WHY CAN'T TRANSFORMERS LEARN MULTIPLICATION? REVERSE-ENGINEERING IMPLICIT CHAIN-OF-THOUGHT REVEALS CHALLENGES OF LEARNING LONG-RANGE DEPENDENCIES

Anonymous authors
Paper under double-blind review

ABSTRACT

Language models are increasingly capable, yet still fail at a seemingly simple task of multi-digit multiplication. In this work, we study why, by reverse-engineering a model that successfully learns multiplication via *implicit chain-of-thought*, and report three findings: (1) Evidence of long-range structure: Logit attributions and linear probes indicate that the model encodes the necessary long-range dependencies for multi-digit multiplication. (2) Mechanism: the model encodes long-range dependencies using attention to construct a directed acyclic graph to "cache" and "retrieve" pairwise partial products. (3) Geometry: the model implements partial products in attention heads by forming Minkowski sums between pairs of digits, and digits are represented using a Fourier basis, both of which are intuitive and efficient representations that the standard fine-tuning model lacks. With these insights, we revisit the learning dynamics of standard fine-tuning and find that the model converges to a local optimum that lacks the required long-range dependencies. We further validate this understanding by introducing an auxiliary loss that predicts the "running sum" via a linear regression probe, which provides an inductive bias that enables the model to successfully learn multi-digit multiplication. In summary, by reverse-engineering the mechanisms of an implicit chain-of-thought model we uncover a pitfall for learning long-range dependencies in Transformers and provide an example of how the correct inductive bias can address this issue.

1 INTRODUCTION

Large language models demonstrate striking capabilities across reasoning, planning, and tool use. Yet, they also fail on surprisingly simple algorithmic tasks (Nye et al., 2021; Lee et al., 2023). Why do Transformers excel at some tasks, but fail to learn others? One such example is multi-digit multiplication. Despite having *billions* of parameters, models like Llama-3.2 90B or GPT4 still fail at 4x4-digit multiplication (Gambardella et al., 2024),[1] even when explicitly fine-tuned on the task (Yang et al., 2023). Why do Transformers fail to learn multiplication?

We study these questions by contrasting a standard fine-tuned model (SFT), which fails at multiplication, with a model trained with *implicit chain-of-thought* (ICoT) (Deng et al., 2024; 2023), which succeeds. ICoT works by providing explicit chain-of-thought tokens during training, but gradually removes them and thus forces the model to internalize intermediate steps in its latent states.

We partially reverse-engineer the ICoT model and uncover several insights. First, unlike the SFT model, the ICoT model learns the correct long-range structure needed for multi-digit multiplication. We provide evidence of this using logit attributions and linear regression probes. *Mechanistically*, the ICoT model encodes long-range dependencies by organizing its attention into a sparse, binary-tree-like graph, which (i) selects the correct digit pairs to compute partial products and (ii) "caches" these intermediate computations into earlier tokens for later retrieval. Lastly, *geometrically*, attention heads realize partial products as Minkowski sums of digit embeddings, and represent digits

Code: `https://anonymous.4open.science/r/icot-F822`

[1] Note that some recent proprietary models that do solve multi-digit multiplication may rely on tool-use.

with Fourier bases, yielding a pentagonal prism structure – both of which are intuitive and efficient representations that the SFT model lacks.

With these insights, we revisit the dynamics of standard fine-tuning: under gradient descent and an auto-regressive loss, the model never learns these long-range dependencies, and thus loss plateaus on the middle digits. To confirm our understanding, we introduce a simple fix by introducing an auxiliary loss that supervises the model to predict a "running partial sum" through a lightweight linear regression probe. This provides an inductive bias to learn the proper long-range dependencies, allowing it to achieve perfect accuracy, without any supervision from chain-of-thought.

In summary, by partially reverse-engineering a network that successfully implements multi-digit multiplication, we uncover how it implements long-range dependencies, a mechanism that the unsuccessful model lacks. Our work highlights a challenge for Transformers to learn long-range dependency using gradient descent and an auto-regressive loss. While we demonstrate a task-specific inductive bias to address this issue, we anticipate generic improvements to address this limitation.

2 Experiment Setup, Training ICoT, Notations

Task, Models. We are interested in understanding the difference in a model trained with standard fine-tuning and ICoT. From experiments, we find that the simplest multi-digit multiplication in which standard fine-tuning fails but ICoT works is 4×4 digit multiplications. Similarly, the smallest architecture in which ICoT works is a 2-layer model with 4 attention heads. Thus we carefully study a 2-layer 4-head ICoT model and a standard fine-tuned model trained on 4×4 multiplication.

Training Procedures. Our ICoT setup is the same as that Deng et al. (2024). Here we provide an informal overview of ICoT, with details in Appendix A.1. Namely, assume two operands $a = (a_3, a_2, a_1, a_0), b = (b_3, b_2, b_1, b_0)$ and their product $c = (c7 \dots c_0)$. Operands are written least-significant digit first, similar to other algorithmic setups (Deng et al., 2024; 2023; Lee et al., 2023).

For ICoT, the training data includes intermediate chain-of-thought (CoT) tokens q_i that explicitly record the step-by-step calculations. As a simple illustration, consider 12×34. The tokens appearing between the two equal signs follow the same CoT format used in our 4×4-digit multiplication tasks:

$$12 * 34 = \underbrace{48}_{12*4} + \underbrace{360}_{12*30} \underbrace{(408)}_{\text{running sum}} = 408$$

At each training epoch, a fixed number of CoT tokens are removed from the *left* of the chain. Concretely, the training examples at each epoch may have the following form:

$$\begin{aligned}
&\text{(Epoch 1)} \quad a_0a_1a_2a_3 * b_0b_1b_2b_3\%\%\% \; q_0 \dots q_i \dots q_j \dots q_k \dots q_\tau \; \#\#\#\# \; c_0 \dots c_7 \\
&\text{(Epoch 2)} \quad a_0a_1a_2a_3 * b_0b_1b_2b_3\%\%\% \; q_i \dots q_j \dots q_k \dots q_\tau \; \#\#\#\# \; c_0 \dots c_7 \\
&\text{(Epoch 3)} \quad a_0a_1a_2a_3 * b_0b_1b_2b_3\%\%\% \; q_j \dots q_k \dots q_\tau \; \#\#\#\# \; c_0 \dots c_7 \\
&\dots \\
&\text{(Epoch N)} \quad a_0a_1a_2a_3 * b_0b_1b_2b_3\%\%\% \; \#\#\#\# \; c_0 \dots c_7
\end{aligned}$$

where q_i are CoT tokens and $\%, \#$ are special delimiters.[2] Note that after each epoch, the model sees a shorter chain by truncating some tokens, and that by the end, only the operands and final answer remain. For comparison, standard fine-tuning only trains on the operands: $a_0a_1a_2a_3 * b_0b_1b_2b_3\%\%\%\#\#\#\# \; c_0 \dots c_7$.

Interestingly, the ICoT model is able to achieve 100% accuracy on 4×4 digit multiplication, while standard fine-tuning only achieves less than 1% accuracy. Note that scaling does not help – scaling to a 12 layer 8 head model achieves the same $< 1\%$ accuracy, and Yang et al. (2023) show that fine-tuning a 2B model still plateaus at 95% accuracy.

For more details regarding training (data format, sample size, hyperparameters), see Appendix A.

