
Iteration Head: A Mechanistic Study of Chain-of-Thought

Vivien Cabannes
FAIR, Meta AI

Charles Arnal
Datashape, INRIA

Wassim Bouaziz
FAIR, Meta AI

Alice Yang
FAIR, Meta AI

Francois Charton
FAIR, Meta AI

Julia Kempe
Courant University and Center for Data Science, NYU & FAIR, Meta AI

Abstract

Chain-of-Thought (CoT) reasoning is known to improve Large Language Models both empirically and in terms of theoretical approximation power. However, our understanding of the inner workings and conditions of apparition of CoT capabilities remains limited. This paper helps fill this gap by demonstrating how CoT reasoning emerges in transformers in a controlled and interpretable setting. In particular, we observe the appearance of a specialized attention mechanism dedicated to iterative reasoning, which we coined "iteration heads". We track both the emergence and the precise working of these iteration heads down to the attention level, and measure the transferability of the CoT skills to which they give rise between tasks.

1 Introduction

In the rapidly evolving field of artificial intelligence, Large Language Models (LLMs) have emerged as a pivotal component [45]. Their ability to understand, generate, and manipulate human language has opened up new avenues towards advanced machine intelligence. Interestingly, despite being primarily trained on next-token prediction tasks, LLMs are able to produce much more sophisticated answers when asked to generate steps of reasoning [30, 58]. This phenomenon, often referred to as Chain-of-Thought (CoT) reasoning, and illustrated on Table 1, appears paradoxical: on the one hand, LLMs are not explicitly programmed to reason; on the other hand, they are capable of following logical chains of thoughts to produce relatively complex answers.

Table 1: Chain-of-Thought consists in eliciting reasoning steps before answering (A) a question (Q).

[Q] What is $8 \times 8 \times 3$? : [A] 210.

[Q] What is $8 \times 8 \times 3$? Take it step by step. : [A] $8 \times 8 = 64$, $64 \times 3 = 192$. It is 192.

Recent studies have shown that the class of problems a transformer can solve with single-token prediction, i.e. by outputting a single token meant to be the correct answer, is rather limited [24, 54, 15]. In contrast, when transformers are allowed to freely generate tokens before providing a final answer, they can use those generated tokens as a tape to emulate a form of Turing machine [48]. This enables them to solve a larger class of problems [38, 18, 39, 35]. However, our understanding of why and how transformers gain CoT abilities when trained with next-token predictions remains limited. We aim to provide insights on the matter.

Summary of Contributions. We adopt a “mechanistic interpretability” approach [see 17]: we work with simple, controlled problems and architectures that capture the key aspects of the problem and allow us to observe and analyze, down to the network’s weights and attention, the emergence of CoT in our models. In practice:

- We describe the simple yet rich setting of iterative tasks and iterative algorithms, including three simple examples: a copying, a polynomial iteration, and the parity problems.
- We explain why such problems are hard to solve for transformers with single-token prediction. Conversely, we describe how a certain distribution of weights within the first two attention layers of a transformer, which we call an “iteration head”, enables a transformer to solve iterative tasks with CoT reasoning with relative ease.
- We hypothesize that iteration heads naturally appear in transformers trained on (hard enough) iterative tasks, and verify this hypothesis in small-scale experiments.
- Ablation studies demonstrate the impact of the training set and choice of hyperparameters in their emergence. We also observe the good transferability of the iterative reasoning skills granted by the attention heads from one iterative task to another, from which we deduce the usefulness of data curation.

Our controlled yet illustrative experimental setup sheds light on the emergence of CoT capabilities in larger LLMs, whose attention patterns are much harder to interpret. In particular, our experiments suggest that transformers are likely to develop “inner circuits” specially dedicated to multistep reasoning, which can then be applied, in combination with more specialized skills, to a variety of tasks that share the same underlying logical structure. This gives a credible explanation of the strong CoT reasoning capabilities of current state-of-the-art LLMs, as their training corpora (human-written texts, computer code) include many examples of complex multistep reasoning.

Related Work. This work is set in the realm of mechanistic interpretability [e.g., 43, 7]. A top-down line of work is trying to explicit algorithms implemented by transformers in the wild [e.g. 57, 20, 25], although some findings might be fallacious [9]. A bottom-up line of work, to which we belong, consists in building understandings from small models that are relevant for bigger models, in particular regarding in-context learning [see, e.g. 61, 19, 8, 23, 2, 34, 49, 16, 60]. In-context learning relates to the reproduction of reasoning patterns that appear in a prompt or context [10]. In contrast, our study of CoT relates to reproducing reasoning patterns that appear in the training set.

2 Controlled Setup: Learning Iterative Algorithms

Human language and human reasoning are often organized in a multistep, cumulative fashion, with each new thought or group of sentences building upon the ones that precede to work towards some final conclusion. LLMs naturally benefit from learning such reasoning patterns: not only are they prevalent through much of their training data, but they also represent an efficient way to divide the total processing effort required into easier intermediate steps. In what follows, *we choose to focus on iterative algorithms and iterative tasks as a controlled proxy for more general forms of CoT reasoning*. Indeed, though conceptually simple, iterative algorithms exhibit a key property: they are simultaneously hard to learn for transformers using next-token predictions, and comparatively easy to learn using CoT reasoning. As such, iterative tasks are ideally suited to illustrate the usefulness of CoT reasoning, and to study its emergence.

Algorithm 1 Iterative Schemes

```

s = Init
for x in Sequence do
    s ← F(s, x);
end for
return s

```

We define *iterative algorithms*, or iterative schemes, as follows: an iterative algorithm is the combination of an input sequence, denoted as *Sequence*, and made of L elements $(x_t)_{t \in [L]}$ (with $[L] = \{1, \dots, L\}$), and an internal state, denoted as s , initialized to some default value $s_0 = \text{Init}$, and updated as the sequence is processed according to some rule $s_t = F(s_{t-1}, x_t)$ for some function F . Pseudo-code illustrating the concept is provided

by Algorithm 1, see also Figure 1. By extension, we informally call iterative task a task which is naturally solved by outputting the end product of some iterative algorithm applied to some input sequence. As an example, consider the *parity problem*, i.e. the problem of computing the parity of the sum of a sequence of 0s and 1s: it can be easily framed as an iterative task. Using the notations of Algorithm 1, let the initial state `Init` be equal to 0, and let $F(s, x)$ be equal to 0 if s is equal to x , and 1 otherwise. Then the final s_L gives the parity of the sum. Although this task could also be solved in a non-iterative fashion, the iterative solution can be seen as simpler and more parsimonious.

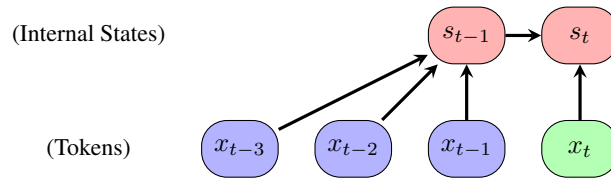


Figure 1: Arguably, reasoning involves updating an internal state (red) as new information is processed (green). The diagram above, where each element represents a piece of information, is an abstract depiction of this idea. This observation motivates our use of iterative tasks as a proxy for more general reasoning processes. At first glance, a limitation of transformers is their lack of an internal state, which makes it challenging to implement this diagram [32].

