ETH Zürich - Deep Learning class 2025 - Project Report

Making Arithmetic Transformers Generalizable and Interpretable

Riddhi Barbhaiya Zhaorui Gong Jakub Kaššák Yiqu Yang

Abstract

Understanding how Transformers acquire algorithmic reasoning abilities and when such behaviors generalize remains a central challenge. This project studies the conditions under which arithmetic Transformers can generalize across both digit length and number of operands. We first conduct a systematic study of two-operand multi-digit addition, comparing one-dimensional and two-dimensional positional representations, different embedding mechanisms, and data representations. Building on these findings, we scale to multi-operand addition using a scratchpad-based formulation with intermediate sums. We find that even a small Transformer model can generalize nearly perfectly to sequences with two to three times longer digit lengths and number of operands. Finally, mechanistic analysis of attention patterns and tuned-lens probes reveals how arithmetic computation is decomposed across layers. [1]

1. Introduction

Length generalization – extrapolating to longer inputs than seen during training – remains a core challenge for transformers on algorithmic tasks. Addition is a canonical case: models often achieve perfect training accuracy but fail at longer lengths or more operands. In our report, we show techniques how to achieve great generalization performance.

2. Two-Operand Addition

This section describes the data representation, positional encoding schemes, and embedding mechanisms used for two-operand addition. We begin by defining the task and data representation, then describe the model architecture and training setup, and finally present the experimental results.

Task Definition. We consider the addition of two numbers with at most L digits. The input operands are represented in most-significant-digit (MSD) order, while the output sum is represented in least-significant-digit (LSD) order. The resulting input sequence is

$$\mathbf{x} = [a_L, \ldots, a_1, +, b_L, \ldots, b_1, =, s_1, \ldots, s_{L+1}]$$

where $a_i, b_i \in \{0, \ldots, 9\}$ denote digit values and the additional output digit s_{L+1} captures a possible final carry.

Implicit vs Explicit Carry Representations. We investigated two ways of representing the final sum, implicit and explicit. Implicit carry simply the represents the solution as you would expect($s_i \in \{0, \ldots, 9\}$, e.g $[3, 8, +, 6, 7, =, 5, 0, 1]$). Explicit carry carry represents each digit of the solution with a 1 if there was a carry that took place ($(s_i \in \{0, \ldots, 19\}$ e.g. $[3, 8, +, 6, 7, =, 15, 10, 1]$).

Positional Encodings. We define the following position encoding schemes:
1. Sequence-Index (SI): Sequential indices $p_{x_i} = L - i$.
2. Digit Offset (DO):

$$p_t = \begin{cases} i & \text{if token } t = a_i \vee t = b_i \vee t = s_i \\ 0 & \text{otherwise} \end{cases}$$

DO encoding ensures digits with the same significance share the same position encoding.

3. Digit Offset + Operand Role (DO+OR): – 2D encoding (digit offset, operand) where operand $\in \{1, 2, 3\}$ represents operand role (1st addend, 2nd addend, result). We encode the second (operand dimension) of token '+' with value 2 and '=' with value 3 . This representation is inspired by (Cho et al., 2025) and an of the same idea for multiple operands is presented in Figure 2.

Positional Embeddings.
1. Learned APE: Additive learned embeddings, formally $\mathbf{h}_t^{(0)} = \mathbf{x}_t + \mathbf{Emb}[p_t]$.
2. 2D APE: Learned APE with separate lookup tables for operand and digit: $\mathbf{x}_t + \mathbf{Emb}_{\text{operand}}[r_t] + \mathbf{Emb}_{\text{digit}}[p_t]$.
3. RoPE: A multiplicative scheme that rotates query and key vectors to encode positions, achieving shift invariance, thus converting an absolute encoding to a relative one (Su et al., 2023).
4. 2D Mixed: RoPE applied to digit dimension (captures

[1] Our code and data are available at https://github.com/jakub-kassak/dl-addition-transformer.

relative digit offsets) + learned APE on operand dimension (encodes operand membership).

2.1. Experimental Methodology

Data Sampling. Training batches are generated as follows. We first sample per operand length $L \sim Uniform(1, L_{max})$ where $L_{max} \leq 10$. Next, we sample digits $a_i, b_i \sim Uniform(0, ..., 9)$ independently. Finally, we compute sums with appropriate carry representation.

Model Architecture and Training. We conduct experiments with three architectures. The small models have 1 or 2 attention heads and 2 layers with about 2.2 million parameters. The medium model consists of 4 attention heads and 6 layers with 11.6 million parameters. The models are trained up to operands with 10 digits. No dropout is used in the two-operand training. See full experimental configurations (Table 1) and hyperparameters (Table 2) in appendix. We also investigated the impact of curriculum learning, starting with problems with smaller lengths and slowly increasing. Due to either mixed or negative, we didn't use it in the multi-operand setting. For more information, we refer the reader to the appendix.