Notations. $\mathbf{h}_t^\ell$ indicates the hidden states at layer ℓ timestep t. Timesteps for solution tokens $c_k, k = [0, \dots, 7]$ are notated t_{c_k}. $\text{ATT}_h^\ell(\cdot)$, $\text{MLP}^\ell(\cdot)$ indicate the output of the attention heads or MLP blocks at layer ℓ, head index h. $E, U \in \mathbb{R}^{V \times d}$ indicate (un)embedding weights.

[2] These delimiters have no special meaning beyond matching the setup of Deng et al. (2024).

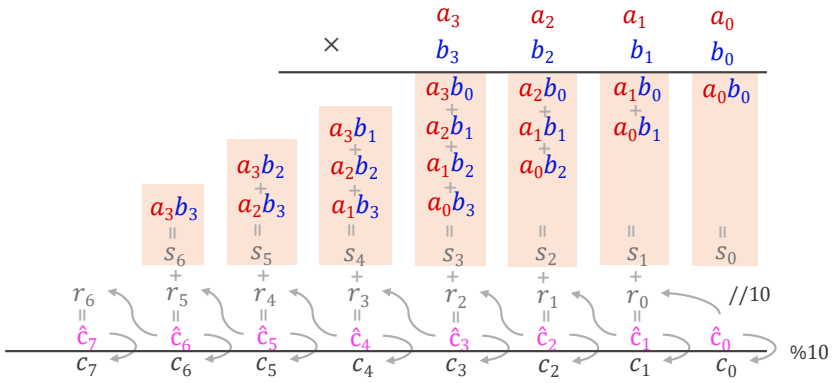

Figure 1: **Multiplication has long-range dependencies**, which can be captured by an intermediate value $\hat{c}_i$, from which both the solution (c_i) and carries (r_i) can be derived from.

3 Comparing the Mechanisms of ICoT versus SFT

3.1 Long-range dependencies in multi-digit multiplication

Here we discuss how one might solve multi-digit multiplication, and the required long-range dependencies needed to solve multiplication.

One approach to compute each digit, c_k, is as follows:

$$s_k \triangleq \underbrace{\sum_{i+j=k} a_i b_j}_{\text{sum of partial products}}, \qquad c_k = (s_k + r_{k-1}) \bmod 10, \qquad r_k = \underbrace{\lfloor \frac{s_k + r_{k-1}}{10} \rfloor}_{\text{carry}}, \qquad r_{-1} = 0 \tag{1}$$

Note that both c_k and r_k can be expressed with an intermediary term $\hat{c}_k$, which encapsulates both the relevant information from the partial products and the carry:

$$\hat{c}_k \triangleq s_k + r_{k-1}, \qquad c_k = \hat{c}_k \pmod{10}, \qquad r_k = \lfloor \frac{\hat{c}_k}{10} \rfloor \tag{2}$$

Importantly, note the *long-range dependencies* needed for multi-digit multiplication. Specifically, we highlight two observations: **(i)** To determine c_k, one must use all the partial products $\{a_i b_j | i + j \leq k\}$, since all of these terms contribute to c_k. **(ii)** Knowing the intermediary term $\hat{c}_k$ suffices to compute c_k and to propagate necessary information for later digits. Thus we use $\hat{c}_k$ as a probing signature (Section 3.2) at each timestep t_{c_k} to check if the model is utilizing all the necessary long-range information to predict the correct tokens c_k.

In the following sections, we demonstrate how the ICoT model satisfies such long-range dependency while the standard fine-tuning model does not.

3.2 Evidence of Long-Range Dependencies in ICoT

We first demonstrate two lines of evidence that the ICoT model satisfies long-range dependencies in multi-digit multiplication, while the standard fine-tuning SFT model does not.

Logit Attributions. Note from Figure 1 that digits a_i, b_i can only affect c_k terms where $k \geq i$. Also note that at timestep t_{c_k}, the pairwise products $\{a_i b_j | i + j = k\}$ affect the final prediction c_k the most. "Earlier" pairwise products $\{a_i b_j | i + j = k - m\}$ can still affect c_k, but with diminishing effects as m increases.

We directly test for these relationships in our ICoT and SFT models using logit attributions. Namely, given an input sample ORIG $:= a_0a_1a_2a_3 * b_0b_1b_2b_3$, we measure the logits of the model's predictions for c_{0-7} : $\text{logit}_{c_k}(\text{ORIG})$. We then randomly swap out one of the operand digits at timestep t (e.g., $\tilde{a}_2$) to construct a counterfactual input $\text{COUNTER}_t = a_0a_1\tilde{a}_2a_3 * b_0b_1b_2b_3$ and measure the change in logits: $\Delta_{t,k} = \text{logit}_{c_k}(\text{ORIG}) - \text{logit}_{c_k}(\text{COUNTER}_t)$ Thus $\Delta_{t,k}$ measures the effect that digit at timestep t has on the prediction of token c_k.

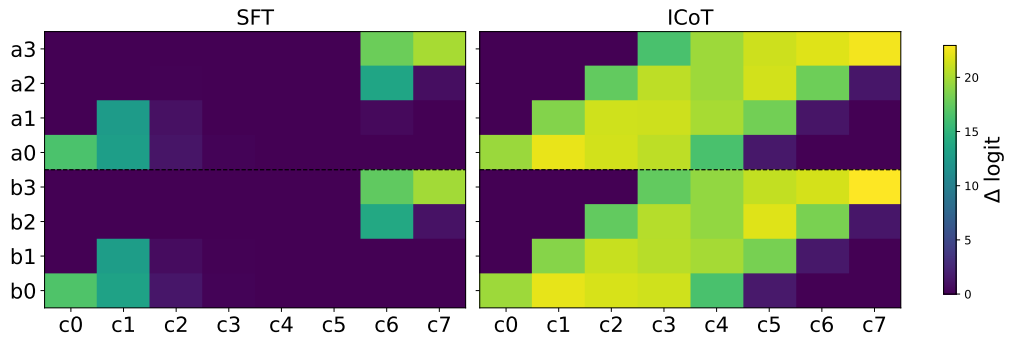

Figure 2: **Logit Attribution.** We test for whether each model has correctly learned long-range dependencies by measuring how sensitive the logits of output digits c_i are to each operand digit (i.e., a_i, b_j). This is done by measuring the change in c_i's logits when a single operand digit is perturbed.

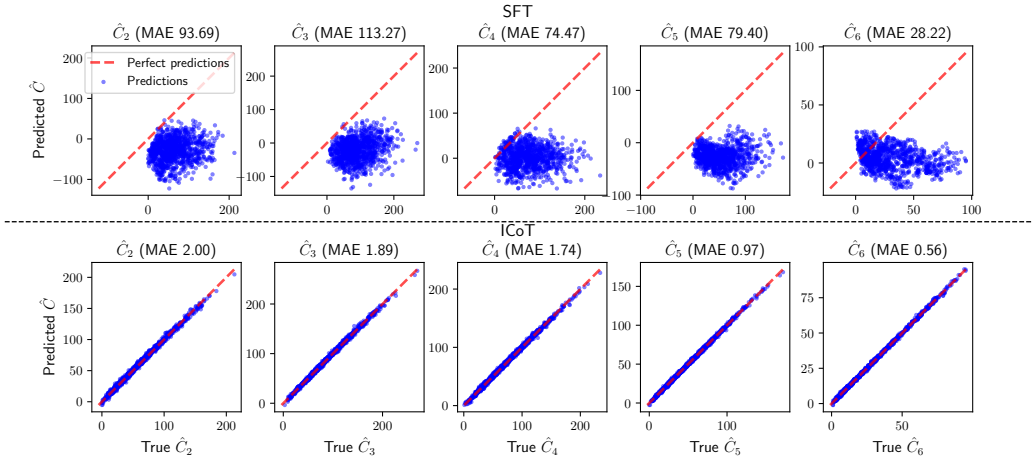

Figure 3: **Linear regression probing results for $\hat{c}$.** We probe from the middle of the last Transformer block, after attention heads but before MLPs.