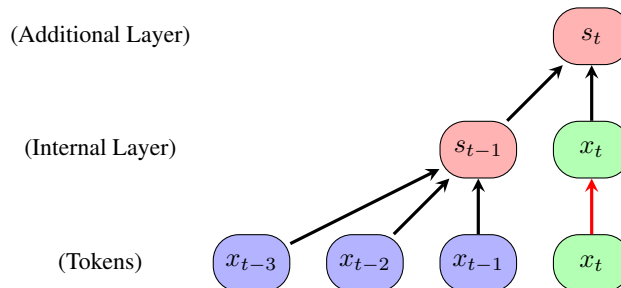


Figure 2: A single transformer layer cannot implement the diagram from 1, as it cannot access its previous outputs. This limitation can be bypassed by stacking transformer layers, as illustrated here. The red arrow indicates a residual connection. This naive method requires as many layers as there are reasoning hops.

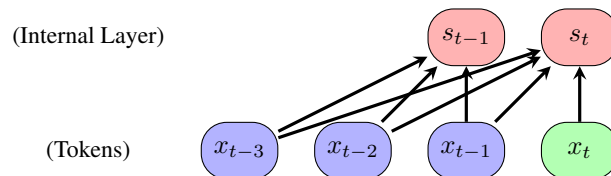


Figure 3: Alternatively, a transformer could compute each state s_t from scratch. This implementation does not require additional layers, but it is not parsimonious, which could lead to computational inefficiencies. This explains the difficulty for a transformer to output the final answer of a chain of reasoning within a token (i.e., with next-token prediction, and without chain-of-thought)

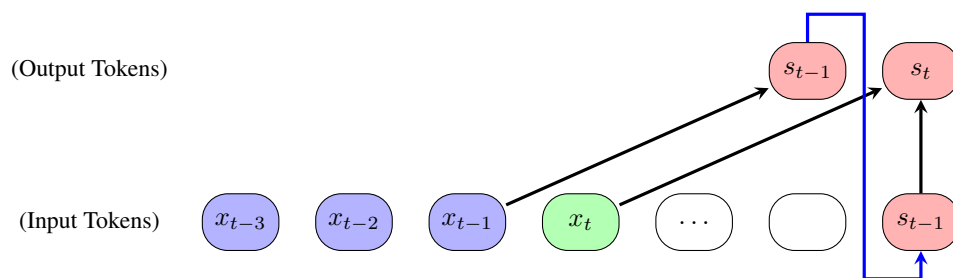


Figure 4: Chain-of-thought addresses this issue by explicitly representing the reasoning process in token space. The auto-regressive nature of LLMs (blue arrow) allows for the implementation of iterative algorithms, as long as the states are encoded in token space. A concrete implementation of such a mechanism, which we call *iteration head*, is described in Section 3. In practical applications of LLMs, one could imagine earlier layers summarizing the t -th input sentence (or some other coherent semantic information of varying token length) into x_t , as well as summarizing the generated CoT sentences into some s_{t-1} , with later layers translating the state s_t into readable text.

Can a Transformer Learn Iterative Algorithms? Briefly summarized,¹ a transformer is composed of a series of transformer blocks and operates on the space of sequences. A transformer block performs cross-operations that combine elements of a sequence through the use of *attention heads* to generate new sequences, and parallel operations applied to each element of a sequence separately through the use of *feedforward layers* (or MLP, i.e., multi-layer perceptrons). Auto-regressive transformers in particular are trained to perform next-token prediction; in other words, from a training corpus that contains sequences (z_t) of tokens, the transformer is trained to output z_{t+1} from the truncated sequence $S_t = (z_r)_{r \in [t]}$.

Given a certain number of transformer blocks, a transformer can only apply a corresponding number of cross-operations to predict the next token. This limits its ability to learn even relatively simple iterative tasks, see Figures 2 and 3. E.g., consider the task where the input sequence is (x_1, \dots, x_L) , possibly restricted to $x_i \in [a, b]$ for some $a < b$, and the desired output is the product $\prod x_i$. The product is multilinear in the entries of the input sequence (x_1, \dots, x_L) . If we model the output of an attention layer as sums of monomials of degree at most three in its input variables (due to key-query-value interaction), this makes learning the task quite hard for a transformer, and bounds the maximum length of the sequences that a transformer with a given number of blocks can correctly process [see 51, and related literature for formal discussions on the matter that capture this log-depth dependency].

However, when transformers are allowed to generate many tokens before providing an answer, which implicitly lifts the constraint on the number of operations performed by the transformer (see Figure 4), the picture changes [48, 39, 35] [see also 14, 21]. In particular, Figure 5, explained in the next subsection, illustrates how a two-layer transformer can implement what we named an “iteration head”. This potentially enables it to learn any iterative algorithm, assuming that its second layer MLP is big enough to implement any successor function $F : (x_t, s_{t-1}) \mapsto s_t$.

Synthetic Data. To study the emergence of CoT in controlled settings, we introduce two simple iterative problems. The first problem is a straightforward instance of Algorithm 1, where the tokens and the states are elements of the finite field $\mathbb{F}_p = \mathbb{Z}/p\mathbb{Z}$ (for some prime number p), i.e., integers modulo p , and the iterative step is the evaluation of a polynomial function $P \in \mathbb{F}_p[X, Y]$ in those two variables:

$$x \in \mathbb{F}_p, \quad \text{Init} = 0, \quad F(s, x) = P(s, x) \quad (\text{Polynomial Iteration})$$

Letting $P(s, x) = s + x$ and $p = 2$, the problem reduces to the so-called parity problem:

$$x \in \{0, 1\}, \quad \text{Init} = 0, \quad F(s, x) = s + x. \quad (\text{Parity Problem})$$

For ease of study, we also consider an even simpler problem: the copying problem, where the goal is simply to output an exact copy of the input sequence.

$$x \in \{0, 1\}, \quad F(s, x) = x \quad (\text{Binary Copy})$$

Note that there is a small abuse of notation here, since we are interested in the unrolled sequence of states produced iteratively by Algorithm 1, rather than the last token only. While copying may seem like an overly simplistic task, it should be put in parallel with the seminal work of Olsson et al. [44] that advocates studying a copying mechanism to better understand in-context learning.

For each of our problems, we encode the data, i.e. the sequences (z_t) , in the following form:

$$[\text{Problem}] \quad [x_1] \quad [x_2] \quad \cdots \quad [x_L] \quad [\text{EoI}] \quad [s_1] \quad [s_2] \quad \cdots \quad [s_L] \quad [\text{EoS}].$$

A first token indicates the problem generating the sequence (e.g., “copy”, or “parity”), after which L input tokens x_t are provided. The end of the input is specified by an end-of-input token (EoI). Subsequent tokens encode the states s_t of Algorithm 1 at each iteration, until termination, which is indicated by an end-of-sequence token (EoS).

3 One Head to Rule Them All

We have discussed how transformers are limited in the iterative tasks that they can efficiently solve using only next-token prediction. By contrast, we describe in this section a certain distribution of weights which, if correctly learnt, would allow a two-layer transformer to efficiently implement iterative algorithms by using chain-of-thought reasoning. After that, we perform various experiments to identify the conditions under which this theoretical circuit does appear.

¹We assume that the reader is familiar with the transformer architecture [see, e.g., 56, 36, for details].