2.2. Results

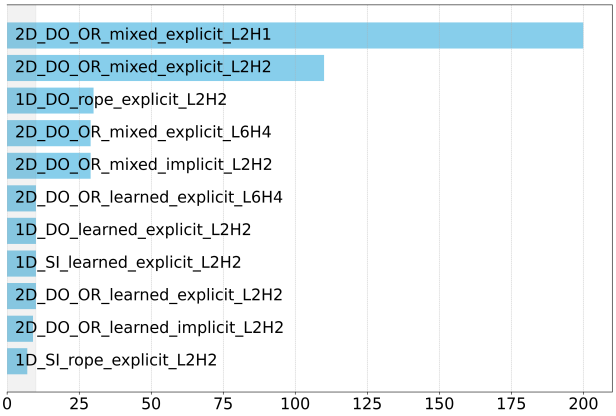


Figure 1. Length generalization performance of different positional encoding methods on 2-digit addition. The bars represent the maximal measured digit length on which the model achieved seq. accuracy $\geq 90\%$. Models were trained on sequences up to $L = 10$ digits (grey shaded region) and validated on lengths up to $L = 200$.

The model using the DO+OR positional encoding, mixed positional embedding, explicit carry representation, and a 2-head 1-layer architecture achieves near-perfect extrapolation to almost 200 digits, exhibiting no measurable degradation within the evaluation range (full results in Table 3). In contrast, the second-best model (2-head 2-layer architecture) generalizes reliably only up to approximately 115 digits.

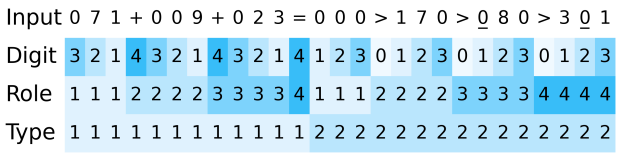


Figure 2. Input sequence with tri-level position encoding for multi-operand addition. Tokens with addition into next step are marked with underline.

This gap suggests that head dimension may be more important for length generalization than overall model size.

Across all experiments, strong length generalization is observed when a two-dimensional positional representation is employed. One-dimensional encodings with the exception of DO combined with RoPE consistently fail to extrapolate beyond the training regime. Among two-dimensional models, the mixed embedding scheme outperforms learned absolute embeddings, smaller architectures generalize better than larger ones, and explicit carry representations substantially improve robustness.

3. Scaling to Multi-Operand Addition

Building on the insights obtained from two-operand addition, we extend our framework to the more challenging setting of multi-operand addition. This task requires generalization not only across digit length but also across the number of operands, significantly increasing the complexity of the underlying computation. To support this extension, we introduce a scratchpad-based formulation, modify the data sampling procedure, and augment the positional representation to capture additional structural information.

3.1. Preliminaries

Scratchpad Design. We adopt the scratchpad formulation proposed by Cho et al. (Cho et al., 2025), which exposes intermediate computation steps to the model. The input sequence is given by

$$[X_1, +, \ldots, +, X_n, =, S_0, >, \ldots, >, S_n, \#],$$

where $X_i = [X_{(i,L)}, \ldots, X_{(i,1)}]$ denote the input operands in most-significant-digit (MSD) order, zero-padded to a common length. The scratchpad consists of cumulative intermediate sums $S_i = [S_{(i,1)}, \ldots, S_{(i,L)}] = \sum_{j=1}^{i} X_j$, represented in least-significant-digit (LSD) order, with $S_0 = 0$. The final sum S_n is interpreted as the output of the computation.

Position Encoding. Following Cho et al. (Cho et al., 2025) and the findings from our two-operand experiments, we design a tri-level positional representation which can be

seen in 2. The first dimension represents the digit significance, the second aligns n-th operand with n-th sum, and the third encodes the type of the computational region (input or scratchpad) of each token. An important note is that the smaller '>' tokens in the scratchpad are encoded together with the subsequent sum. In our first implementation, we made a mistake in encoding them into the previous, which resulted in no generalization across length. The explanation is that the '>' token makes prediction of the least significant digit in the output (recall that transformers are trained for autoregressive generation where the training sample output is input shifted by one). If the smaller token is encoded in the next block it will always have coordinates $(0, i, 2)$, whereas if it's in the previous block, the coordinates will be $(max\_len, i, 2)$.