We use 1,000 samples for each (t, k) pair and show the results in Figure 2. Note that for SFT, the model does not see the correct dependencies between earlier tokens to middle tokens, while the ICoT model does, suggesting that the model has indeed learned the correct long-range dependencies.

Probing for $\hat{\mathbf{c}}_k$. Note from Figure 1 and Equation 2 that the long-range dependencies can be captured by an intermediate term, $\hat{c}_k$. We test for whether $\hat{c}_k$ information can be decoded from the hidden states of the models using linear regression probes. Namely, at each timestep t_{c_k} we predict for $\hat{c}_k$ by training a single vector $\mathbf{w}_k \in \mathbb{R}^d$ such that $\mathbf{w}_k \mathbf{h}^{2.\text{mid}}_{t_{ck}} = \hat{c}_k$ using a MSE loss, where $\mathbf{h}^{2.\text{mid}}$ is the hidden state at layer 2 after attention heads, before MLPs.

Figure 3 reports the mean absolute error from probing for $\hat{c}_k$ for middle and late digits, $k = 2, \ldots, 6$. Note that the accuracy from the ICoT model is much higher than that of SFT, further suggesting that the ICoT model has learned the correct long-range dependencies while SFT has not.

3.3 Encoding Long-Range Dependencies via Attention Trees

How does the ICoT model compute long-range dependencies? Here we describe how the model's attention patterns induce a shallow directed acyclic graph, akin to a binary expression tree, in order to encode long-range dependencies.

Namely, in the first layer, across all timesteps $t > 5$,[3] each attention head only attends to a *pair* of digit tokens, $\{a_i, b_j\}$ (Figure 4, left). This allows the model to produce the pairwise product $a_i b_j$ (see Section 4.1 for *how* attention heads represent pairwise products), but also allows the model to

[3]Note that only after timestep 5, both a and b tokens appear in the context.

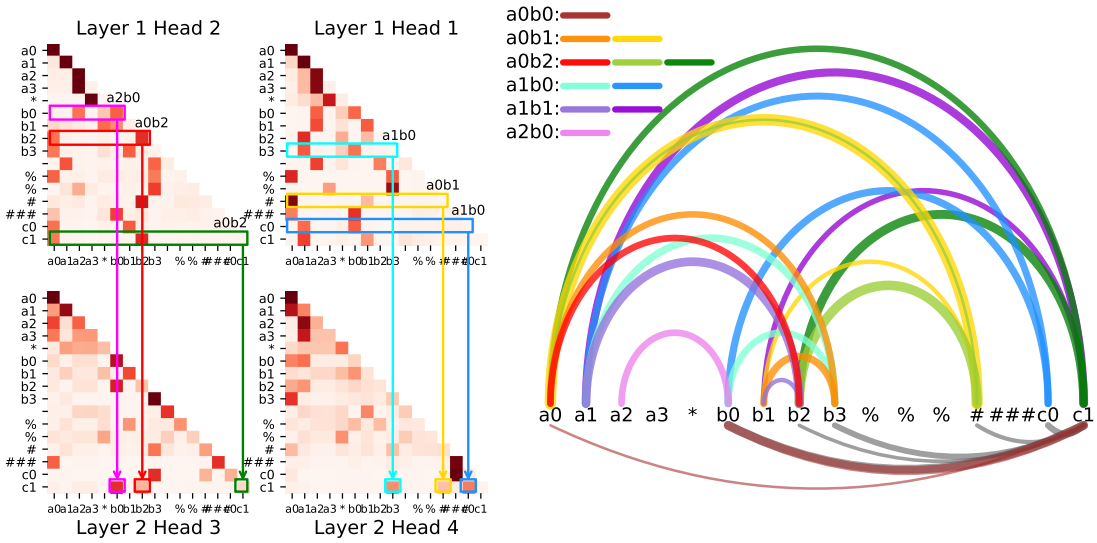

Figure 4: **Visualization of attention tree to compute c_2.** Left: Attention maps for selected heads show the first layer "cache" pairwise products ($a_i b_j$) across earlier timesteps, from which the second layer reads from (Not all tree paths are shown). Right: A visualization of the attention tree. Each arc indicates tokens being attended to at specific timesteps. Colored arcs above and below the digits indicate attention patterns from the first and second layers respectively. Example: orange arc indicates that at timestep b_3, the model attends to a_0 and b_1, from which the second layer reads from.

cache the product $a_i b_j$ in the hidden state of layer 1 at timestep t (i.e., $\mathbf{h}_t^1$). Put differently, product pairs $\{a_i b_j\}_{i,j \in \{0,\ldots 4\}}$ are "cached" in the first layer across different timesteps ($\mathbf{h}_t^1, t < t_{c_k}$).

At later timesteps $t \geq t_{c_k}$, when the model predicts solution tokens c_k, this allows the second layer attention heads to attend to a small set of previous *cache sites*, i.e., where the appropriate pairs of products $a_i b_j, i + j = k$ are stored from earlier timesteps.

Example: Figure 4 depicts the attention patterns when the model predicts c_2, given input "$a_{0\ldots3} * b_{0\ldots3} = c_0 c_1$". These attention maps are averaged from 1,000 samples from a held out test set. The necessary terms to compute c_2 are $a_2 b_0, a_1 b_1, a_0 b_2$, and $\hat{c}_1$ (which in turn requires $a_1 b_0, a_0 b_1, a_0 b_0$).

Attention heads ATT_3^2, ATT_4^2 each attend to positions (b_0, b_2, c_1) and $(b_3, \text{"\#"}, c_0)$. Inspecting what was "cached" in the first layer at those timesteps reveals the necessary partial products to compute c_2. For example, at timestep b_0, ATT_1^1, ATT_2^1 attend to a_2, b_0; at timestep b_2 ATT_1^1 attends to a_1, b_1 while ATT_2^1 attends to a_0, b_2; at timestep c_0 ATT_1^1 attends to a_1, b_0, ATT_2^1 attends to $a_0 b_1$. Thus the model can derive partial products, $a_2 b_0, a_1 b_1, a_0 b_2, a_1 b_0, a_0 b_1$ with its attention tree.[4]

While Figure 4 shows an example of the "attention tree" for predicting c_2, one can similarly reconstruct the correct trees for all digits $c_0, \ldots, c_7$ using the attention patterns for all digits in Figure 10.

In summary, for each output step c_k, the ICoT model constructs a binary-tree-like graph, spread out across timesteps, to attend to the correct pairs of tokens, allowing it to compute partial products.

4 Feature Geometry of ICoT

In addition to the mechanisms seen in Section 3, we also study the *geometry* of features in ICoT.

4.1 Digit-wise Multiplications as Minkowski Sums

Note from Section 3.3 that the attention patterns are sparse, often only attending to the two digits a_i, b_j being multiplied. In such a case, the outputs of the attention head form a Minkowski sum.

