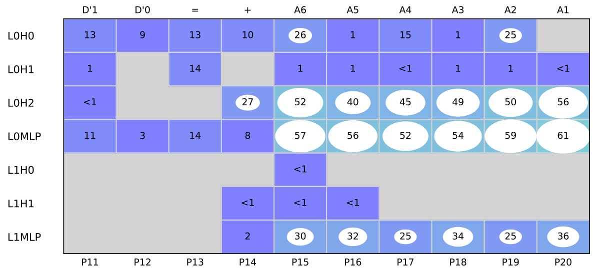

Figure 11: This map shows the **% of questions** that fail when we ablate each node in the **6-digit** 2-layer 3-head addition model. The model only uses nodes in token positions P11 to P20. Lower percentages correspond to rarer edge cases. The grey space represents nodes that are not useful.

intuition was that initializing the first layers and heads would be more likely to cause the model to re-use the addition circuits adding interpretability, so we used this approach.

N Appendix: N-Digit Subtraction

The mixed models perform addition and subtraction accurately. Visualizations that provided insights into the behavior of the model, aiding our interpretation of the algorithm, are below:

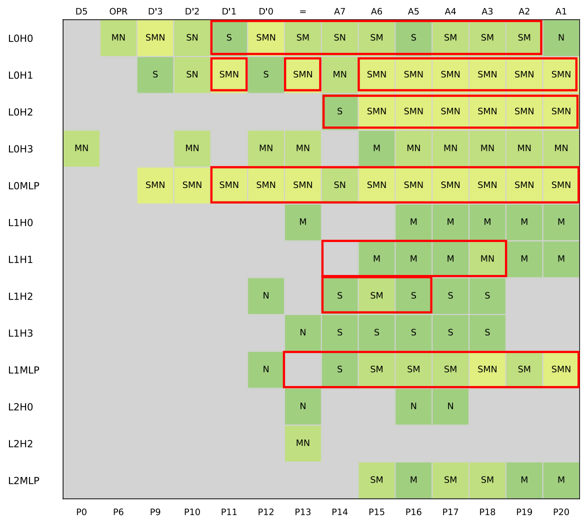

Figure 12: This map of a sample 6-digit **mixed** model shows the 98 nodes used to predict answers to addition (S), positive-answer subtraction (M) and/or negative-answer subtraction (N) questions. Before training the mixed model, 48 nodes were initalized pre-training with a smaller **addition** model's weights. These are have a red border. During mixed model training, 39 of 48 of the initalized monosemantic nodes were generalized (become poly-semantic) and now help predict two or three question classes.

Some notes about the mixed models:

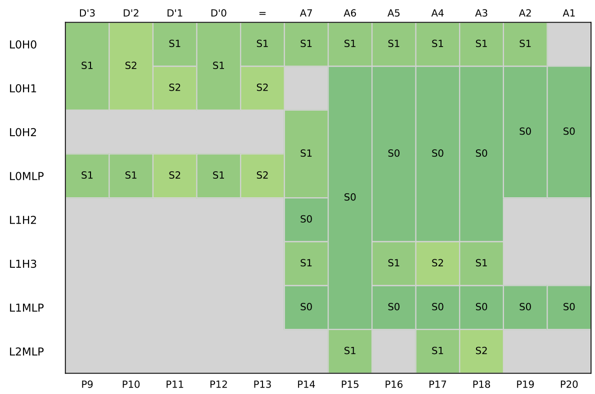

Figure 13: This map shows the simplest **complexity** quanta S0, S1, etc used in each useful node of the **6-digit** 3-layer 4-head **mixed** model when doing **addition** questions.

Figure 14: This map shows the simpliest **complexity** quanta M0, M1, etc used in each useful node of the **6-digit** 3-layer 4-head **mixed** model for **subtraction** questions with positive answers.

- All the notes about the addition model (above) also apply to the mixed model.
- The model contains a new sub-task that stands out: The algorithm relies on calculations done at token position P0, when the model has only seen one question token! What information can the model gather from just the first token? Intuitively, if the first token is a “8" or “9" then the first answer token is more likely to be a “+" (and not a “-"). The model uses this heuristic even though this probabilistic information is sometimes incorrect and so will work against the model achieving very low loss.