3.2. Experimental Methodology

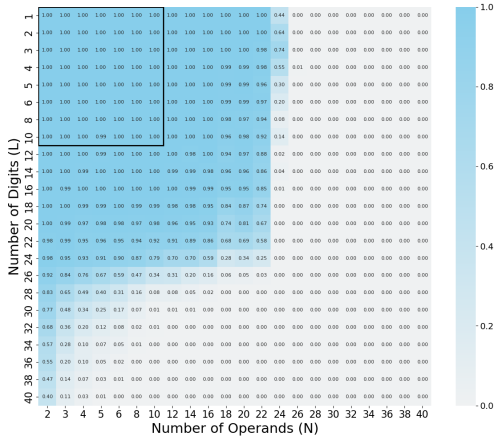
Data Sampling. We denote a training dataset by $S([l_{\min}, l_{\max}], [o_{\min}, o_{\max}])$, which consists of addition problems where each operand has between $l_{\min}$ and $l_{\max}$ digits, and the number of operands ranges from $o_{\min}$ to $o_{\max}$. Training samples are generated on the fly in batches. Each batch is constructed by concatenating eight independently generated mini-batches. For each mini-batch, the number of operands is sampled uniformly from $[o_{\min}, o_{\max}]$, the operand length is sampled uniformly from $[l_{\min}, l_{\max}]$, and all digits are drawn independently from $\{0, \ldots, 9\}$. Since mini-batches may differ in sequence length, each sequence is padded with at least one end-of-sequence token #. After padding, the corresponding scratchpad region is computed and appended to the input sequence. We denote this by $T(l, o) := S([l, l], [o, o])$. For validation, we evaluate generalization across both dimensions by organizing test data into a 2D grid over digit length and operand count.

Architecture & Hyperparameters. We started experimenting with multiple operands before we formed our head dimension hypothesis, the most promising architecture was the one with only 2 layers and 2 heads. We trained two models M_2 and M_{15} with 2M and 15.5M parameters respectively. The dataset for the smaller model was $S([1, 10], [2, 10])$. For the larger model, we decided to also explore generalization on shorter sequences, therefore training on the dataset $S([5, 10], [5, 10])$. See full hyperparameters for (Section 5.3 Section 5.3).

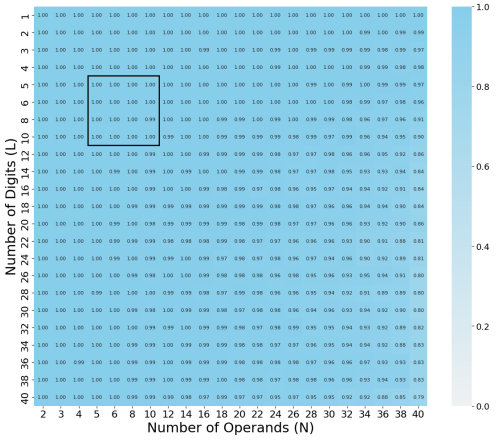
Dropout. While in the two-operand setting we decided not to use dropout, during the training of multi-operand addition a smaller dropout 0.2 helped with the learning process.

Checkpointing. During training, generalization performance showed spiky development. To address this, model checkpoints were saved after every epoch. We report results for the checkpoint with the best generalization.[2]

[2]We acknowledge that selecting the best checkpoint based on generalization does not follow the strict train/validation separation. However, our in training evaluation is limited in both operand length and count, and all learning signals come exclusively from the training set. This procedure ensures that model updates are unbiased while still allowing us to reliably capture the architecture's ability to generalize to longer or more complex addition problems.



(a)



(b)

Figure 3. Exact sequence accuracy validation results. The training ranges are marked by a black box. (a) M_2 generalization beyond the training range. (b) M_{15} exhibits strong extrapolation to both smaller and significantly larger digit lengths and operand counts.

3.3. Results

With scratchpad, operand–digit–region representation and explicit carries, the small model M_2 generalizes to ≈20 digits and ≈22 operands (Figure 3); larger model M_{15} attains non-trivial accuracy (79%) up to 40 operands with 40 digits each (Figure 3). These problems are extremely difficult, as the model must accurately predict 1849 consecutive tokens to solve the problem correctly. The larger model also achieves perfect out-of-distribution performance on smaller tasks.

4. Mechanistic Analysis of Multi-Operand Transformer

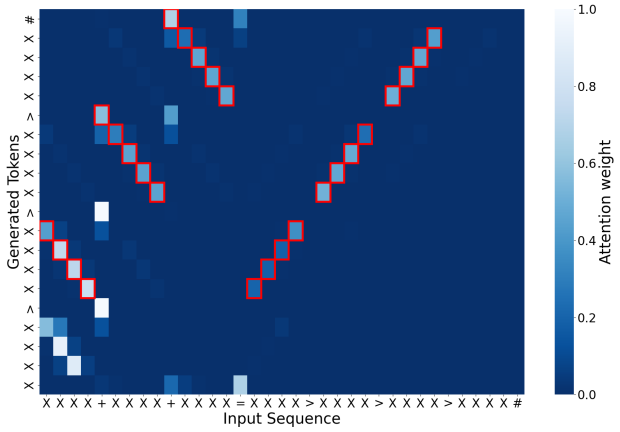


Figure 4. Average attention patterns for M_{15} model. The staircase is apparent in layer 0, indicating that the model focuses on the significant digits.