[4]Note that there may be a couple of different ways that $a_0 b_0$ is derived. One possibility is to re-use a_0, b_0 information that was fetched at various timesteps. Another possibility is when a_0 is slightly attended to at ATT_3^2 (difficult to see in our visuals). Note that $a_0 b_0$ plays a relatively minor role in computing c_2 compared to all other partial products.

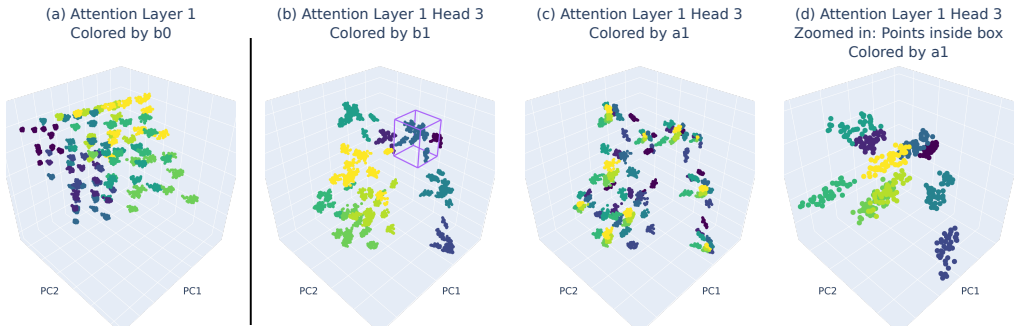

Figure 5: **3D PCA of attention head outputs can form Minkowski sums**, which in turn can form nested representations. Each color represents a different digit.

Namely, consider a single head $\text{ATT}^1(i, j)$ at the first layer, attending to two digits a_i, b_j. Let $W_O \in \mathbb{R}^{d \times d_{head}}, W_V \in \mathbb{R}^{d_{head} \times d}$ be the output and value weights of the attention head, $E[a_i] \in \mathbb{R}^d$ the token embedding for token a_i, and $A_i := W_O W_V E[a_i], B_j := W_O W_V E[b_j], A_i, B_j \in \mathbb{R}^d$.

In such a case, when the model spends $\alpha\%$ of its attention on digit a_i, and thus attends to digit b_j by $(1-\alpha)\%$, the set of all possible values for the output of the attention head forms a Minkowski sum:

$$\text{ATT}^1(i, j) = \alpha A_i + (1-\alpha)B_j + \epsilon \tag{3}$$

$$\{\text{ATT}^1(i, j)\}_{i,j} \subseteq (\alpha A) \oplus ((1-\alpha)B) \oplus \epsilon \tag{4}$$

(ignoring position embeddings). See Figure 5 (a) for a visualization.

Visually, 3D PCAs can reveal nested representations. Namely, we can observe clusters, each cluster corresponding to a feature (i.e., a_i). These clusters form a "global" geometry. When zoomed in to each cluster, we observe additional clusters for a second feature (i.e., b_j) that form a "local" geometry of the same shape as its global counterpart. See Figure 5 (b-d) for examples.

This observation can be explained by deconstructing the covariance of the attention output:

$$\Sigma_{\text{ATT}} = \alpha^2 \Sigma_A + (1-\alpha)^2 \Sigma_B, \tag{5}$$

where $\Sigma_A = \text{Cov}(A_i), \Sigma_B = \text{Cov}(B_j)$. First, note that if we ignore positional encodings, Σ_A and Σ_B share the same eigenvectors, as they each depend on the same terms $(E[\cdot], W_O, W_V)$, which are picked by PCA. Further note that fixing a value for a_i leaves a local covariance, $\Sigma_{local|a_i} = (1-\alpha)^2 \Sigma_B$, which again share the same eigenvectors with the global Σ_{ATT} term, leading to the same local geometry when projected onto.

4.2 Embedding Digits on a Pentagonal Prism via Fourier Bases

Similar to Kantamneni & Tegmark (2025), we find that our model encodes digits in Fourier space. Specifically, the model's embeddings E, the final hidden layer $\mathbf{h}^L$, and even the weights of the last MLP can be well reconstructed from a small set of Fourier basis functions.

Figure 6 shows a 3D PCA visualization of the final hidden layer at timestep t_{c_2}, for both the SFT and ICoT models. While the SFT hidden states do not reveal any obvious patterns, the ICoT hidden states reveal a striking pattern: the ten digits form vertices of a *pentagonal prism*.

This structure is naturally explained by Fourier modes. Consider the Fourier expansion

$$\sum C_n * e^{-2\pi i \frac{kn}{10}}, \quad n = 0, \ldots, 9.$$

where $C_n (\neq c_k)$ is some constant per digit n. Following Kantamneni & Tegmark (2025), we take frequencies $k \in \{0, 1, 2, 5\}$, yielding the real Fourier basis

$$\Phi(n) = \left[\underset{(k=0)}{\mathbf{1}(n)} \quad \underset{(k=1)}{\cos\left(2\pi \tfrac{n}{10}\right)} \quad \underset{(k=1)}{\sin\left(2\pi \tfrac{n}{10}\right)} \quad \underset{(k=2)}{\cos\left(2\pi \tfrac{n}{5}\right)} \quad \underset{(k=2)}{\sin\left(2\pi \tfrac{n}{5}\right)} \quad \underset{(k=5)}{\boldsymbol{p}(n)} \right],$$

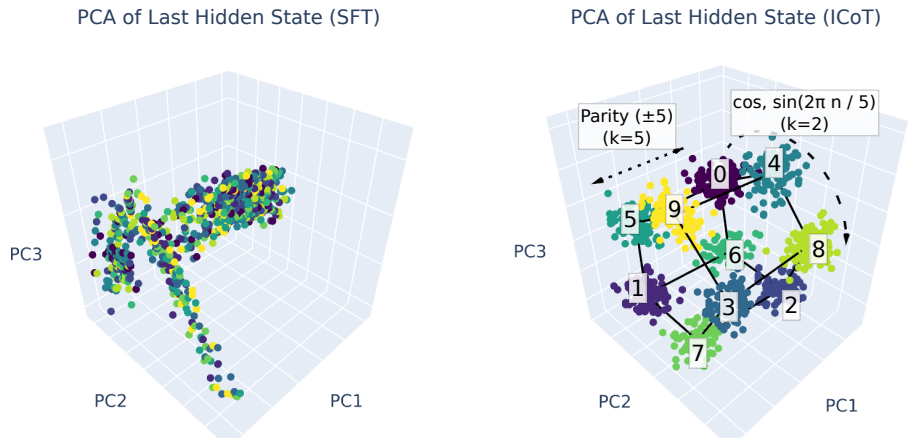

Figure 6: **Digits embedded in a pentagonal prism, using Fourier bases.** No obvious patterns in the SFT model, but the ICoT model encodes digits in a pentagonal prism using Fourier bases.

where $\mathbf{1}(n) \equiv 1$ (the DC component) and $\boldsymbol{p}(n) \equiv (-1)^n$ (the Nyquist/parity vector). The sine terms for $k = 0$ and $k = 5$ vanish over $n = 0, \ldots, 9$ and are omitted.

The final hidden layer $\mathbf{h}^L$ can be reconstructed via these six terms (see Appendix B), indicating that the final hidden state is encoded using Fourier bases.