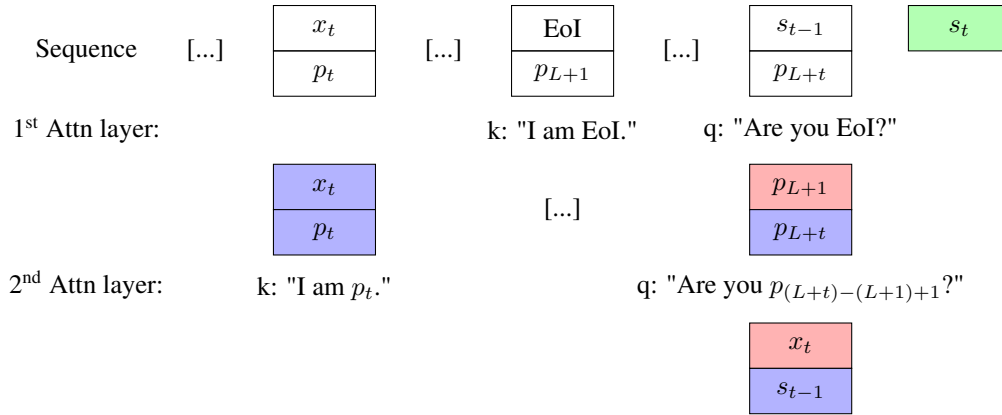


Figure 5: Implementation of an iteration head with a two-layer transformer. Contiguous box: superposition in high-dimensional space. Blue: information brought to working space thanks to residual connections. Red: information brought thanks to attention. Green: next-token prediction. The first layer MLP implements a subtraction $t = (L + t) - (L + 1) + 1$ for the second attention to be able to query p_t from (p_{L+1}, p_{L+t}) . The second layer MLP implements F to be able to predict s_t from (s_{t-1}, x_t) , with the “end-of-input” mark assimilated to the initial state s_0 of Algorithm 1.

3.1 Theoretical Circuit

This subsection describes a natural way to implement an iterative algorithm with a transformer. Let us consider a prompt $(x_t)_{t \in [L]}$, to which we append a special “end-of-input” (EoI) token that marks the end of the input. The completion sequence will be generated with the t -th new element encoding for the state variable after t steps of Algorithm 1. The t -th new element (i.e. the $L + 1 + t$ -th token of the full sequence) is produced as follows. The first attention head is tasked with retrieving the position of the end of the initial prompt, i.e. the position of the EoI token. As illustrated in Figure 5, it does so using a query-key combination which informally encodes the question “Are you EoI?” and the answer “I am EoI.”. Thus it extracts the positional encoding p_{L+1} (which is the value associated to the $L + 1$ -th token) regardless of the sequence length $L \in \mathbb{N}$, and brings p_{L+t} into its working space as well through the residual connection (we formalize such statements in the next paragraph). As shown further below in Figure 5, the next attention head then generates a query “Are you p_t ?” from p_{L+1} and p_{L+t} , which is answered positively by a key “I am p_t ” associated to the t -th position. Hence the head retrieves the value associated to this position, which is x_t . It also obtains s_{t-1} (or rather the approximation of it that was produced at the previous step) through the residual stream. The MLP can finally compute the new state $s_t = F(s_{t-1}, x_t)$ from s_{t-1} and x_t . This can always be done by a large-enough MLP assuming that the second attention layer outputs all the relevant information regarding s_{t-1} and x_t , as a result of universal approximation [27]. Note that the operations performed by the two attention layers are totally independent from the precise iterative task considered, i.e. from the choice of F ; their only goal is to retrieve x_t and s_{t-1} . We call the pattern of weights that realize these operations, as well as the underlying algorithm, an “iteration head”.

Information Superposition in Working Spaces. In our description of an iteration head, we have rather informally said that some variable x is “extracted” or “obtained”. Formally, a transformer transforms a sequence $(x_t)_{t \in [L]}$ into a series of sequences $(e_{t,l})_{t \in [L]}$, where l is an index specifying layers, and $e_{t,l} \in \mathbb{R}^d$, with \mathbb{R}^d being referred to as the “working space”. The input tokens and their positions are brought into working spaces using embeddings that are typically learned, then added together. Assuming that the working spaces are high-dimensional enough, and because those embeddings are learned, a transformer can use different parts of \mathbb{R}^d to simultaneously store token and positional information, as if those embeddings were actually *concatenated* rather than added. Likewise, transformer layers output variables are learned functions of their input; if needed, and assuming that d is large enough, $e_{t,l+1}$ can superpose some $e_{s,l}$ and $e_{r,l}$ for different $s, r \leq t$, in which case one may consider $e_{t,l+1}$ as somewhat equivalent to the concatenation of $e_{s,l}$ and $e_{r,l}$. This is why our exposition above focuses on “information pathways”, i.e. which variable is generated using which variable, and sentences such as “ x_t and s_{t-1} are brought to the working space” should be understood as “some vector encoding the relevant information of both x_t and s_{t-1} is produced”.

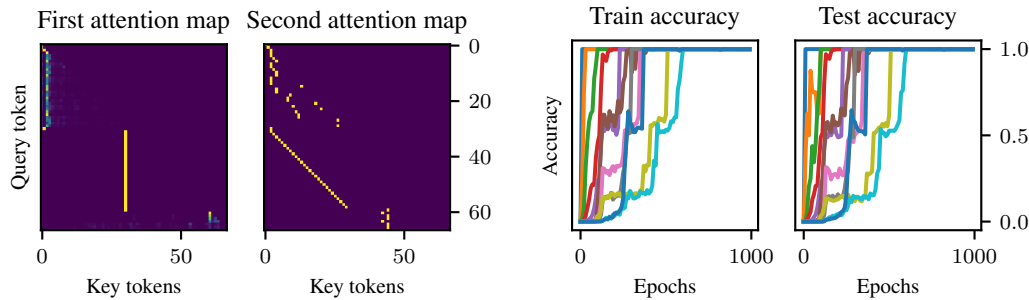


Figure 6: Left: attention maps learned for the parity problem when processing a sequence of length $L = 29$. Yellow indicates high attention score. The yellow line on the left plot shows that all the queries after the EoI token at position $t = 30$ point to the EoI token. In other terms, the first attention implements the “Are you EoI?” query of Figure 5, while the second implements the “Are you p_t ?” query. Right: accuracy dynamics for different sequence lengths when learning the parity problem. We observe fast learning of short sequences (we used the `tab10` color scheme of Matplotlib [28] with $L \in \{8, 11, 14, 17, \dots, 32\}$), and characteristic staircase behaviors.

Approximate Iteration Heads. Iteration heads are an efficient, flexible and parsimonious way to implement iterative algorithms; as such, we expect them to naturally emerge during training. Nonetheless, transformers have flexible architectures that can perform similar operations in different ways. Hence we also expect to see some variations with respect to the schematic architecture described above (see Figures 8 and 13), in particular when the embedding dimension becomes too small for the information superposition from the previous paragraph to be correctly implemented.

3.2 Learning an Iteration Head

In this subsection, we examine the circuit that a transformer actually learns when trained on an iterative task with chain-of-thought. We observe that the theoretical circuit described in the previous subsection does appear in practice.

Experimental Design. Unless otherwise stated, our experimental setup is as follows. Data was generated for the binary-copy, parity, and polynomial iteration problem with $P(X, Y) = XY + 1$ in \mathbb{F}_{11} . For each length L from $L_{\min} = 1$ to $L_{\max} = 32$, we generated $n = 1024$ input sequences of length L (corresponding to a total sequence length of $2L + 3$) uniformly at random for both training and testing sets, creating datasets of $N = 16,384 = 16 \times 1024$ sequences in total.² We utilized auto-regressive transformers [10] with two layers and one attention head per layer. The embedding dimension was set to $d = 128$, with learned absolute positional encoding added to the learned token embedding. The weights were optimized over 1000 epochs with Adam [29], a batch size of 256, and a fixed learning rate set to $\gamma = 3 \cdot 10^{-4}$, with default PyTorch parameters otherwise [46]. Our source code is available at <https://github.com/facebookresearch/pal>. Our experiments consumed 12k V100-hours.