Attention Patterns & Head Specialization. We analyzed attention patterns of both models, given their similarity we present in figure Figure 5 we present only results from model M_{15}. We can observe that the attention patterns the Heads in the Layer 0 form a distinct mirrored staircase pattern – i.e. token $S_{i,j}$ attends to digits $X_{i,j}$ and $S_{i-1,j}$. In the figure Figure 5 we denote these coordinates with red rectangles. This led us first to believe that the zero layer computes base addition while the first layer enriches the prediction to propagated carry. However we observe that Layer 1 displaces it's attention mass onto multiple tokens and only a small amount is placed on tokens which could actually contain the necessary information, namely on the tokens $S_{(i-1,j-1)}$ (the previous digit of previous sum) and $X_{(i,j-1)}$ (the previous tokens of currently operand). We propose the following explanation – there is no need to transfer carry information, because the model token making that prediction already contains that information when we use the special carry tokens $[10-19]$. Therefore, the model needs to only attend to the two digits that will be added, and we believe that the heads in the first layer are not important for the prediction. Attention patterns for all heads and layers are in the appendix (Figure 10)

Tuned Lens. To validate this theory, we inspect the outputs of intermediate layers applying the tuned lens proposed by Belrose et al. (2025). The tuned lens trains an affine translator from the output of the intermediate layer into the unembedding space. This allows us to obtain an estimate of the prediction at that layer. We measured the performance on the dataset $T(20, 20)$ on 1000 samples. The translated zero layer output performs comparable to entire model – the recorded seq. accuracy was $\approx 100\%$.

No-Carry Models. To further test whether the model relies on an explicit special token representation in the presence of carry, we train a variant $M_2^{\text{no-carry}}$ in which the augmented digit tokens $[10-19]$ are not used, forcing the model to infer carries from context. This modification leads to a clear qualitative and quantitative shift in behavior. First, the attention patterns in the second layer become sharply concentrated on the immediately less significant digits, consistent with the need to explicitly recover carry information from prior positions. Second, mechanistic probes support a layer-wise decomposition of the computation: while the original model M_2 exhibits perfect Tuned Lens accuracy at layer 0 on dataset $T(20, 20)$, the no-carry variant achieves only 4% already on much easier dataset $T(5, 5)$. We conclude that in the absence of explicit carry tokens, carry handling must be reconstructed through attention to the less significant positions, effectively pushing carry propagation into a deeper layer of the network.

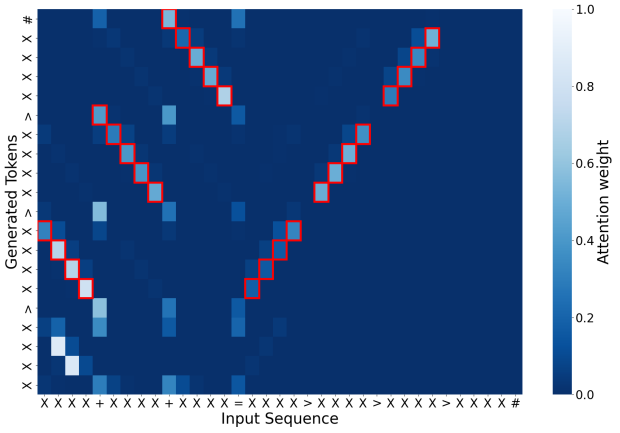


Figure 5. Average attention patterns for the model trained without explicit carry representations shows strong attention to immediately less significant digits.

5. Conclusion & Limitations

We reproduce prior results on transformer-based addition and show that they can be substantially improved. Integrating the idea of a structured scratchpad and operand–digit positional coupling with RoPE for relative positions, and explicit carry tokens yields models that are more accurate, smaller, and much faster to train. Moreover the we provide important insights into how our model work. While limited to decimal addition and specific architectures, our findings highlight the importance of principled positional design for both performance and understanding.

References

Bai, X., Pres, I., Deng, Y., Tan, C., Shieber, S., Viégas, F., Wattenberg, M., and Lee, A. Why can't transformers learn multiplication? reverse-engineering reveals long-range dependency pitfalls. *arXiv preprint arXiv:2510.00184*, 2025.

Belrose, N., Ostrovsky, I., McKinney, L., Furman, Z., Smith, L., Halawi, D., Biderman, S., and Steinhardt, J. Eliciting latent predictions from transformers with the tuned lens, 2025. URL https://arxiv.org/abs/2303.08112.

Bengio, Y., Louradour, J., Collobert, R., and Weston, J. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pp. 41–48, 2009.

Cho, H., Cha, J., Awasthi, P., Bhojanapalli, S., Gupta, A., and Yun, C. Position coupling: Improving length generalization of arithmetic transformers using task structure. In Langley, P. (ed.), *Advances in Neural Information Processing Systems, volume 37 (NeurIPS 2024)*, pp. 22233–22315, 2024.

Cho, H., Cha, J., Bhojanapalli, S., and Yun, C. Arithmetic transformers can length-generalize in both operand length and count. In *Proceedings of the Twelfth International Conference on Learning Representations (ICLR)*. PMLR, 2025. To appear.