Revisiting Figure 6, the first principal component (PC1) aligns with the parity vector $\boldsymbol{p}(n)$, separating even from odd digits. Second and third principal components span the $k = 2$ Fourier pair $(\cos, \sin(\frac{2\pi n}{5}))$, so the digits lie on two regular pentagons: one each for even and odd digits. The digits within each pentagon advance by $n+4 \pmod{10}$ (e.g., $n = 0 \to 4 \to 8 \ldots$, same for odd digits), allowing a walk around the pentagon while staying within the even/odd set. Interestingly, taking $\pmod 5$ on such a sequence yields decreasing steps of 1 ($n \pmod 5 = 0 \to 4 \to 3 \ldots$). Lastly, the two pentagons are parallel and stacked along PC1, with corresponding vertices differing by ± 5 (same phase, opposite parity). Together, these yield the pentagonal-prism geometry in Figure 6.

5 Pitfalls of Learning: Lack of Long-Range Dependency

Given our insights, we revisit why Transformers fail at multiplication under standard fine-tuning.

In particular, in Figure 7 (a), we inspect the gradient norms (top row) and losses (bottom row) *per token* c_k over the course of training. There are a few observations to make.

First, note from the loss curves that the first two digits, c_0, c_1, followed by the last digit, c_7, are learned first, as indicated by their immediate drop in loss to near zero. This aligns with the gradient norms observed for these tokens: within the first few steps, these tokens receive gradients, but their norms quickly drop to near zero once the loss for these tokens reach zero. Also note that the order in which tokens are learned according to gradient norms and losses is consistent.

The model then eventually learns to predict c_2. However, middle digits, c_3 to c_6 are never learned. Despite only the middle digits receiving gradients (as they are the only sources of loss remaining), their losses plateau, suggesting that the model is stuck in a local optimum that lacks the long-range dependencies to properly learn the middle digits.

Note that scaling to a larger model does not address this issue, as the same pattern can be found in a 12 layer 8 head model: see Appendix C.

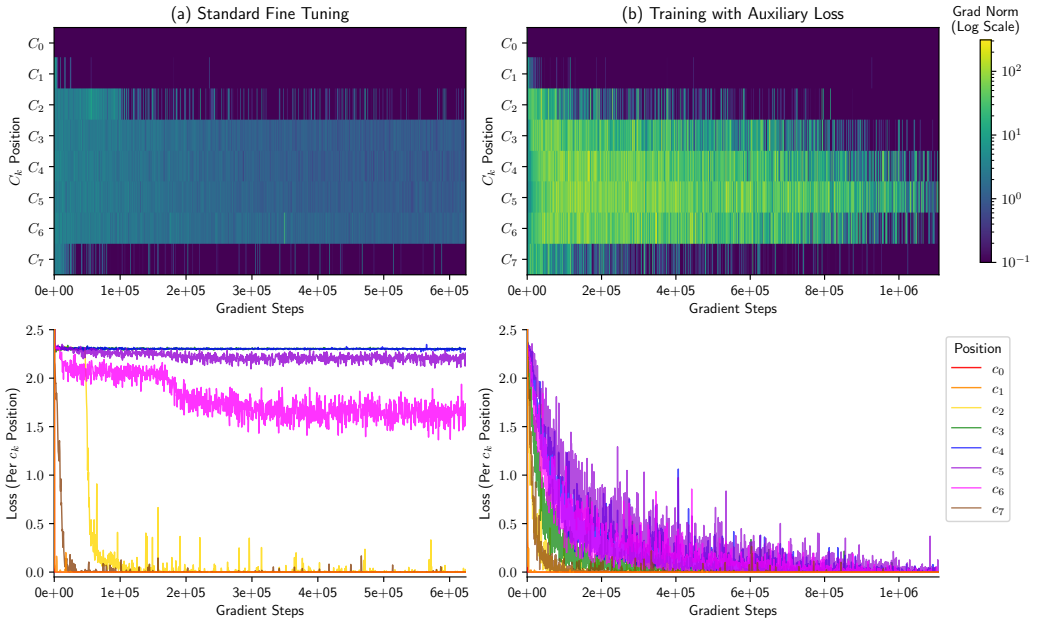

Figure 7: **Gradient norms and losses *per token* c_k.** While both methods learn digits c_0, c_1, c_7 first, standard fine-tuning gets stuck in a local optimum without having learned the right long-range dependencies, while training with the auxiliary loss allows the model to learn the middle digits.

6 Learning Multiplication Without ICoT

To further validate our understanding of why Transformers fail to learn multiplication, we demonstrate an example of a simple fix to teach Transformers multiplications without needing ICoT.

In particular, we leverage the observation from Section 3.1 in that (i) multi-digit multiplication requires long-range dependencies between digit c_k and pairwise products $\{a_i b_j | i + j \leq k\}$, and (ii) such dependency can be summarized by an intermediate value $\hat{c}_k$ to produce c_k.

Thus in order to guide the Transformer to learn long-range dependencies, we simply add an auxiliary loss term to predict $\hat{c}_k$ at each timestep t_{c_k}. We attach an additional linear regression head $\mathbf{w}_h \in \mathbb{R}^d$ to the output of $H(=2)$ attention heads in the second layer. These regression heads are trained to predict the correct accumulated sum $\hat{c}_k$ at each timestep $t_{c_k}, c_k \in [0, \dots 7]$ with a MSE loss:

$$z_i^h = \mathbf{w}_h^\top \text{ATT}_h^2(\cdot) \tag{6}$$

$$\mathcal{L}_{aux} = \frac{1}{H} \sum_{h \in H} \frac{1}{8} \sum_{i=0}^{7} (z_i^h - \hat{c}_i)^2 \tag{7}$$

$$\mathcal{L} = \mathcal{L}_{LM} + \lambda \mathcal{L}_{aux} \tag{8}$$

where $\mathcal{L}_{LM}$ is the standard language modeling loss.

This introduces an inductive bias for the task, and allows our 2-layer model to correctly learn 4x4 multiplication with 99% accuracy. Again, note that a larger 12-layer model still fails at multiplication under standard fine-tuning.

Revisiting Figure 7 (b) demonstrates a very different learning dynamic. We observe the model learn early and last digits (c_0, c_1, c_7) and work inwards (c_2, c_3, c_4, c_6, and finally c_5).

Limitation. Obviously the suggested inductive bias pertains specifically to our task. However, our experiments demonstrate the pitfall of Transformers that require long-range dependencies, and that it is possible to overcome such a pitfall with the correct inductive biases. We speculate that there are other generalizing inductive biases that can improve performance on tasks with long-range dependencies (Tay et al., 2020), and leave this for future work.

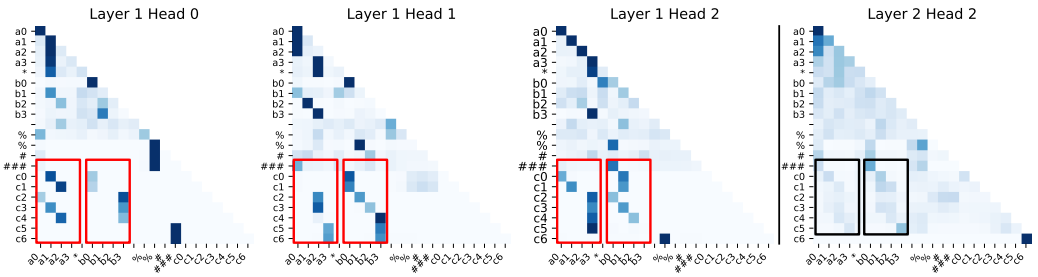

Figure 8: **Attention pattern of model trained with auxiliary loss.** This model similarly produces a "binary attention tree" (red boxes), but interestingly, we also see an attention head that attends to all necessary pairwise digits simultaneously (black box), producing a pattern akin to Figure 2.