Attention Heads. In our initial experiment, we trained a transformer to solve either the parity task or the copying task only. The “iteration head” pattern of weights, described in the previous sub-section, can be seen in the attention maps of the first and second attention layers: an example is reported in Figure 6. Namely, we observe that when the model produces the $(L + t + 1)$ -th token (meant to be s_t), the following happens. The attention of the first transformer block is fully focused on the position of the EoI token, corresponding to the informal query “Is this token equal to EoI?”, creating a yellow line on the left of Figure 6. This allows the first attention layer to retrieve the positional encoding p_{L+1} of the EoI token, in addition to the positional encoding p_{L+t} of the last token of the current sequence (the state s_{t-1}) coming from the residual stream. Using this information, the second attention layer is able to generate the informal query “Is this token in position t ?”, to extract some encoding of the token x_t . Consequently, the attention of the second layer is fully focused on the position of the t -th entry, creating the yellow off-diagonal line on the second plot of

²Note that, even when $x_t \in \mathbb{F}_2$, the expected percentage of collision between the training and testing set decreases exponentially fast with L , ensuring minimal contamination between training and testing.

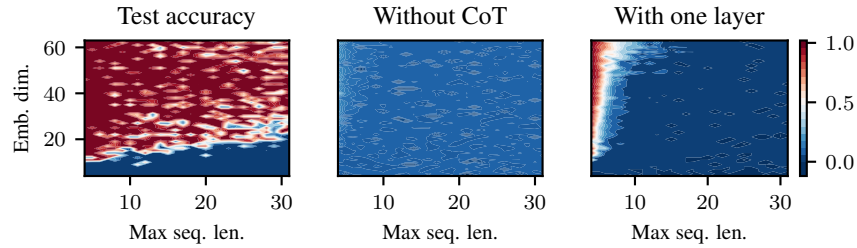


Figure 7: Test accuracy (where red indicates better performance) after learning the polynomial iteration task with $P(X, Y) = XY + 1$ in \mathbb{F}_{11} for 1000 epochs. The accuracy is reported as a function of the embedding dimension (on the y -axis), and the maximum sequence length L_{\max} (on the x -axis). The learning was conducted with a two-layer transformer with CoT (left), without CoT (middle), or with a one-layer transformer with CoT (right). This illustrates the usefulness of CoT and two-layer architectures.

Figure 6. Using the information of x_t and s_t , the following MLP can then compute $s_t = F(x_t, s_t)$. For a given sequence length L , the learned attention maps were found to be invariant to the input token $(x_t)_{t \in [L]}$: the standard deviation of attention patterns computed over all the data was negligible.

Successor Function. We empirically verified that after training a two-layer transformer with one attention head per layer on the copying dataset, fine-tuning only the second layer MLP on parity data enabled us to achieve 100% accuracy on the parity problem. In the context described previously, this transfer was accomplished in fewer than 20 epochs of fine-tuning on the parity dataset. This confirms that the feed-forward layer of the second transformer block is computing the successor function F . In more general contexts, we found the successor function to be implemented jointly by the second layer MLP, the second attention values and output matrices, as well as the un-embedding matrix.

Position Subtraction. The accuracy of the model decreases with the embedding dimension d , as shown on the left of Figure 7. Figure 8 suggests that when d is small, the first attention layer remains capable to accurately locate the “EoI” token, but the second attention layer struggles to retrieve x_t . This can be explained as follows: in a model that implements an iteration head, the first layer’s MLP, in conjunction with the second attention key and query matrices, is expected to generate the query-key pair “Are you p_t ” and “I am p_t'' ” by transforming p_t on the one hand, and a superposition of p_{L+t} and p_{L+1} on the other hand, so that the end results are aligned. More abstractly, this encodes the positional subtraction $L + t - (L + 1) + 1 = t$. In high dimensions, it is relatively easy to find a set of weights to align a large number of vectors (viz., the ones encoding for (p_{L+1}, p_{L+t}) and for p_t). However, in lower dimensions, this can only be achieved when the vectors form certain special geometrical patterns [see e.g. 40, 63], which the model struggles to learn in our setting, at least with the optimization choices we made.

Evaluation Dynamics. With a sufficiently small model, we might expect to understand the training dynamics quite well, which could in turn provide insights on design choices for larger models to minimize training costs. While a detailed study of the training dynamics of our two-layer transformers is beyond the scope of this paper [see e.g., 41], we note several interesting facts that align with recent findings in the literature, such as the staircase profile of accuracy plots in Figure 6 [1, 3, 13], as well as the usefulness of small batch sizes and large learning rates reported in Figure 12 in Appendix [11] despite the risk of loss spikes [12, 59].

3.3 Ablation Studies

In addition to visualizing the attention map, we validated the learning of iteration heads through attention patching, i.e., intervening to “patch” certain attention maps. Specifically, we observed that patching the ideal attention maps (i.e., zeroing out other routes) does not disrupt perfect accuracy. In contrast, zeroing out the focus on the EoI by the first attention head, or on p_t by the second, reduced performance to near random.

Next-token Prediction; One or Two Layers. As an initial ablation study, we considered the polynomial iteration problem with $P(X, Y) = XY + 1$ in \mathbb{F}_{11} , and compared the performance of

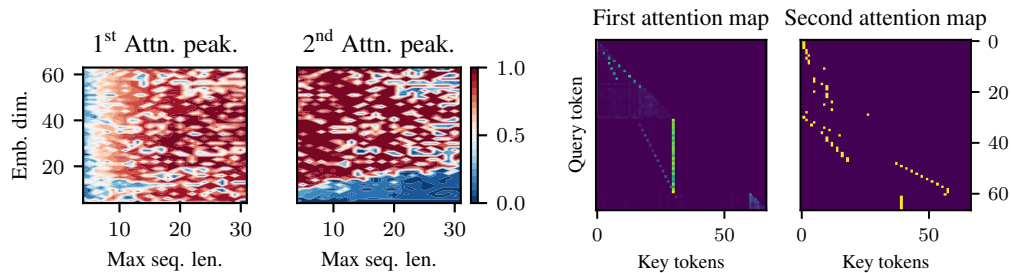


Figure 8: Left: attention peakiness score after 1000 epochs of learning with the polynomial iteration task parameterized by $P(X, Y) = XY + 1$ in \mathbb{F}_{11} as a function of the embedding dimension d and the maximum sequence length L_{\max} . Right: example of attention maps of sub-sampled iteration heads.

CoT reasoning with next-token prediction (i.e., without CoT), as well as CoT with a single layer transformer. Two parameters come into play: the length of the sequence, which can be seen as a difficulty parameter regarding the data; and the embedding dimension, which can be seen as a model capacity parameter [31, 55, 53]. The results, unequivocal in favor of CoT and two-layer transformers, are reported in Figure 7.

Alternative Circuits. Next, we explored other circuits that a two-layer transformer can learn to perform the same tasks as an iteration head. We proceed by assigning a score to measure how closely the attention maps follow the patterns of Figure 6. For the first attention map, we would like a measure of the concentration of the attention at the “Are you EoI? I am” query-key pairs, which correspond to the vertical yellow line from the left of Figure 6. For the second attention map, we would like a measure of the concentration of the attention at the “Are you p_t ? I am” query-key pairs, which corresponds to the yellow off-diagonal from Figure 6. To avoid scaling issues, we define an attention score a_i as “peaky” if it is greater than 50% after softmax averaging. We then measure the average number of peaky scores (within one sequence, and over sequences), i.e., we compute $\sum \mathbf{1}_{a_i > .5}$ instead of $\sum a_i$. This provides a clear measure of the degree to which a transformer is implementing the attention mechanisms described in the previous section.

On the left of Figure 8, we report the average peakiness found for the first and second attention layers when training a transformer for different maximum sequence lengths L_{\max} and embedding dimensions d . We only run one experiment per pair (L_{\max}, d) with a fixed random seed. The texture of the figure indicates a certain randomness between runs for similar pairs (L_{\max}, d) . The first attention layer almost always learns the “Are you EoI?” query-key combination, except when the maximum sequence lengths are very small. In these cases, the transformer might find different circuits to solve for different sequence lengths.