Giannoulis, P., Pantis, Y., and Tzamos, C. Teaching transformers to solve combinatorial problems through efficient trial & error, 2025. URL https://arxiv.org/abs/2509.22023.

Su, J., Lu, Y., Pan, S., Murtadha, A., Wen, B., and Liu, Y. Roformer: Enhanced transformer with rotary position embedding, 2023. URL https://arxiv.org/abs/2104.09864.

Appendix

5.1. Two-Operand

Common Hyperparameters across models

Table 1. Model configurations. PC = Position Coupling.

Name	Dim	Embedding	Encoding	Encoding Type	Carry	$\lvert\Sigma\rvert$
1D-APE-DO	1	APE	DO	APE (PC)	Implicit	13
1D-RoPE-SI	1	RoPE	SI	Rotary PE (Abs.)	Implicit	13
1D-RoPE-DO	1	RoPE	DO	Rotary PE (PC)	Implicit	13
2D-APE	2	APE	DO + OR	APE (role + PC)	Explicit	23
2D-Mixed	2	Mixed	DO + OR	APE (role) + RoPE (PC)	Explicit	23
2D-Mixed-NC	2	Mixed	DO + OR	APE (role) + RoPE (PC)	Implicit	13

Table 2. Common hyperparameters.

Hyperparameter	Value/Range
Embedding dimension (d)	$\{256, 384\}$
Attention heads (h)	$\{2, 4\}$
Layers (ℓ)	$\{2, 4, 6\}$
Feedforward dim	$4d$
Dropout	0.0 (two-operand only)
Optimizer	AdamW
LR scheduler	OneCycleLR
$\text{lr}_{\max}$	$\{5 \times 10^{-4}, 8 \times 10^{-4}\}$
Batch size	256
Gradient clipping	1.0
Training steps	8K–20K

Table 3. Generalization performance across encoding schemes and model sizes. All models trained on $L \leq 10$. In-Dist Acc: average accuracy on $L \in [1, 10]$; Gen Acc: average accuracy on $L \in [11, 60]$; Breakdown Point: largest L with $\geq 90\%$ accuracy.

Model	In-Dist Acc	Gen Acc	L_{90}
1D Encodings (No Explicit Carries)			
1D-APE-DO	83.6%	5.8%	3
1D-RoPE-SI	6.3%	0.0%	1
1D-RoPE-DO	64.0%	16.0%	1
2D Encodings: Learned APE (Explicit Carries)			
2D-APE (2H2L)	100%	32.5%	20
2D Encodings: Mixed (No Explicit Carries)			
2D-Mixed-NoCarry (2H2L)	100%	82.0%	42
2D-Mixed-NoCarry (4H6L)	100%	15.4%	18
2D Encodings: Mixed (Explicit Carries)			
2D-Mixed (2H2L)	100%	**99.3%**	**>60**
2D-Mixed (2H4L)	100%	79.7%	44
2D-Mixed (4H2L)	100%	51.9%	25
2D-Mixed (4H6L)	100%	28.2%	23

5.2. Curriculum learning

Curriculum learning is a training paradigm in which the data distribution presented to the model changes over time according to a predefined notion of sample difficulty, rather than remaining fixed throughout training. The concept was introduced by Bengio et al. (Bengio et al., 2009), who showed that presenting training examples from easier to harder can improve optimization and generalization in non-convex learning problems.

5.2.1. CURRICULUM LEARNING (6 HEADS/4 LAYERS)

In this work, we investigate the effect of curriculum learning on the generalization ability of models for addition. We use a Transformer (6L4H), trained with a learning rate of 6×10^{-4}, and 1k steps per epoch,using 2D APE and explicit carry.

As shown in Figure 6, we report post-training evaluation results obtained using 500 samples per digit with an offset range of 30 and greedy decoding. When each epoch is trained on a fixed sequence length (i.e., `min_train_digits` = `max_train_digits`), the ascending curriculum yields marginally better generalization than the descending curriculum. Nevertheless, both curricula are consistently outperformed by training directly on the full target distribution (e.g., always using digit length 10).

A closer inspection reveals asymmetric forgetting behaviors between the two curricula. During ascending training, performance on single-digit addition degrades steadily as training progresses, whereas under the descending curriculum, accuracy on initial training length (10-digit addition) remains largely stable throughout training.

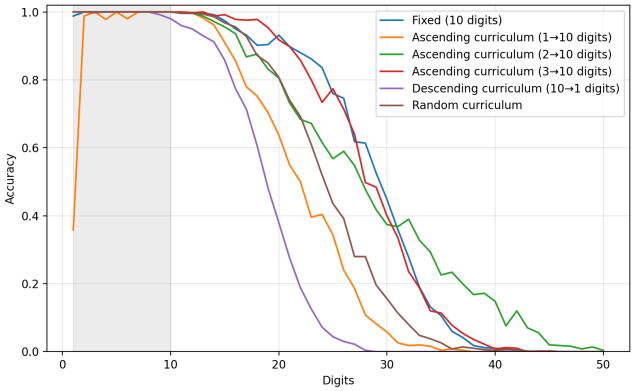


Figure 6. Generalization performance across digit lengths for different curriculum strategies. The shaded region indicates the training range (digits 1–10) in 6L4H transformer.