6.1 Does the model with auxiliary loss learn the same mechanism as ICoT?

A natural question that arises is whether ICoT and our inductive bias leads to the same mechanisms.

Inspecting the attention patterns suggests that a similar (but not necessarily exact) mechanism is learned: see Figure 8. Namely, the model similarly forms an "attention tree" to sparsely attend to the correct pairs of digits for each c_i in the first layer (red boxes). Interestingly, in the auxiliary-loss model we also observe an attention head (Layer 2 Head 2) that simultaneously attends to all the necessary digits, $\{a_{i\leq k}, b_{i\leq k}\}$, at each timestep t_{c_k}, forming a parallelogram-like attention pattern (black box) akin to the shape seen in Figure 2.

7 Related Work

Studying Transformers with Arithmetic Tasks. A growing line of work study Transformers under controlled settings to better characterize their behavior (Allen-Zhu & Li, 2023a;b; Li et al., 2023; Nanda et al., 2023b; Park et al., 2024b;a). Often, arithmetics is a natural and popular domain (Lee et al., 2023; Ye et al., 2024; Nikankin et al., 2024), which has led to numerous insights. For instance, Nanda et al. (2023a) study how Transformers perform modular addition to explain grokking. Kantamneni & Tegmark (2025) find that large language models use trigonometry to do addition, encoding digits using Fourier bases, while Nikankin et al. (2024) suggest that they also rely on heuristics. Cai et al. (2025) study length generalization in Transformers using arithmetic tasks. Similarly, we study the limitations of Transformers by studying why it fails to learn multi-digit multiplication.

Process Supervision. Recent work trains models with *process supervision*, in which feedback is given not just on final correctness but on each intermediate reasoning step. For example, Uesato et al. (2022) demonstrate that process-supervision can yield less reasoning errors on GSM8K compared to outcome-only supervision. Similarly, Lightman et al. (2023) show that step-level human feedback on MATH leads to stronger reward models. More recently, Zhong et al. (2023)'s Math-Shepherd automates step-wise rewards via continuation-based verification, improving performance on both GSM8K and MATH. ICoT similarly plays the role of process supervision in latent space, by slowly removing chain-of-thought tokens during training such that the model internalizes the reasoning procedure. We thus use ICoT's success on multiplication to study why Transformers fail.

8 Conclusion

In this work, we study why Transformers fail on a seemingly simple task of multi-digit multiplication. We answer this question by reverse-engineering a model trained with implicit chain-of-thought, and uncover that it has learned to compute the correct long-range dependencies needed for multi-digit multiplication. Our findings point to a pitfall of the standard recipe for training language models: using gradient descent with an auto-regressive loss on Transformers. While we provide a simple example of how the right inductive bias can address such a limitation, we anticipate future work to provide a generic solution to improve on tasks with long-range dependencies.

9 Reproducibility Statement

Our code to reproduce all of our experiments can be found in `https://anonymous.4open.science/r/icot-F822/`. Appendix A provides details of our training setup, including data formats, sample size, and hyperparameters.

References

Zeyuan Allen-Zhu and Yuanzhi Li. Physics of language models: Part 1, context-free grammar. *arXiv e-prints*, pp. arXiv–2305, 2023a.

Zeyuan Allen-Zhu and Yuanzhi Li. Physics of language models: Part 3.1, knowledge storage and extraction. *arXiv preprint arXiv:2309.14316*, 2023b.

Ziyang Cai, Nayoung Lee, Avi Schwarzschild, Samet Oymak, and Dimitris Papailiopoulos. Extrapolation by association: Length generalization transfer in transformers. *arXiv preprint arXiv:2506.09251*, 2025.

Yuntian Deng, Kiran Prasad, Roland Fernandez, Paul Smolensky, Vishrav Chaudhary, and Stuart Shieber. Implicit chain of thought reasoning via knowledge distillation. *arXiv preprint arXiv:2311.01460*, 2023.

Yuntian Deng, Yejin Choi, and Stuart Shieber. From explicit cot to implicit cot: Learning to internalize cot step by step, 2024. URL `https://arxiv.org/abs/2405.14838`.

Andrew Gambardella, Yusuke Iwasawa, and Yutaka Matsuo. Language models do hard arithmetic tasks easily and hardly do easy arithmetic tasks. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 85–91, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-short.8. URL `https://aclanthology.org/2024.acl-short.8/`.

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

Nayoung Lee, Kartik Sreenivasan, Jason D Lee, Kangwook Lee, and Dimitris Papailiopoulos. Teaching arithmetic to small transformers. *arXiv preprint arXiv:2307.03381*, 2023.

Kenneth Li, Aspen K Hopkins, David Bau, Fernanda Viégas, Hanspeter Pfister, and Martin Wattenberg. Emergent world representations: Exploring a sequence model trained on a synthetic task. *ICLR*, 2023.

Alex Lightman, Yuntao Bai, Saurav Kadavath, Tamera Lanham, Nicholas Schiefer, et al. Let's verify step by step, 2023. URL `https://arxiv.org/abs/2305.20050`.

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

Neel Nanda, Andrew Lee, and Martin Wattenberg. Emergent linear representations in world models of self-supervised sequence models. *arXiv preprint arXiv:2309.00941*, 2023b.

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

Maxwell Nye, Anders Johan Andreassen, Guy Gur-Ari, Henryk Michalewski, Jacob Austin, David Bieber, David Dohan, Aitor Lewkowycz, Maarten Bosma, David Luan, et al. Show your work: Scratchpads for intermediate computation with language models. 2021.

Core Francisco Park, Andrew Lee, Ekdeep Singh Lubana, Yongyi Yang, Maya Okawa, Kento Nishi, Martin Wattenberg, and Hidenori Tanaka. Iclr: In-context learning of representations. *arXiv preprint arXiv:2501.00070*, 2024a.

Core Francisco Park, Ekdeep Singh Lubana, Itamar Pres, and Hidenori Tanaka. Competition dynamics shape algorithmic phases of in-context learning. *arXiv preprint arXiv:2412.01003*, 2024b.

Yi Tay, Mostafa Dehghani, Samira Abnar, Yikang Shen, Dara Bahri, Philip Pham, Jinfeng Rao, Liu Yang, Sebastian Ruder, and Donald Metzler. Long range arena: A benchmark for efficient transformers. *arXiv preprint arXiv:2011.04006*, 2020.

Jonathan Uesato, Po-Sen Huang, Tim Rocktäschel, Pushmeet Kohli, et al. Solving math word problems with process- and outcome-based feedback, 2022. URL `https://arxiv.org/abs/2211.14275`.

Zhen Yang, Ming Ding, Qingsong Lv, Zhihuan Jiang, Zehai He, Yuyi Guo, Jinfeng Bai, and Jie Tang. Gpt can solve mathematical problems without a calculator. *arXiv preprint arXiv:2309.03241*, 2023.