The second attention layer tends to vary more. In particular, for small embedding dimensions, the position subtraction might be challenging for the transformer to perform, leading it to find alternative mechanisms. For instance, the first layer attention might perform a previous token copy when processing the input tokens, superposing the current token x_t and the previous one x_{t-1} in the current working space. This allows the second layer to solely point at every other position, e.g., only attend even positions $t \in 2 \cdot \mathbb{N}$, either recovering the current token x_t , or the previous one x_{t-1} . Implementing position subtraction towards even positions only reduces the learning capacity needed by the first layer MLP [see 11, for related scaling laws]. Such a sub-sampling mechanism is notably observed on the right of Figure 8.

4 Skill Transfer

Some skills might be easier to acquire when trained on certain data rather than others, highlighting the importance of data curation when training LLMs. For example, datasets of code or math [e.g., 26] might exhibit formal reasoning structures that compel LLMs to learn multistep reasoning patterns when trained on them, leading to improved reasoning abilities of the final model, even in plain English [see, e.g., 37]. Our synthetic problems are ideally suited to highlight the mechanisms at play in these observations. This section illustrates how strategic data curation can facilitate learning to solve the parity problem. The crux is to find a dataset that helps the creation of iteration heads, which, once present, significantly eases the learning of the parity problem by a transformer.

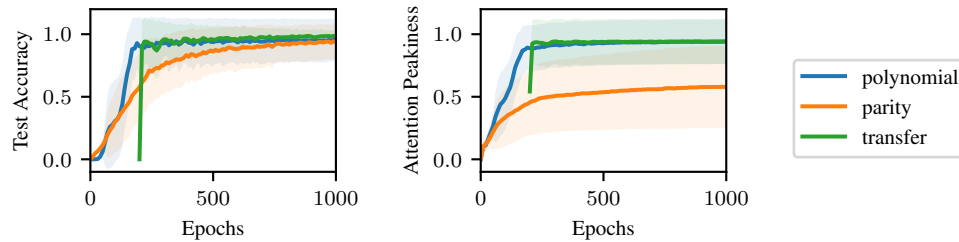


Figure 9: Left: Test accuracy as a function of the number of epochs, averaged over 100 runs, when learning the polynomial iteration task with $P(X, Y) = XY + 1$ in \mathbb{F}_{11} (blue) and the parity problem (orange). Right: The second attention peakiness score indicates whether the network is learning the iteration head described in Figure 5. The green curve corresponds to the accuracy on the parity problem when learning the polynomial iteration for the first 200 epochs before switching the dataset to learn the parity problem.

4.1 Inducing Induction

We now address a simple question: can we “pretrain” a model on a task A, and then “finetune” it on a task B in order to learn to solve the task B with a smaller total number of flops than if we were to learn the task B from scratch? We will see that the answer is positive.

Figure 9 compares three learning scenarios. The learning of the polynomial iteration task corresponding to $P(X, Y) = XY + 1$ in \mathbb{F}_{11} is reported in blue. The learning of the parity problem is reported in orange. Finally, the green curve represents training on the polynomial iteration task for 200 epochs (these epochs are not reported in the graph, hence the curve offset), before switching tasks and continuing the training on the parity problem. When switching from the polynomial iteration dataset to the parity dataset, we chose to reset the Adam buffers to zero. Moreover, our default experimental parameters were changed to $L_{\max} = 16$ and $n = 512$, generating training and testing sets of $N = 8,192 = 16 \times 512$ sequences. The left side of Figure 9 reports testing accuracy averaged over 100 runs, along with its standard deviation. The polynomial iteration task is learned relatively quickly, while the parity problem takes longer. The right side of Figure 9 reports the second attention peakiness score, capturing whether or not the second attention is implementing the “Are you p_t ?” query. After 200 epochs of training with the polynomial iteration task, the iteration head is formed, and fine-tuning the network on the parity problem for less than 30 epochs enables the reuse of this circuit on the parity data (green curve, right plot), thus solving the parity task (green curve, left plot). Overall, the data curation represented by the green plot enables the computation of parities in less than 300 epochs, compared to 1000 epochs when learning solely with parity data.

This example provides a controlled setup to understand the usefulness of data curation when training larger models. It biases the model toward the implementation of specific circuits. In particular, adding code or math datasets to the training of LLMs might induce the learning of more circuits that implement various forms of reasoning patterns. These could be viewed as atomic skills that could be reused to solve more generic problems [see, e.g., 4, for further discussions on skill factorization].

4.2 The Role of Inductive Biases

To illustrate the usefulness of data curation and skill transfer, we needed to find a problem that is hard to learn from scratch. The parity problem was well-suited to play this role in our synthetic setting. On the other hand, the polynomial iteration task with $P(X, Y) = XY + 1$ in \mathbb{F}_{11} was the easier task. One might wonder why learning with $P(X, Y) = XY + 1$ in \mathbb{F}_{11} turned out to be a simpler task than learning parities, which corresponds to $P(X, Y) = X + Y$ in \mathbb{F}_2 . Our intuition is that the parity problem can be solved in many different ways, which leads to competing signals in the gradient for updating the weights, reminiscent of the theoretical study by Shalev-Shwartz et al. [52] [see also 50, 63]. For example, we see on the right of Figure 9 that the standard deviation of the attention peakiness score is quite high when learning with parity data. This can also be observed from the texture in Figure 11 in the Appendix. This creates a *challenging optimization landscape*. In contrast, the polynomial $P(X, Y) = XY + 1$ was chosen to make the final state dependent on the token order. Removing permutation invariance is useful to reduce the variety of circuits that can solve the polynomial iteration task, and seems to speed up the training dynamics. Finally, starting from a pretrained model that already implements an iteration head creates a strong inductive bias toward the

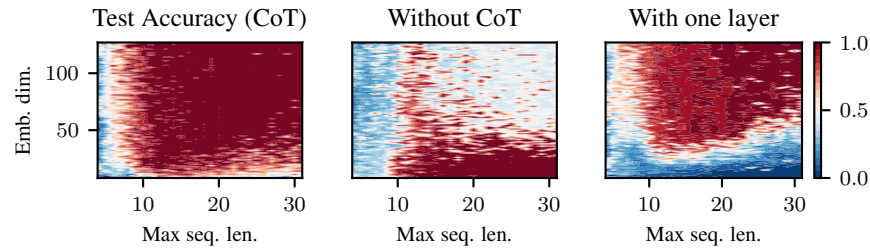


Figure 10: Same as Figure 7, except that we considered the parity problem, and 5000 training epochs.

iteration head circuit to solve the parity problem, allowing the parity problem to be learned within a very small number of epochs.

To deepen our understanding of the parity problem, we conducted the same scaling study depicted in Figure 7 using the parity dataset. The results are presented in Figure 10. Training was conducted over 5000 epochs. The data generation process was slightly modified to generate all sequences of length less than $L = \log_2(1048)$, and these were evenly split between training and testing (instead of generating redundant random sequences). In some sense, the parity problem can be considered relatively easy to solve in a non-iterative fashion: simply add all the elements of sequences, and reduce the sum modulo two. In theory, a single-layer transformer can use uniform attention to bring all the input tokens into superposition as input for the two-layer MLP layer, which is a universal approximator [27]. As a result, the parity with a fixed sequence length can be solved with such an architecture [see e.g. 5]. Indeed, the bottom right of the left and middle plots in Figure 10 indicate that next-token prediction performs better than chain-of-thought for sequence lengths up to $L_{\max} = 32$ with an embedding dimension of $d = 32$. This is due to the difficulty of performing position subtraction necessary for the CoT circuit, compared to the relative ease of performing addition of up to 32 bits with our two-layer architecture. Similarly, we found that a one-layer transformer was able to learn to produce correct CoT sequences for this task, demonstrating the existence of circuits fundamentally different from our iteration head to solve it. Anecdotally, the top right of the left and middle plots of Figure 10 indicate that as the model capacity increases, next-token prediction tends to overfit the training data, while CoT induces the transformer toward understanding the underlying structure that generated the data.