Possible explanation. The catastrophic forgetting observed under the ascending curriculum can be interpreted as a form of *carry overfitting*. In multi-digit addition, the computation at digit i follows

$$c_i = (a_i + b_i + \text{carry}_{i-1}) \bmod 10$$

where carry_{i-1} is propagated from the lower digit. Single-digit addition corresponds to the degenerate case with no incoming carry ($\text{carry}_{i-1} = 0$). If the model learns a strongly activated carry pathway with weak gating(degenerated to always-on), spurious activation of the carry mechanism becomes a natural failure mode.

After extensive training on multi-digit addition, the model may incorrectly activate the carry-propagation mechanism even when no incoming carry is present, leading to degraded performance on single-digit problems. Paradoxically, this behavior also indicates successful acquisition of the carry mechanism: the carry signal becomes sufficiently dominant to be applied outside its valid regime. In some cases, the carry pathway is not only spuriously activated but also misaligned in its readout, producing structurally valid but numerically incorrect outputs (e.g., $7 + 7 \rightarrow 148$; see Figure 7). This suggests that single-digit and multi-digit addition rely on distinct computational regimes, and that curriculum order influences the severity

of interference. While the descending curriculum also exhibits forgetting, its effect is substantially weaker than that of the ascending curriculum.

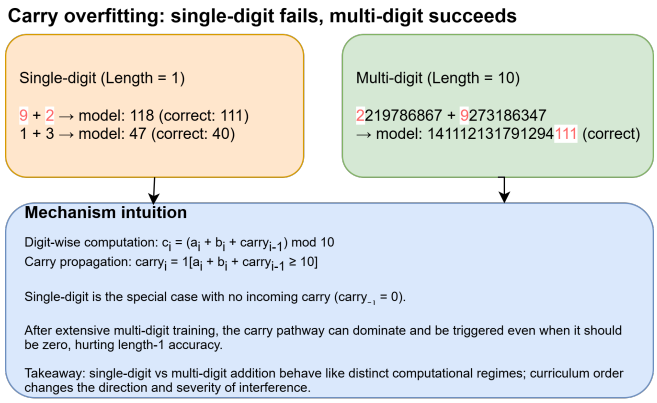


Figure 7. Illustrative example of carry overfitting.

5.2.2. CURRICULUM LEARNING (2 HEADS/2 LAYERS)

In this work, we investigate the effect of curriculum learning on the generalization ability of models for addition. We use a Transformer (2L2H), trained with a learning rate of 6×10^{-4}, and 1k steps per epoch,using 2D APE and explicit carry.

Figure 8 reports post-training evaluation results obtained using 500 samples per digit, an offset range of 30, and greedy decoding. When each epoch is trained on a fixed sequence length (i.e., `min_train_digits` = `max_train_digits`), training directly on the target length (Fixed, 10 digits) achieves near-perfect accuracy within the training range but exhibits a rapid degradation beyond moderate lengths, indicating limited extrapolation. Ascending curricula partially alleviate this issue by gradually introducing longer sequences, with earlier exposure to single-digit addition (1→10) yielding stronger extrapolation than curricula starting from longer digits (2→10). In contrast, the descending curriculum (10→1) suffers an abrupt performance collapse immediately outside the training range, suggesting severe interference between early multi-digit learning and later single-digit adaptation. Notably, the random curriculum yields the most stable performance across lengths, implying that mixing difficulties during training discourages over-specialization to any single computational regime.

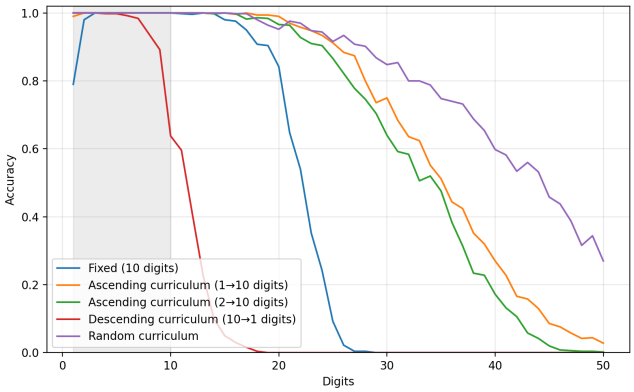


Figure 8. Generalization performance across digit lengths for different curriculum strategies. The shaded region indicates the training range (digits 1–10) in 2L2H transformer.

Possible explanation. These results suggest that curriculum learning affects generalization in addition not simply by ordering examples by difficulty, but by shaping which intermediate computational regimes dominate during training. Curricula that concentrate on a single regime—such as fixed-length or strictly ordered schedules—encourage over-specialization and lead to limited extrapolation. In contrast, mixing difficulties promotes more balanced internal representations and yields more robust generalization across input lengths.