Tian Ye, Zicheng Xu, Yuanzhi Li, and Zeyuan Allen-Zhu. Physics of language models: Part 2.1, grade-school math and the hidden reasoning process. *arXiv preprint arXiv:2407.20311*, 2024.

Zexue Zhong, Zihan Zhao, Yutao Sun, et al. Math-shepherd: Process supervision for large language models, 2023. URL `https://arxiv.org/abs/2312.08935`.

A Training Details

Here we provide additional details regarding the training procedures of each of our models.

A.1 ICoT Training

Our ICoT training setup follows the practice outlined in the original ICoT paper (Deng et al., 2024).

ICoT works by initially presenting explicit chain-of-thought tokens during training, but gradually removing them across numerous "stages" (e.g., epochs). Concretely, the training examples at each epoch may have the following form:

$$\begin{aligned}
&\text{(Epoch 1)} \quad a_0a_1a_2a_3 * b_0b_1b_2b_3\%\%\% \; q_0 \dots q_i \dots q_j \dots q_k \dots q_\tau \; \#\#\#\# \; c_0 \dots c_7 \\
&\text{(Epoch 2)} \quad a_0a_1a_2a_3 * b_0b_1b_2b_3\%\%\% \; q_i \dots q_j \dots q_k \dots q_\tau \; \#\#\#\# \; c_0 \dots c_7 \\
&\text{(Epoch 3)} \quad a_0a_1a_2a_3 * b_0b_1b_2b_3\%\%\% \; q_j \dots q_k \dots q_\tau \; \#\#\#\# \; c_0 \dots c_7 \\
&\dots \\
&\text{(Epoch N)} \quad a_0a_1a_2a_3 * b_0b_1b_2b_3\%\%\% \; \#\#\#\# \; c_0 \dots c_7
\end{aligned}$$

where q_i are CoT tokens and $\%, \#$ are special delimiters. These delimiters have no special meaning beyond matching the setup of Deng et al. (2024). Note that after each epoch, the model sees a shorter chain by truncating some tokens, and that by the end, only the operands and final answer remain.

The actual format of our ICoT data is as follows. Using an example input of 8331×5015, digits are presented in least-significant digits first, resulting in the following format:

$$1338 * 5105||5614 + 013380(569421) + 0000000(5694210) + 0005561\%\%\#\#\#\#56997714$$

Unlike Deng et al. (2024), instead of using a pre-trained 12-layer GPT model, we train a smaller 2-layer, 4-head GPT-based model from scratch, not only to remove any confounding factors from pre-trained knowledge, but also because the 2-layer 4-head architecture is the simplest form in which ICoT succeeds but standard fine-tuning fails. The training data consists of 80,800 samples, while the validation and test sets each contain 1,000 held out samples. We train with a learning rate of `5e-5`, and remove 8 chain-of-thought tokens at every "stage" (which in our case is an epoch). Both training and validation loss converge after 13 epochs, and achieves 100% accuracy on the test set.

A.2 Standard Fine-Tuning

Similar to ICoT, for our standard fine-tuning model, we train a 2-layer, 4-head GPT-based model from scratch, on the same data as ICoT. We use a learning rate of `5e-5`, and the input format

is $a_0a_1a_2a_3 * b_0b_1b_2b_3\%\%\#\#\#\#c_0 \dots c_7$. All other hyperparameters match those in our ICoT setup. The model's loss and accuracy plateaus after 13 epochs, it achieves only about 1% train and validation accuracy, while digit-level accuracy converges at approximately 81%, and remains the same even after 60 epochs.

Note that scaling the model larger to a 12-layer, 8-head model achieves the same low accuracy at 1% and digit-level accuracy of 80%.

B FOURIER STRUCTURE IN MODEL'S WEIGHTS, ACTIVATIONS

Here we provide a deeper dive into the Fourier structure found in the ICoT model's weights and hidden states. Namely, we analyze the model's embedding weights, final MLP's weights, and last hidden layer:

1. Embeddings $\mathcal{E} \in \mathbb{R}^{10\times d}$
2. Final layer MLP output weights $W_{out} \in \mathbb{R}^{d_{mlp}\times d}$, given $\text{MLP}(\mathbf{x}) = \sigma(W_{in}\mathbf{x})W_{out}$
3. Final hidden layer $\mathbf{h}_t^L \in \mathbb{R}^{N\times d}$

where $N(= 1,000)$ is the size of our validation set. For the latter two, we first project them onto the model's embedding space:

$$\widehat{W_{out}} = (\mathcal{E}W_{out})^\top \in \mathbb{R}^{d_{mlp}\times 10}$$
$$\widehat{\mathbf{h}^L} = (\mathcal{E}\mathbf{h}^L)^\top \in \mathbb{R}^{N\times 10}$$

Each item $X \in \{\mathcal{E}, \widehat{W_{out}}, \widehat{\mathbf{h}^L}\}$ is a collection of row vectors $\mathbf{x} \in \mathbb{R}^{10}$ whose ten entries correspond to digits $n = 0, \dots, 9$.

We find that vectors $\mathbf{x}$ are encoded in a low-dimensional trigonometric subspace.

Namely, consider the Fourier expansion

$$\sum C_n * e^{-2\pi i \frac{kn}{10}}, \quad n = 0, \dots, 9.$$

where $C_n(\neq c_k)$ is some constant per n. Following Kantamneni & Tegmark (2025), we take frequencies $k \in \{0, 1, 2, 5\}$, yielding the real Fourier basis

$$\Phi(n) = \left[\underset{(k=0)}{\mathbf{1}(n)} \quad \underset{(k=1)}{\cos\!\big(2\pi\tfrac{n}{10}\big)} \quad \underset{(k=1)}{\sin\!\big(2\pi\tfrac{n}{10}\big)} \quad \underset{(k=2)}{\cos\!\big(2\pi\tfrac{n}{5}\big)} \quad \underset{(k=2)}{\sin\!\big(2\pi\tfrac{n}{5}\big)} \quad \underset{(k=5)}{\boldsymbol{p}(n)} \right],$$

where $\mathbf{1}(n) \equiv 1$ (the DC component) and $\boldsymbol{p}(n) \equiv (-1)^n$ (the Nyquist/parity vector). The sine terms for $k = 0$ and $k = 5$ vanish over $n = 0, \dots, 9$ and are omitted.

Let $F \in 10 \times 6$ be a Fourier matrix with rows indexed by $n \in \{0, \dots, 9\}$ and columns as defined above.

For each row $\mathbf{x} \in \mathbb{R}^{10}$ we fit least squares coefficients

$$C \;=\; \arg\min_{C\in\mathbb{R}^6} \; \|x - FC\|_2^2$$

and quantify goodness-of-fit using coefficient of determination

$$R^2(x) \;=\; 1 - \frac{\|x - FC\|_2^2}{\|x - \bar{x}\|_2^2},$$

We report the median R^2 over the set of rows in each X (i.e., over d_{mlp} rows for $\widehat{W_{out}}$, d rows for $\mathcal{E}$, and over batch examples for $\mathbf{h}^L$.

In Table 1 we observe strong fits: the per-row medians lie between 0.85 and 0.99, indicating that a six-dimensional trigonometric basis over digits captures the vast majority of variance:

We can extend the Fourier bases to include additional terms, for $k = 3, 4$, which forms a 8 dimensional basis (excluding sine terms for $k = 0, 5$), which leads to perfect R^2 fits.

Table 1: Median R^2 of Fourier fits over digits ($n = 0 \dots 9$).