5 Conclusion

In this paper, we have explored the emergence of Chain-of-Thought (CoT) reasoning in Large Language Models (LLMs) through the lens of iterative algorithms. We have shown that, despite being trained on next-token prediction tasks, transformers can learn to solve iterative tasks efficiently using CoT reasoning. In particular, we have demonstrated that a two-layer transformer can implement what we named an “iteration head”, enabling it to learn any iterative algorithm, assuming that it has enough feedforward layers following its two transformer blocks.

We have also shown that data curation can play a significant role in guiding the model towards the implementation of specific circuits. While our study has focused on simple, controlled problems and architectures, we hope that our findings shed light on the emergence of CoT capabilities in larger LLMs, whose attention patterns are much harder to interpret. In particular, they suggest that transformers are likely to develop “inner circuits” dedicated to multistep reasoning, which can then be applied to a variety of tasks that share the same underlying logical structure.

Interestingly, our work also highlights a limitation of the transformer architecture: they are stateless models. Indeed, our CoT implementation of Algorithm 1 requires the generated states (s_t) to have a token representation. This allows us to recover the state of the iterative algorithm at the root (i.e., the input) of the transformer. For complex iterative algorithms, or generic language modeling, it would be more logical to maintain a state internal to the model in the embedding space. The fact that GPT architectures do not allow this is arguably a shortcoming of the current transformer architecture [see 32, 6, 22, 47, 62, for interesting discussions], [see also 42, 33].

Acknowledgments. The author thanks Alberto Bietti, Carles Domingo-Enrich, and Denny Wu for useful discussions.

References

- [1] Emmanuel Abbe, Enric Boix-Adsera, and Theodor Misiakiewicz. The merged-staircase property: a necessary and nearly sufficient condition for SGD learning of sparse functions on two-layer neural networks, 2022.
- [2] Ekin Akyürek, Dale Schuurmans, Jacob Andreas, Tengyu Ma, and Denny Zhou. What learning algorithm is in-context learning? Investigations with linear models, 2023.
- [3] Maksym Andriushchenko, Aditya Varre, Loucas Pillaud-Vivien, and Nicolas Flammarion. SGD with large step sizes learns sparse features, 2023.
- [4] Sanjeev Arora and Anirudh Goyal. A theory for emergence of complex skills in language models, 2023.
- [5] Boaz Barak, Benjamin L. Edelman, Surbhi Goel, Sham Kakade, Eran Malach, and Cyril Zhang. Hidden progress in deep learning: SGD learns parities near the computational limit, 2023.
- [6] Adrien Bardes, Quentin Garrido, Jean Ponce, Xinlei Chen, Michael Rabbat, Yann LeCun, Mahmoud Assran, and Nicolas Ballas. Revisiting feature prediction for learning visual representations from video, 2024.
- [7] David Bau, Jun-Yan Zhu, Hendrik Strobelt, Agata Lapedriza, Bolei Zhou, and Antonio Torralba. Understanding the role of individual units in a deep neural network. *Proceedings of the National Academy of Sciences*, 117(48):30071–30078, September 2020. ISSN 1091-6490. doi: 10.1073/pnas.1907375117. URL <http://dx.doi.org/10.1073/pnas.1907375117>.
- [8] Alberto Bietti, Vivien Cabannes, Diane Bouchacourt, Herve Jegou, and Leon Bottou. Birth of a transformer: A memory viewpoint, 2023.
- [9] Tolga Bolukbasi, Adam Pearce, Ann Yuan, Andy Coenen, Emily Reif, Fernanda Viégas, and Martin Wattenberg. An interpretability illusion for BERT, 2021.
- [10] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020.
- [11] Vivien Cabannes, Elvis Dohmatob, and Alberto Bietti. Scaling laws for associative memories, 2024.
- [12] Vivien Cabannes, Berfin Simsek, and Alberto Bietti. Learning associative memories with gradient descent, 2024.
- [13] Angelica Chen, Ravid Shwartz-Ziv, Kyunghyun Cho, Matthew L. Leavitt, and Naomi Saphra. Sudden drops in the loss: Syntax acquisition, phase transitions, and simplicity bias in MLMs, 2024.
- [14] Timothée Darcet, Maxime Oquab, Julien Mairal, and Piotr Bojanowski. Vision transformers need registers, 2024.
- [15] Grégoire Delétang, Anian Ruoss, Jordi Grau-Moya, Tim Genewein, Li Kevin Wenliang, Elliot Catt, Chris Cundy, Marcus Hutter, Shane Legg, Joel Veness, and Pedro A. Ortega. Neural networks and the Chomsky hierarchy, 2023.
- [16] Benjamin L. Edelman, Ezra Edelman, Surbhi Goel, Eran Malach, and Nikolaos Tsilivis. The evolution of statistical induction heads: In-context learning Markov chains, 2024.
- [17] 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. A mathematical framework for transformer circuits. *Transformer Circuits Thread*, 2021.

- [18] Guhao Feng, Bohang Zhang, Yuntian Gu, Haotian Ye, Di He, and Liwei Wang. Towards revealing the mystery behind chain of thought: A theoretical perspective, 2023.
- [19] Shivam Garg, Dimitris Tsipras, Percy Liang, and Gregory Valiant. What can transformers learn in-context? A case study of simple function classes, 2023.
- [20] Mor Geva, Jasmijn Bastings, Katja Filippova, and Amir Globerson. Dissecting recall of factual associations in auto-regressive language models, 2023.
- [21] Sachin Goyal, Ziwei Ji, Ankit Singh Rawat, Aditya Krishna Menon, Sanjiv Kumar, and Vaishnavh Nagarajan. Think before you speak: Training language models with pause tokens, 2024.
- [22] Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces, 2023.
- [23] 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.
- [24] Michael Hahn. Theoretical limitations of self-attention in neural sequence models. *Transactions of the Association for Computational Linguistics*, 2020.
- [25] Michael Hanna, Ollie Liu, and Alexandre Variengien. How does GPT-2 compute greater-than? Interpreting mathematical abilities in a pre-trained language model, 2023.
- [26] Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the MATH dataset, 2021.
- [27] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 1989.
- [28] J. D. Hunter. Matplotlib: A 2D graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007. doi: 10.1109/MCSE.2007.55.
- [29] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.
- [30] Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners, 2023.
- [31] Andrey Kolmogorov and Vladimir Tikhomirov. ε -entropy and ε -capacity of sets in functional spaces. *Uspekhi Matematicheskikh Nauk*, 14(2):3–86, 1959.
- [32] Yann LeCun. A path towards autonomous machine intelligence, 2022.
- [33] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive NLP tasks, 2021.
- [34] Yingcong Li, M. Emrullah Ildiz, Dimitris Papailiopoulos, and Samet Oymak. Transformers as algorithms: Generalization and stability in in-context learning, 2023.
- [35] Zhiyuan Li, Hong Liu, Denny Zhou, and Tengyu Ma. Chain of thought empowers transformers to solve inherently serial problems, 2024.
- [36] Tianyang Lin, Yuxin Wang, Xiangyang Liu, and Xipeng Qiu. A survey of transformers, 2021.
- [37] Yingwei Ma, Yue Liu, Yue Yu, Yuanliang Zhang, Yu Jiang, Changjian Wang, and Shanshan Li. At which training stage does code data help LLMs reasoning?, 2023.
- [38] Eran Malach. Auto-regressive next-token predictors are universal learners, 2023.
- [39] William Merrill and Ashish Sabharwal. The expressive power of transformers with chain of thought, 2024.
- [40] Neel Nanda, Lawrence Chan, Tom Lieberum, Jess Smith, and Jacob Steinhardt. Progress measures for grokking via mechanistic interpretability, 2023.
- [41] Eshaan Nichani, Alex Damian, and Jason D. Lee. How transformers learn causal structure with gradient descent, 2024.
- [42] Maxwell Nye, Anders Johan Andreassen, Guy Gur-Ari, Henryk Michalewski, Jacob Austin, David Bieber, David Dohan, Aitor Lewkowycz, Maarten Bosma, David Luan, Charles Sutton, and Augustus Odena. Show your work: Scratchpads for intermediate computation with language models, 2021.