Importantly, fixed-length training can also degrade accuracy on single-digit addition, despite such inputs being strictly easier. Single-digit addition constitutes a degenerate regime with no incoming carry, whereas longer sequences consistently activate carry-related mechanisms. Over-specialization to these mechanisms can therefore lead to spurious carry activation on single-digit inputs, further supporting the view that curriculum learning shapes generalization by modulating which computational regimes dominate during training.

5.2.3. WHY DO SMALL AND LARGE MODELS BEHAVE DIFFERENTLY?

Summary.

- **Capacity and regime separation.** Larger models can tolerate curriculum bias due to sufficient representational capacity, enabling them to isolate multiple computational regimes, whereas smaller models are more vulnerable to over-specialization.
- **Effect of fixed-length training.** Training exclusively on the target length (all-10) aligns larger models with the dominant regime without suppressing alternatives, but in smaller models it causes carry-related mechanisms to dominate the limited representation space, resulting in poor generalization.
- **Role of mixture curricula.** Fixed or ordered curricula amplify dominant mechanisms, while mixture curricula

implicitly regularize representation usage by preventing any single computational pathway from monopolizing the model.

- **When smaller models perform better.** Under such regularization, smaller models are forced to adopt simpler and more robust algorithmic strategies, which can in some cases yield stronger extrapolation than larger but more flexible models.

Hyperparameter	Value
Architecture	Decoder-only Transformer
Number of Layers	2
Number of Attention Heads	2
Embedding Dimension	256
Dimension per Head	128
Hidden Width of Feed-forward Layer	1024
Hidden Depth of Feed-forward Layer	1
Activation Function of Feed-forward Layer	ReLU
Normalization Layer	LayerNorm
Normalization Layer Position	PreNorm
Trainable Parameter Count	2M
Dropout	0.2
Training Steps	5,000
Batch Size	256
Optimizer	AdamW
Learning Rate Scheduler	OneCycleLR
Maximum LR	0.0003
Percentage of Steps for Increasing LR (pct_start)	10%
Annealing Strategy	Cosine
Final LR Factor	1/10,000 of max LR
Training Dataset	$S(1, 10, 2, 10)$
Evaluation Dataset Size	
Device	NVidia RTX 5060 Ti
Training Time	$<$ 30 min

Table 4. Hyperparameter summary for the multi-operand addition - small model

Hyperparameter	Value
Architecture	Decoder-only Transformer
Number of Layers	2
Number of Attention Heads	2
Embedding Dimension	512
Dimension per Head	256
Hidden Width of Feed-forward Layer	2048
Hidden Depth of Feed-forward Layer	2
Activation Function of Feed-forward Layer	ReLU
Normalization Layer	LayerNorm
Normalization Layer Position	PreNorm
Trainable Parameter Count	15.5M
Dropout	0.2
Training Steps	1,700
Batch Size	256
Optimizer	AdamW
Learning Rate Scheduler	OneCycleLR
Maximum LR	0.0001
Percentage of Steps for Increasing LR (pct_start)	10%
Annealing Strategy	Cosine
Final LR Factor	1/10,000 of max LR
Training Dataset	$S(5, 10, 5, 10)$
Evaluation Dataset Size	
Device	NVidia RTX 5060 Ti
Training Time	1 h

Table 5. Hyperparameter summary for the multi-operand addition - big model

5.3. Multi-Operand

5.4. "Mechanistic Interpretability"

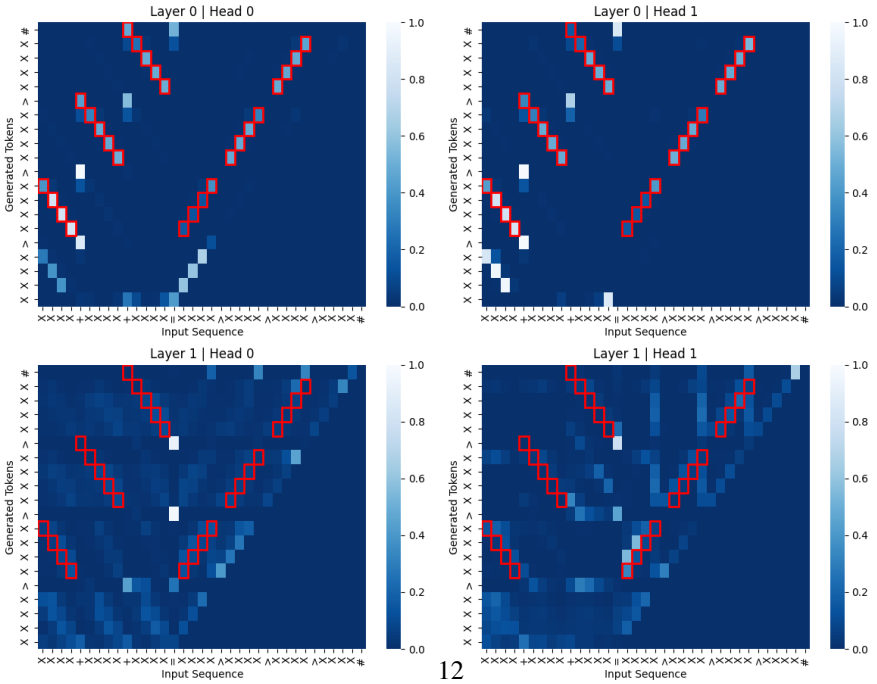


Figure 9. Average attention patterns for M_2 model. The staircase pattern can be seen as apparent in layer 0 indicating that the model focuses on the significant digits.