Object	Fourier Basis	Rows aggregated	Median R^2
$\mathcal{E}$	$k = 0, 1, 2, 5$	d_{model}	0.84
MLP W_{out} weights	$k = 0, 1, 2, 5$	d_{mlp}	0.95
$\mathbf{h}^L$	$k = 0, 1, 2, 5$	batch examples	0.99
$\mathcal{E}$	$k = 0, 1, 2, 3, 4, 5$	d_{model}	1
MLP W_{out} weights	$k = 0, 1, 2, 3, 4, 5$	d_{mlp}	1
$\mathbf{h}^L$	$k = 0, 1, 2, 3, 4, 5$	batch examples	1

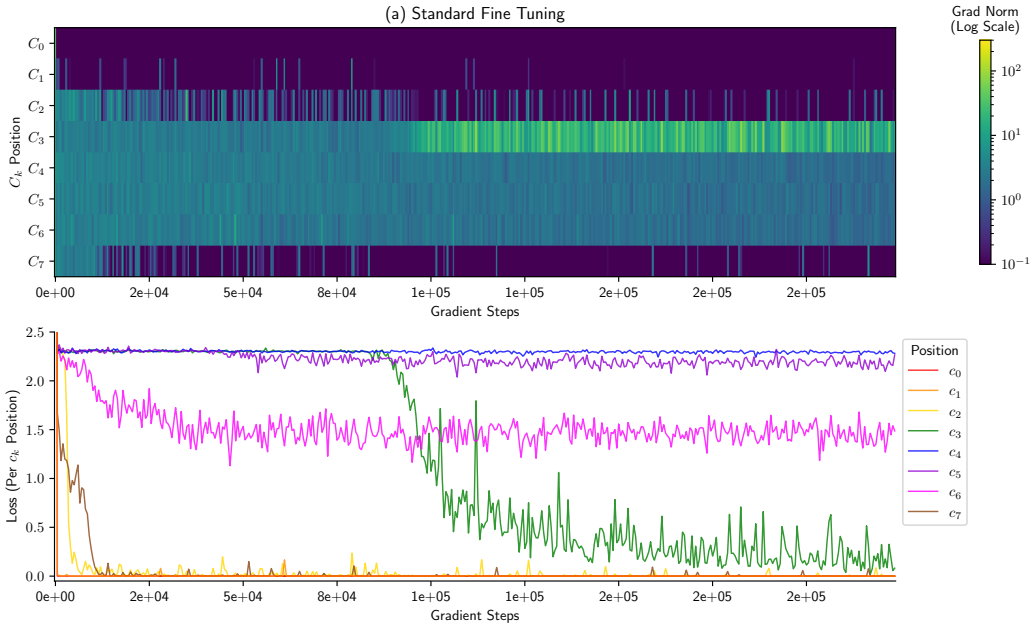

Figure 9: **Gradients and loss per token for a 12-layer model.**

C Per Token Gradients and Losses: 12-Layer Model

Even with a larger 12 layer model, the model fails to learn the right long-range dependencies. Figure 9 displays the results – we see the similar patterns as the 2-layer model, in which middle digits never receive the right gradients and loss does not drop.

D Attention Patterns of All Models

In Section 3.3, we illustrate how a binary tree is constructed for c_2 in the ICoT model. In Figure 10, we present the attention patterns for all digits across the three models, from with attention trees can be derived for the ICoT model for each solution token c_i.

E LLM Usage

We used LLMs to proof read our draft and polish our notations.

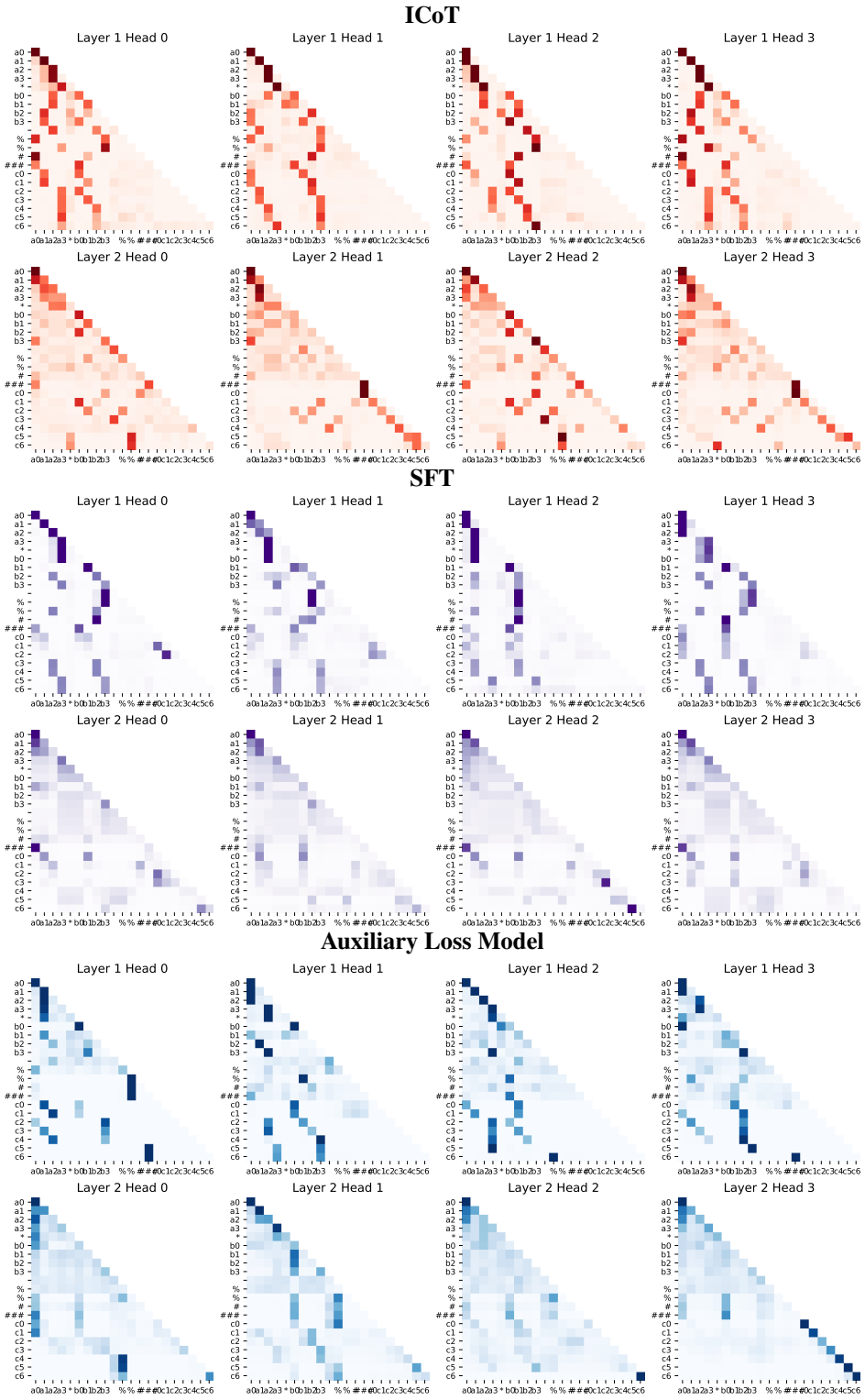

Figure 10: Attention patterns of ICoT, standard fine-tuned model, and the model trained with auxiliary loss