- [43] 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>.
- [44] Catherine Olsson, Nelson Elhage, Neel Nanda, Nicholas Joseph, Nova DasSarma, Tom Henighan, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, Dawn Drain, Deep Ganguli, Zac Hatfield-Dodds, Danny Hernandez, Scott Johnston, Andy Jones, Jackson Kernion, Liane Lovitt, Kamal Ndousse, Dario Amodei, Tom Brown, Jack Clark, Jared Kaplan, Sam McCandlish, and Chris Olah. In-context learning and induction heads. *Transformer Circuits Thread*, 2022.
- [45] OpenAI et al. GPT-4 technical report, 2024.
- [46] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An imperative style, high-performance deep learning library, 2019.
- [47] Bo Peng, Eric Alcaide, Quentin Anthony, Alon Albalak, Samuel Arcadinho, Stella Biderman, Huanqi Cao, Xin Cheng, Michael Chung, Matteo Grella, Kranthi Kiran GV, Xuzheng He, Haowen Hou, Jiaju Lin, Przemyslaw Kazienko, Jan Kocon, Jiaming Kong, Bartłomiej Koptyra, Hayden Lau, Krishna Sri Ipsit Mantri, Ferdinand Mom, Atsushi Saito, Guangyu Song, Xiangru Tang, Bolun Wang, Johan S. Wind, Stanislaw Wozniak, Ruichong Zhang, Zhenyuan Zhang, Qihang Zhao, Peng Zhou, Qinghua Zhou, Jian Zhu, and Rui-Jie Zhu. RWKV: Reinventing RNNs for the transformer era, 2023.
- [48] Jorge Pérez, Javier Marinković, and Pablo Barceló. On the Turing completeness of modern neural network architectures, 2019.
- [49] Gautam Reddy. The mechanistic basis of data dependence and abrupt learning in an in-context classification task, 2023.
- [50] Elan Rosenfeld and Andrej Risteski. Outliers with opposing signals have an outsized effect on neural network optimization, 2023.
- [51] Clayton Sanford, Daniel Hsu, and Matus Telgarsky. Transformers, parallel computation, and logarithmic depth, 2024.
- [52] Shai Shalev-Shwartz, Ohad Shamir, and Shaked Shammah. Failures of gradient-based deep learning, 2017.
- [53] Steve Smale and Ding-Xuan Zhou. Learning theory estimates via integral operators and their approximations. *Constructive Approximation*, 26(2):153–172, 2007.
- [54] Lena Strobl, William Merrill, Gail Weiss, David Chiang, and Dana Angluin. Transformers as recognizers of formal languages: A survey on expressivity, 2023.
- [55] Vladimir Vapnik. *The Nature of Statistical Learning Theory*. Springer, 1995.
- [56] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023.
- [57] 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, 2022.
- [58] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models, 2023.
- [59] Jingfeng Wu, Peter L. Bartlett, Matus Telgarsky, and Bin Yu. Large stepsize gradient descent for logistic loss: Non-monotonicity of the loss improves optimization efficiency, 2024.
- [60] 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.
- [61] Sang Michael Xie, Aditi Raghunathan, Percy Liang, and Tengyu Ma. An explanation of in-context learning as implicit bayesian inference, 2022.
- [62] Jianyu Zhang, Niklas Nolte, Ranajoy Sadhukhan, Beidi Chen, and Léon Bottou. Memory mosaics, 2024.

[63] Ziqian Zhong, Ziming Liu, Max Tegmark, and Jacob Andreas. The clock and the pizza: Two stories in mechanistic explanation of neural networks, 2023.

Societal Impact. Mechanistic interpretability is focused on understanding the key mechanisms at play in deep learning systems by tracing them down to the weights. It is often associated with AI safety, hoping that a deeper understanding of these systems can help us steer them to be more “aligned” with “human values”, and prevent AI dystopia scenarios. On the other hand, it could also prove useful in training more powerful models, which is associated with significant societal issues linked to the rise of advanced AI systems. These issues are too broad to be discussed in this paragraph.

A Additional Figures & Findings

Figure 11 studies the probability of finding the iteration head when learning with the parity data. As mentioned in the main text, it showcases the bigger probability of learning other circuits when learning the parity problem only.

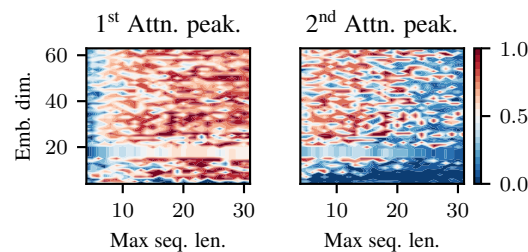


Figure 11: Same figure as Figure 8 yet when learning with the parity dataset. The whitening of the row around $d = 20$ is due to GPU failure, and should not be considered when parsing this figure.

Figure 12 showcases the usefulness of large learning rates and small batch size when using SGD, and the correction brought by Adam.

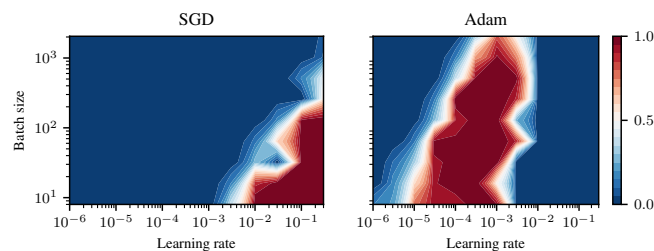


Figure 12: Test accuracy for SGD and Adam after 100 epochs.

Finally, Figure 13 plots some attention maps when learning with a three layers transformer with two attention heads per layer. Knowing the iteration when head circuit, we are able to observe a similar circuit, yet with work shared across heads and layers.

Position embeddings. When we started this project, we were expecting to find some grokking structure emerged from the need to perform position subtraction. In particular, as mentioned in the main text, we were expecting this mechanism to appear as the position embedding dimension was small. When learning on the parity problem, we found that the network was not implementing the iteration head when the position embedding dimension was really small. This can notably be seen on Figures 15 and 14. We notably observed that freezing the position embedding does not change much the picture, which can be seen as a result of overparameterization. Similar type of observation were observed when learning with the polynomial iteration problem, as reported on Figure 16.

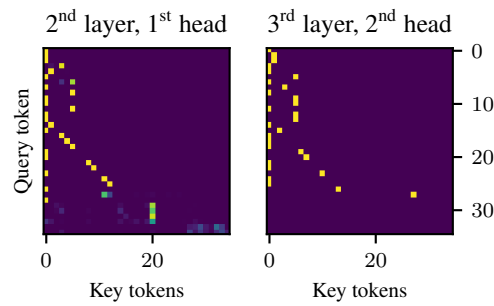


Figure 13: Recovering the “who is p_t ?” key-query association, yet shared across layers and heads when training a three layers transformer with two attention heads per layer.