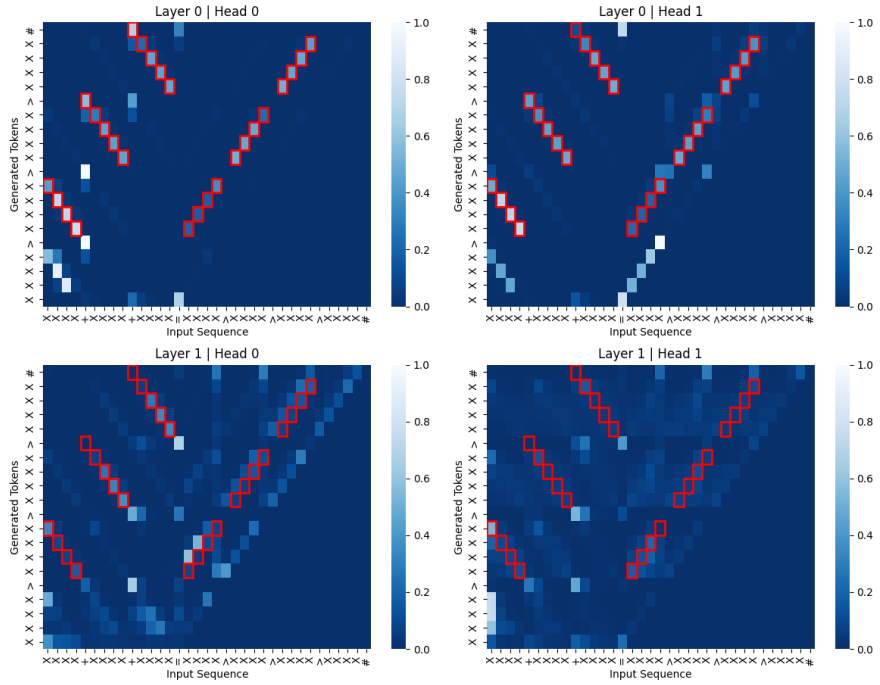


Figure 10. Average attention patterns for M_{15} model. The staircase pattern can be seen as apparent in layer 0 indicating that the model focuses on the significant digits.

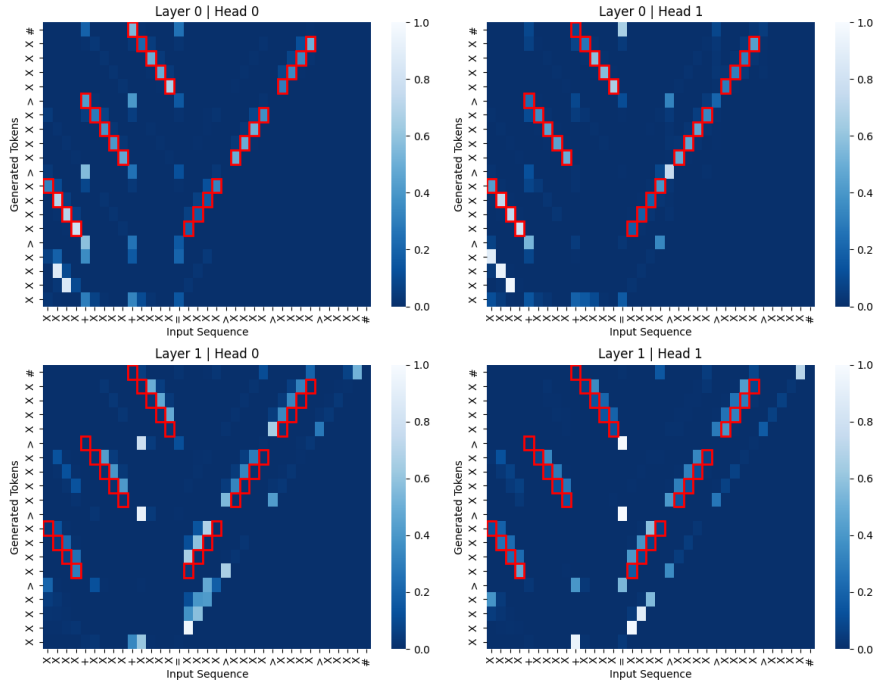


Figure 11. Average attention patterns for the model trained without explicit carry representations shows strong attention to immediately less significant digits.