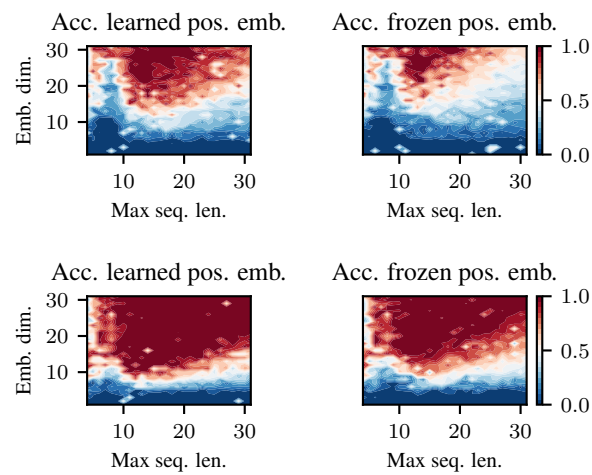


Figure 14: The effect of small embeddings when learning the parity problem. The top row corresponds to what has been learned after 1000 epochs. The bottom one corresponds to 5000 epochs.

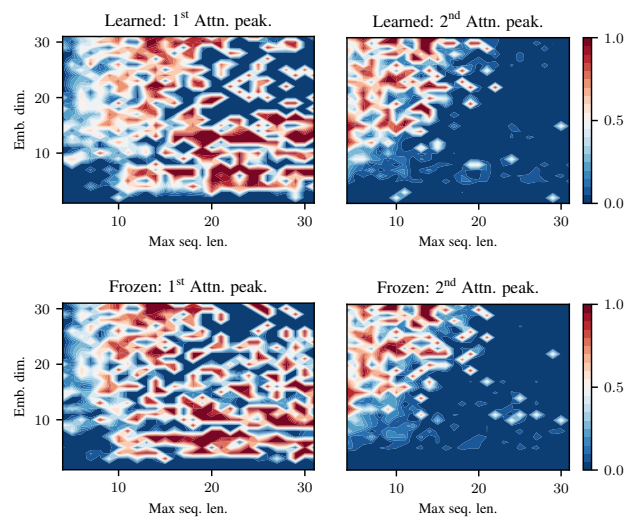


Figure 15: Attention learned when studying frozen vs learned positional embedding.

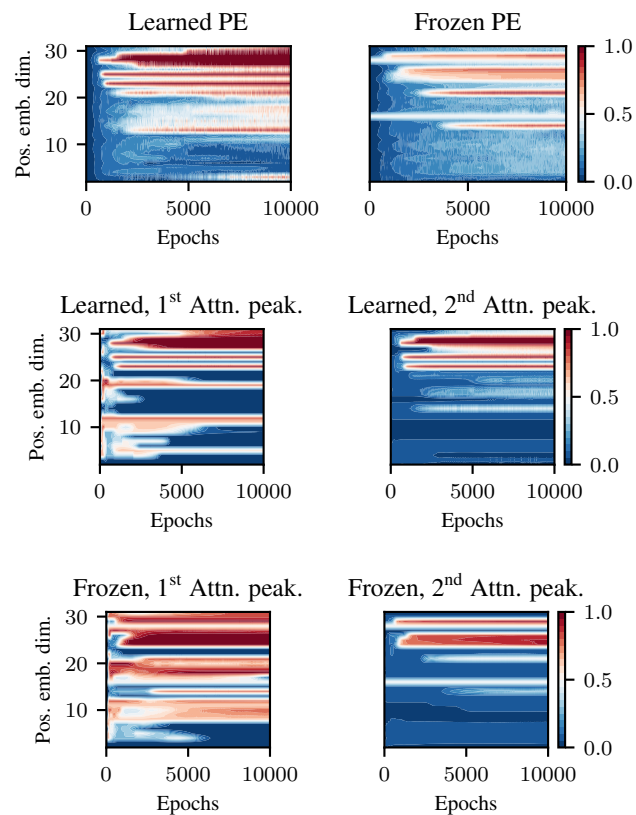


Figure 16: Attention learned when studying frozen vs learned positional embedding. The setting is slightly different, we fixed the token embedding dimension to 32, and added the position embedding only on the first p dimension, where p was varying from 2 to 32.

NeurIPS Paper Checklist

1. Claims

Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

Answer: [\[Yes\]](#)

Justification: We composed the abstract and introduction after establishing the main results.
Guidelines:

- The answer NA means that the abstract and introduction do not include the claims made in the paper.
- The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
- The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
- It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. Limitations

Question: Does the paper discuss the limitations of the work performed by the authors?

Answer: [\[Yes\]](#)

Justification: The limitations of our synthetic data study should be clearly apparent to the reader.

Guidelines:

- The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
- The authors are encouraged to create a separate "Limitations" section in their paper.
- The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
- The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
- The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
- The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
- If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
- While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. Theory Assumptions and Proofs

Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [\[Yes\]](#)

Justification: The results were stated after being demonstrated. Some results that were obvious to us were not demonstrated (assuming one does not have to go back to Peano axiomatic to write additions).

Guidelines:

- The answer NA means that the paper does not include theoretical results.

- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental Result Reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [\[Yes\]](#)

Justification: The codebase contains all the necessary details, and random seeds were set to ensure full reproducibility.

Guidelines:

- The answer NA means that the paper does not include experiments.
- If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general, releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
 - (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
 - (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
 - (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
 - (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [\[Yes\]](#)

Justification: The code is freely available.

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (<https://nips.cc/public/guides/CodeSubmissionPolicy>) for more details.

- While we encourage the release of code and data, we understand that this might not be possible, so “No” is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (<https://nips.cc/public/guides/CodeSubmissionPolicy>) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. Experimental Setting/Details

Question: Does the paper specify all the training and test details (e.g., data splits, hyper-parameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

Answer: [\[Yes\]](#)

Justification: The codebase is available with all the training and testing details. Some details (e.g., pre-norm, embedding dropout rate) were omitted from the main paper to prevent overwhelming the reader with technicalities that do not form the core of the paper.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. Experiment Statistical Significance

Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [\[Yes\]](#)

Justification: Please note that in some 2D plots, “error bars” can be inferred from the texture of the contour lines, which provide a clear sense of the randomness in the process.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The authors should answer “Yes” if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. Experiments Compute Resources

Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

Answer: [Yes]

Justification: We briefly comment on the resources used. The code is open source and the experiments are relatively light. An interested reader could easily calculate a more precise estimate of the energy cost to reproduce the results.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
- The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
- The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. Code Of Ethics

Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics <https://neurips.cc/public/EthicsGuidelines?>

Answer: [Yes]

Justification: Adherence to guidelines was of significant importance to us.

Guidelines:

- The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
- If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. Broader Impacts

Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

Answer: [Yes]

Justification: This work is focused on enhancing our understanding of machine learning systems. While this endeavor comes with clear issues, they are too broad for us to provide any meaningful comments within this context.

Guidelines:

- The answer NA means that there is no societal impact of the work performed.
- If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. Safeguards

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

Answer: [NA]

Justification: This paper was deemed too theoretical to necessitate significant consideration regarding safety concerns.

Guidelines:

- The answer NA means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. Licenses for existing assets

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: We didn't cite everything (for instance, we didn't mention Python or CUDA), but we did cite PyTorch, the transformer paper, as well as GPT-3.

Guidelines:

- The answer NA means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, paperswithcode.com/datasets has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. New Assets

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [NA]

Justification: We have documented our synthetic datasets.

Guidelines:

- The answer NA means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. Crowdsourcing and Research with Human Subjects

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [NA]

Justification: We have not conducted any experiments involving human subjects.

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. Institutional Review Board (IRB) Approvals or Equivalent for Research with Human Subjects

Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [NA]

Justification: See above.

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.